ERES 2017

1ª Escola Regional de Engenharia de Software

18 - 20 / Out UNIPAMPA, Alegrete - RS

ANAIS

I ERES
Anais da 1º Escola Regional Engenharia de Software (ERES 2017)

Universidade Federal do Pampa, UNIPAMPA - Alegrete-RS, outubro de 2017

March 5, 2018

E74a Escola Regional de Engenharia de Software - ERES *(1.: 18-20 out. 2017 : Alegrete, RS)*
Anais da 1ª Escola Regional de Engenharia de Software [recurso eletrônico] / Organização: Universidade Federal do Pampa - UNIPAMPA - Campus Alegrete e Instituto Federal Farroupilha - IFF - Campus Alegrete. – Alegrete: Unipampa, 2018.
221p. ; il.

Anual
Realização: Sociedade Brasileira de Computação - SBC
ISSN 2594-9896

1. Engenharia de Software. 2. Ensino superior. 3. Computação. I. Universidade Federal do Pampa. II. Instituto Federal Farroupilha.
CDU: 004.41

Catalogação na fonte: Cátia Rosana Lemos de Araújo – CRB 10/1451
Biblioteca Campus Alegrete - UNIPAMPA

I ERES 2017

1º Escola Regional de Engenharia de Software - ERES 2017

COMISSÃO ORGANIZADORA

Coordenação Geral - Elder de Macedo Rodrigues (UNIPAMPA)
Coordenação Local - Maicon Bernardino da Silveira (UNIPAMPA)
Coordenador Financeiro - Gilleanes Thorwald Araújo Guedes (UNIPAMPA)
Coordenador do Fórum de Pós-graduação - Edson de Oliveira Junior (UEM)
Coordenador do Fórum de Graduação - Marcelo Hideki Yamaguti (PUCRS)

COMISSÃO EXECUTIVA DA ERES - CEERES

Elder de Macedo Rodrigues (UNIPAMPA)
Maicon Bernardino da Silveira (UNIPAMPA)
Rafael Alves Paes Oliveira (UTFPR)
Edson de Oliveira Junior (UEM)
Marcelo Hideki Yamaguti (PUCRS)
Ingrid Oliveira de Nunes (UFRGS) - Secretária

COMITÊ CIENTÍFICO

Comitê do Fórum de Pós-graduação

Aline Mello (UNIPAMPA)
Amanda Melo (UNIPAMPA)
Andre Endo (UTFPR)
Andréa Bordin (UNIPAMPA)
Andreia Malucelli (PUCPR)
Avelino Zorzo (PUCRS)
Claudio Schepke (UNIPAMPA)
Elder de Macedo Rodrigues (UNIPAMPA)
Fabiane Benitti (UFSC)
Giani Petri (UFSM)
Gilleanes Thorwald Guedes (UNIPAMPA)
Jean Cheiran (UNIPAMPA)
João Pablo S. da Silva (UNIPAMPA)
Lisandra Fontoura (UFSM)
Maicon Bernardino (UNIPAMPA)
Marcelo Hideki Yamaguti (PUCRS)
Ricardo Pereira e Silva (UFSC)
Sabrina Marczak (PUCRS)
Sheila Reinehr (PUCPR)
Silvia Vergilio (UFPR)

Comitê do Fórum de Graduação

Alessandra Costa Smolenaars Dutra (PUCRS)
Alexandre Cidral (UNIVILLE)

Ana Paula Terra Bacelo (PUCRS)
Érica F. Souza (UTFPR)
Daniel Antonio Callegari (PUCRS)
Edson Oliveira Junior (UEM)
Eduardo Kroth (UNISC)
Elder Macedo Rodrigues (UNIPAMPA)
Evandro Franzen (UNIVATES, UNISC)
Fabrício Pretto (UNIVATES)
Ingrid Oliveira de Nunes (UFRGS)
Jean F. P. Cheiran (UNIPAMPA)
João Pablo S. da Silva (UNIPAMPA)
Juliana Herbert (UFCSPA)
Lisandra Manzoni Fontoura (UFSM)
Marcelo Hideki Yamaguti (PUCRS)
Maria Claudete Schorr Wildner (UNIVATES)
Mouriac H. Diemer (UNIVATES)
Pablo Schoeffel (UDESC)
Rafael Alves Paes de Oliveira (UTFPR)
Ricardo Pereira e Silva (UFSC)
Thais Webber (UNISC)

REALIZAÇÃO

Sociedade Brasileira de Computação – SBC

ORGANIZAÇÃO

Universidade Federal do Pampa - UNIPAMPA - *Campus* Alegrete
Instituto Federal Farroupilha - IFF - *Campus* Alegrete

PATROCÍNIO

Sicredi
Zallpy
FAPERGS

APOIO

Conesul
DevPampa

Editorial

É com grande satisfação que apresentamos os artigos aceitos para a primeira edição da Escola Regional de Engenharia de Software (ERES), a qual compõem os anais do evento. A I ERES ocorreu de 18 a 20 de outubro de 2017, na cidade de Alegrete/RS, sob a organização da Universidade Federal do Pampa (UNIPAMPA) e Instituto Federal Farroupilha (IFF), ambos dos campi Alegrete.

A ERES é promovida anualmente pela Sociedade Brasileira de Computação (SBC) com o objetivo de disseminar o conhecimento e boas práticas em Engenharia de Software (ES) - tanto do ponto de vista profissional quanto acadêmico. A ERES 2017 é um espaço regional para que possam ser apresentados os resultados de pesquisas de graduação e pós-graduação e relatos de experiência na indústria. Além disso, possibilitará um ambiente natural para a discussão de abordagens no ensino-aprendizagem na ES nas instituições de ensino superior da Região Sul do Brasil.

Mantendo a tradição de outras escolas regionais, foram aceitas submissões de artigos em duas categorias: Fórum de Graduação e Fórum de Pós-graduação. Todos os artigos foram avaliados por pelo menos 3 membros do Comitê Científico. Além das sessões técnicas, a programação do evento contou com palestras e minicursos proferidas por pesquisadores de renome e profissionais da indústria da área de Engenharia de Software.

O Fórum de Graduação recebeu 39 submissões, das quais 19 foram aceitas para apresentação oral, o que representa 48,7% de taxa de aceitação. Por outro lado, o Fórum de Pós-graduação obteve 15 submissões, das quais 8 foram aceitas, o que representa 53,3% de taxa de aceitação.

Os Anais da I ERES representam o resultado do esforço coletivo de um grande número de pessoas. Agradecemos ao docentes, discentes e técnicos-administrativos da UNIPAMPA e IFF que trabalharam arduamente para garantir o bom andamento do evento. Gostaríamos de agradecer também aos membros do Comitê Científico que fizeram revisões de excelente qualidade. Um agradecimento especial aos professores Edson de Oliveira Junior (UEM) e Marcelo Hideki Yamaguti (PUCRS) na dedicação e empenho no gerenciamento e organização dos fóruns científicos. Por fim, agradecemos aos autores que submeteram seus trabalhos para a ERES 2017.

Prof. Dr. Elder Macedo Rodrigues – *Coordenador Geral*
Prof. Dr. Maicon Bernardino – *Coordenador Local*

Autorizo a reprodução parcial ou total desta obra, para fins acadêmicos, desde que citada a fonte

Índice/Programa em Sessões técnicas

Fórum de Graduação **1**

Sessão Técnica 01 – Tema: Processos de Software **1**

1 Desenvolvimento de Técnicas para Sumarização de Vídeos de Jogos Digitais: um projeto de pesquisa
Luciano Marchezan (UNIPAMPA), Elder Rodrigues (UNIPAMPA), Maicon Bernardino (UNIPAMPA), Marcelo Schmitt Laser (PUCRS) e Fernando de Lima (UNIPAMPA)

9 Atta Method: A Nature Based Approach
Miguel Ecar (UNIPAMPA)

17 Integrando Engenharia de Software e Interação Humano-Computador em um Processo de Desenvolvimento de Software para/com Crianças
Jéssica Brasil Pereira (UNIPAMPA) e Amanda Melo (UNIPAMPA)

25 Desenvolvimento de Jogos Digitais do Gênero RPG com Adolescentes
Leonardo Anschau (UNIPAMPA) e Amanda Melo (UNIPAMPA)

Sessão Técnica 02 – Tema: Engenharia de Requisitos & Linguagem Específica de Domínio **33**

33 Um Editor para a Linguagem de Especificação de Requisitos RELAX
Rafael Torres (UNIPAMPA), Luiz Paulo Franz (UNIPAMPA) e João Pablo S. da Silva (UNIPAMPA)

41 Multivocal Literature Review on User Story Models for COSMIC Sizing
Miguel Ecar (UNIPAMPA) e João Pablo S. da Silva (UNIPAMPA)

49 Proposta de Linguagem de Modelagem Gráfica com o Sirius: Uma Versão Open Source da DSL Canopus
João Carbonell (UNIPAMPA), Bruno Medeiros (UNIPAMPA), Pedro Henrique Silva (UNIPAMPA), Elder Rodrigues (UNIPAMPA) e Maicon Bernardino (UNIPAMPA)

57 Criando uma Linguagem Específica de Domínio usando o Xtext: Uma Opção Open Source para Implementar a Canopus
Aníbal Iung (UNIPAMPA), Luiz Paulo Franz (UNIPAMPA), Maicon Bernardino (UNIPAMPA), Jonnathan Lopes (UNIPAMPA) e Elder Rodrigues (UNIPAMPA)

Sessão Técnica 03 – Tema: Reutilização de Software & Evolução de Software **65**

65 Desenvolvimento de Software Baseado em Componentes Usando IDL
Yury Lima (UNIPAMPA), Giliardi Schmidt (UNIPAMPA), Elder Rodrigues (UNIPAMPA) e Maicon Bernardino (UNIPAMPA)

73 Uma Ferramenta para Sincronização de Conteúdos Produzidos pelo Software QGIS
Madson Verdi Junior (UNIPAMPA) e Claudio Schepke (UNIPAMPA)

81 Identificação de características de sistemas legados a partir da uma análise de conteúdo da literatura
Daniele Martins dos Santos (UNIPAMPA), Alex Severo Chervenski (UNIPAMPA) e Andréa S. Bordin (UNIPAMPA)

89 DevOps adoption in junior enterprise: an experience report of software development
Peterson Rodrigues (UNIPAMPA), Juliano Macedo (UNIPAMPA), Pedro França (UNIPAMPA), Luiz Paulo Franz (UNIPAMPA), João Pablo S. da Silva (UNIPAMPA) e Jean Cheiran (UNIPAMPA)

Sessão Técnica 04 – Tema: Educação em Engenharia de Software **97**

97 Resolução de Problemas no Curso de Engenharia de Software: Uma Experiência Envolvendo Extensão e Ensino
Jonnathan Lopes (UNIPAMPA), Gabriela Córdova Medeiros (UNIPAMPA), Dienefer Fialho dos Santos (UNIPAMPA) e Andréa S. Bordin (UNIPAMPA)

105 Percepção da Contribuição da AGES para o Aprendizado em Engenharia de Software
Alessandra Dutra (PUCRS), Azriel Majdenbaum (PUCRS), Cássio A.W. Trindade (PUCRS), Daniel Callegari (PUCRS) e Marcelo Yamaguti (PUCRS)

111 Experiência de uso do eXtreme Programming (XP) em disciplina de Engenharia de Software
Lis Ângela de Bortoli (IFRS)

119 Web-Atlas: Uma Ferramenta Web Para Modelagem de Características
Fernando de Lima (UNIPAMPA), Luciano Marchezan (UNIPAMPA), Marcelo Schmitt Laser (PUCRS), Elder Rodrigues (UNIPAMPA) e Maicon Bernardino (UNIPAMPA)

Sessão Técnica 05 – Tema: Verificação & Validação **127**

127 Aplicando Técnicas de V&V na Fase de Projeto do Modelo em Cascata
Naihara Amorim (UNIPAMPA) e Gilleanes Thorwald Guedes (UNIPAMPA)

135 Segurança na Web: análise black-box de scanners de vulnerabilidades
Isadora Ferrão (UNIPAMPA) e Diego Kreutz (UNIPAMPA)

143 SOFIA: Um Sistema de Suporte para as Inspeções de Software
Thiago Krug (UFSM), Juliano Pires (UNIPAMPA), Vitor Hugo dos Santos (UNIPAMPA) e João Pablo S. da Silva (UNIPAMPA)

Fórum de Pós-graduação **151**

Sessão Técnica 01 – Tema: Ecossistemas, Teste e Domain-Specific Languages **151**

151 Redes de Colaboração em Ecossistemas de Software - Uma Análise baseada em uma Revisão Longitudinal de Literatura
Caio Borges (PUCRS), David Guaspari Barreto (Pesquisador Independente), Sabrina Marczak (PUCRS), Cleidson de Souza (ITV e UFPA) e Fernando Figueira Filho (UFRN)

161 Aquila: uma Linguagem de Domínio Específico para Teste Baseado em Modelos para Projetos Ágeis
Aline Zanin (PUCRS) e Avelino Zorzo (PUCRS)

171 Utilização do Algoritmo de QSM para automação de testes a partir de cenários Gherkin
Aline Zanin (PUCRS), Henry Nunes (PUCRS), Avelino Zorzo (PUCRS) e Flávio Moreira de Oliveira (PUCRS)

181 Usa-DSL: Um Framework para Avaliação de Usabilidade de Linguagens Específicas de Domínio
Ildevana Rodrigues (PUCRS), Avelino Zorzo (PUCRS), Maicon Bernardino (UNIPAMPA), Elder Rodrigues (UNIPAMPA) e Márcia de Borba Campos (PUCRS)

Sessão Técnica 02 – Tema: Engenharia de Software Preditiva, Gamification, Fábrica de Software e Metodologias Ágeis **191**

191 Mapeamento Sistemático de gamificação em Engenharia de Software
Gabriel Belinazo (UFSM) e Lisandra Fontoura (UFSM)

201 `Learning by doing em Fábrica de Software: Relato de uma experiência no mestrado profissional em computação aplicada`
Alexandre Zanatta (UPF), Tainara Bernardi (UPF), Jucelia Beux (UPF), Daiana Biduski (UPF) e Ericles Bellei (UPF)

211 `On the Characterization of the SDET Role in Agile Projects: Expected Skills, Responsibilities, and Practices`
Carolina Toscani (PUCRS), Julia Couto (PUCRS), Josiane Kroll (PUCRS), Alessandra Dutra (PUCRS), Sabrina Marczak (PUCRS) e Rafael Prikladnicki (PUCRS)

Lista de Autores **221**

Towards a Generic Process for SPL Re-engineering

Luciano Marchezan[1], Elder Rodrigues[1], Maicon Bernardino[1], Marcelo Laser[2], Fernando Lima[1]

[1]Federal University of Pampa (UNIPAMPA)
Av. Tiarajú, 810, Ibirapuitã – Alegrete, RS – Brasil

[2]Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 – Partenon, 90619-900 – Porto Alegre – RS – Brazil

{lucianomarchezan94, marcelo.laser, fernando.fortlima}@gmail.com

elderrodrigues@unipampa.edu.br, bernardino@acm.org

Abstract. *Software Product Lines (SPL) are a well known solution to create reusable software products. Usually a set of product variants is analyzed and commonalities and variabilities are extracted. In this paper, we present our first step towards a generic process to extract those characteristics from product variants. Our process aims to have flexibility to be customizable for different scenarios. We applied a survey and obtained an early feedback which enables us to measure the impact and the contribution of our proposal.*

Resumo. *Linha de Produto de Software (LPS) é uma solução bem conhecida para se criar uma família de produtos de software. Geralmente, um conjunto de variantes de produto é analisado e suas variabilidades são extraídas. Neste trabalho, apresentamos um primeiro passo na direção de um processo genérico para extração dessas características de variantes de produto. Nosso processo busca uma flexibilidade que possibilita sua customização para diferentes cenários. Nós aplicamos um questionário e obtivemos um feedback inicial que nos permite medir o impacto e a contribuição da nossa proposta.*

1. Introduction

While many systems are being developed proactively as a Software Product Line (SPL) [Pohl et al. 2005], there are still those that emerge when a company has one or more software products and wants to create a software product line out of them [Van der Linden et al. 2007]. In this case, should they start from scratch or can they use those software products and transform then into a product line? If they choose to transform their products into a product line, how can they do it? We may find an answer to the latter question in the software product line re-engineering process. The goal of this process is to take a number of product variants and transform them into a SPL with the use of techniques, methods and tools [Assunção et al. 2017]. To the best of our knowledge, a solid and well-known generic process for performing the SPL re-engineering process has not been proposed yet. We find that this kind of process would be a relevant contribution for the Software Product Line Engineering (SPLE) [Van der Linden et al. 2007] field because it may reduce the complexity and effort of SPL re-engineering.

In accordance with the data we found regarding the need for a generic SPL re-engineering process, we have created a generic process for SPL re-engineering. The main

goal of our process is to be customizable and applicable in different scenarios to detect and extract features from a set of product variants. With our process, we intend to provide enough flexibility regarding artifacts, strategies and techniques used for feature retrieval. This flexibility is given by allowing the choice and combination of different techniques for feature retrieval based on information gathered during the process' execution. This information includes: team experiences and skills, domain engineering artifacts, requirements artifacts, product artifact types and extensions, and technologies used to develop the products. Before concluding and executing our process and conducting its evaluation, we sought to obtain feedback regarding the proposal. To this purpose, we have created a survey to assess the impact, contribution and originality of our proposal. We have sent this survey to researchers and professionals in the SPL and re-engineering field and analyzed the results to help us refine and improve our proposal.

The main objective of this paper is propose a generic process for SPL re-engineering. We also justify, explain and analyze the results of the survey we created to assess our proposal. With the survey, we have obtain a good degree of confidence about the impact and relevance of our process. The rest of this work is structured as follows: a background for SPLs and the related work are described in Section 2. Our process is showed in Section 3 and the results of the survey are discussed in Section 4. Lastly, the conclusions and future work will be presented in Section 5.

2. Background and Related Work

Software Product Line: To the best of our knowledge, the product line concept and its characteristics came from the automobile industry, introducing an important characteristics called variability [Pohl et al. 2005]. Variability refers to the flexibility, in other words, the ability to create artifacts to be reused among different products of the same family [Kang et al. 1990]. Some of the main benefits for implementing software product lines are: long-term cost reduction, significant quality improvement of the products and reduction of the time-to-market [Van der Linden et al. 2007]. The field dedicated to the study of this development of SPLs is Software Product Line Engineering.

Software Product Line Engineering (SPLE): Plenty of technologies are used in SPLE as facilitators, making the SPL creation process easier. From these technologies, we can highlight: component-based architecture, software project patterns and the object-oriented paradigm [Kang et al. 1990]. A model for Software Product Line Engineering was proposed by Van der Linden et al. (2007), it is divided between domain engineering and application engineering. Each activity of domain engineering, except product management, generates a high-level artifact that will be used in application engineering. These high-level artifacts are: reference requirements, reference architecture, domain realization mechanisms, and domain tests. They are later specialized in application artifacts: software requirements specifications, software architecture, source code, and test cases. A common practice in the SPL development is to get a set of product variants and extract their variabilities to create the SPL. This process is called Software Product Line Re-engineering [Assunção et al. 2017].

Software Product Line Re-engineering: The term re-engineering can be described as "the examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form" [Chikofsky and Cross 1990].

In the SPL context, re-engineering is used to transform a system, system family, or system variants into a software product line as illustrated in Figure 1. According to [Assunção et al. 2017], the SPL re-engineering process is composed of three main phases: detection, analysis and transformation. During the detection phase, the variability points and commonalities of the products are identified and extracted through the use of feature retrieval techniques. The second phase is analysis, where the discovered features are organized as a feature model. The last phase, transformation, is when artifacts linked with these features are managed and modified in order to create the SPL.

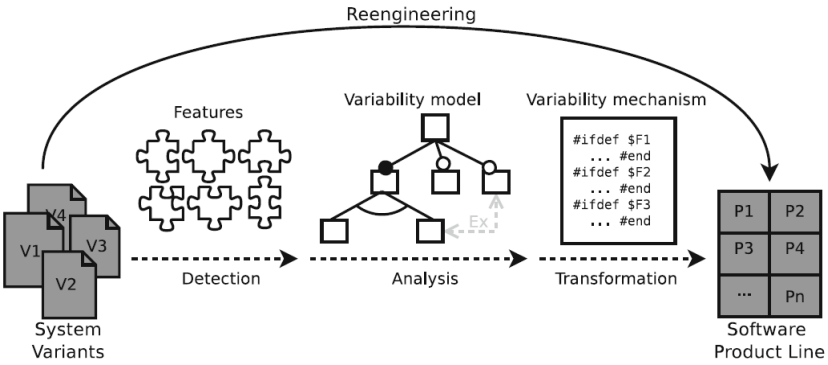


Figure 1. SPL re-engineering process [Assunção et al. 2017]

Related Work: We sought the state of the art of the SPL Re-engineering field considering three aspects of our work: re-engineering phases covered, retrieval techniques proposed or recommended, and the flexibility of the proposal. For the first topic, most of the works had their proposal focusing on the first two phases: detection and analysis [Stoermer and O'Brien 2001] [Rubin and Chechik 2012] [Eyal-Salman et al. 2012] [Damaševičius et al. 2012]. These works, however, are not concerned in offering different options for retrieval techniques. What is showed in those works is a static process for feature extraction and analysis. Regarding the retrieval techniques, we found works that proposed a combination of different techniques in [Beuche 2012] [Martinez et al. 2014] [Rubin et al. 2013]. Still, their focus in a specific strategy, such as information retrieval, different from our work that intends to give different strategies to provide a better adaptability for different scenarios. For the last topic, we found fully flexible proposals in [Krueger and Clements 2013] [Beuche 2012] [Martinez et al. 2014]. These proposals proved that can be used for different situations with the support of tools, different techniques and artifacts. We think, however, that our proposal gives more attention to the planning phase, something we have not found in those works. Nonetheless, those works present a better validation than ours, giving them more reliability. Additional related works can be found in [Assunção et al. 2017].

3. A Generic Process for Software Product Line Re-engineering

Based on the information collected when analyzing related work, we have created our generic re-engineering process for SPL. The process takes a set of product variants or a single product as input, analyzes its artifacts and extracts its features. The process is detailed in the following sections. The process structure can be divided in: 1) Sub-processes: activities performed during the process which contain other activities inside them;

2) Activities: actions performed that use and generate different artifacts; 3) Actors/Roles: which define the actors that will perform an activity; 4) Artifacts: which define the artifacts used in each activity; 5) Techniques: which are the possible techniques used to extract features from the product artifacts.

Concerning the roles, the possible are: 1) Domain Engineer: possesses valid knowledge related to an area of human endeavour, an autonomous computer activity, or other specialized discipline [Kang et al. 1990]; 2) Architect: has valid knowledge related to software architecture, design and architectural patterns; 3) Analyst: requirements specialist; 4) Developer: possesses knowledge of programming languages and technologies; 5) Feature Tester: must know domain glossary, domain constraints, and requirements information; 6) Feature Retriever: must have knowledge about feature retrieval strategies and techniques.

Regarding the artifacts, they can be of two types: input or output. Input artifacts can be categorized as mandatory, optional or alternative. Mandatory are artifacts that must be used in an activity, optional are those that do not have to be used, and an alternative group is one in which at least one artifact must be used. This allows the process to be instantiated in different forms according to the needs of those performing it. Subprocesses and activities are described below.

Process Overview: The process is composed of three main sub-processes: `Planning`, `Detection` and `Documentation Analysis`. `Detection` and `Documentation Analysis` are parallel sub-processes, shown in Figure 2. During `Planning`, team information is gathered to be used during `Documentation Analysis` to make choices about artifact collection. During `Detection`, the features are extracted, categorized and grouped. The extraction is made based on the techniques which are also chosen during `Detection`. During the `Documentation Analysis`, the re-engineering documentation is collected and compiled. All the activities of this subprocess are optional and they can be performed at the same time of any activity from `Detection`.

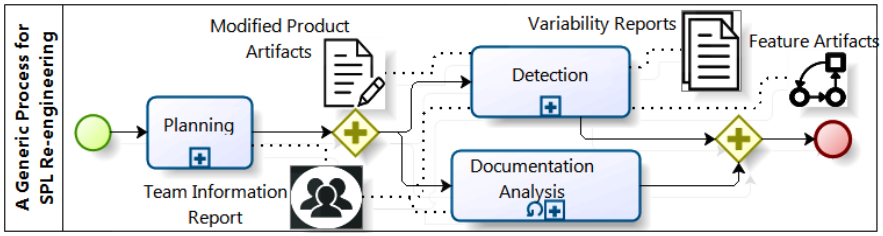


Figure 2. Process Overview

Planning: The first activity of this sub-process is to `Collect Team Information`, as shown in Figure 3. During this activity, information about the team that will execute the process is collected. This information includes experience, skills, knowledge and preferences of each member. The information is registered in a document, and stored in a database. This activity can be performed by any actor and has two optional inputs: `Project Plan` and `Business Organization Chart`. Its output is the team information report. The second activity is to `Define Roles and Tasks` based on the information collected on the previous activity. Here, all actors participate

and the team information report is updated.

Detection: The first activity performed here is `Feature Search`, which is a sub-process shown in Figure 4. During `Feature Search`, strategies and techniques are chosen and applied in product artifacts to find and extract features. To decide which strategies and techniques must be used, the team information report and the documentation generated during the `Documentation Analysis` are analyzed.

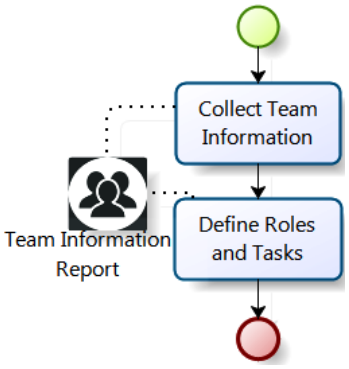


Figure 3. Planning Sub-process

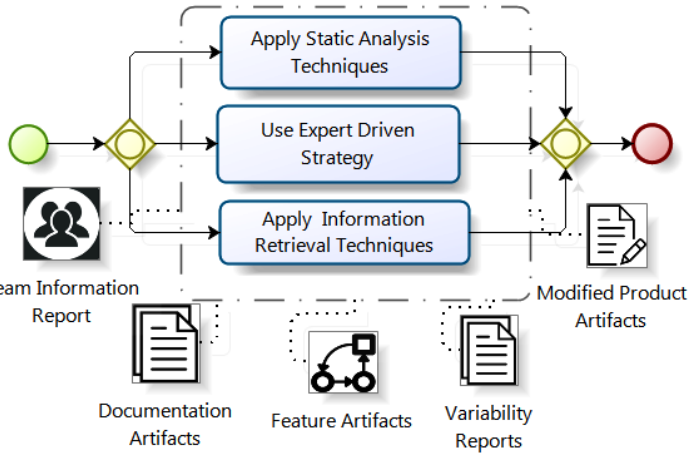


Figure 4. Feature Search Sub-process

`Detection` generates three outputs: 1) Variability reports: written in natural language, contain information about the feature extraction results; 2) Feature artifacts: outputs generated from the feature retrieval techniques, such as dependency trees and feature descriptions. 3) Modified product artifacts: product artifacts modified to include feature entry points (*e.g.* comments, markings, and tags.). These modifications do not have impact in the products execution. `Detection`, however, may not be singly performed; it is executed along side `Documentation Analysis`.

Documentation Analysis: The activities of this sub-process are optional, and are performed (or not) based on retrieval techniques, strategies, and artifacts available. All activities of `Documentation Analysis` can be performed at the same time of any activity from `Detection`. During `Documentation Analysis`, domain information is collected, a domain glossary is created, requirements information of the products are gathered, and information about architecture, technologies and artifacts is registered. The main contribution of this sub-process is to serve as a support process for `Detection`. This is done by using the documentation collected and registered in `Documentation Analysis` to make decisions, extract data, and guide the feature retrieval activity.

4. Assessing Our Process

We have applied a survey[1] to researchers and professionals of the SPLE and Re-engineering field to evaluate the impact, relevance and originality of our proposal in the SPL re-engineering field. We believe that this early feedback may give us information

[1]The details of the questionnaire instrument of the survey can be found at `http://tiny.cc/eres2017`

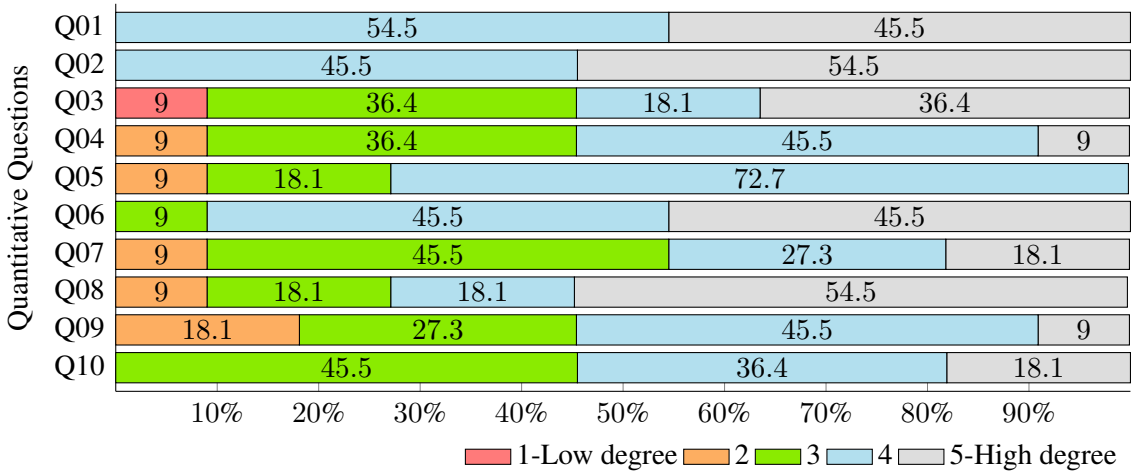
needed to improve and expand our process. In the following sections we explain the survey and discuss the results.

Survey Structure: We split the survey in six sections, intending to better organize the information and get better understanding from the participants. The first section is an introduction to the survey, briefly showing the goals of our research as well as the goals of the survey itself. The second section of the survey has questions regarding the participants' data such as academic degree, knowledge and experience in the SPLE area. The third section has our process documentation and is responsible for giving the information the participants needed to answer the later questions. The fourth and fifth sections contained quantitative questions to assess our process. We have used the Likert Scale to give participants options to rate the answers based on their opinion where the number 1 represents a low degree of agreement and 5 represents a high degree of agreement with the question. The last section was created to ask for feedback regarding the survey, such as clarity of the questions, comments about the survey, and comments about our process.

Participants' Profile: To answer our survey, we considered researchers and professionals that are working in either the SPLE or the re-engineering field. Within the survey, we included questions to obtain information about the participants, such as knowledge, skills and experience. The total number of answers was eleven, which we consider a small amount of participants; however, it do not reduces the importance of the survey. Regarding the experience with SPL, we have asked how many years the participants have been working or researching in the SPL engineering field. The answers proved to be satisfactory for us because 64% of the participants have been working or researching in this field for 1 to 5 years. We have also asked about the participants technical knowledge related with topics of interest for this work. The great majority of the participants answered to have technical knowledge regarding SPL, Re-engineering and Process Documentation. Less than a half of the participants also express to have knowledge about Feature Retrieval Techniques and SPL Re-engineering.

Survey Questions: Our survey had a total of 10 questions, the quantitative results are showed in Figure 5. In the first question, we asked if a generic process for performing SPL re-engineering would be a relevant contribution to the field (**Q01**). For the second question, we asked about the relevance of customization and applicability of the process in different scenarios (**Q02**). For both question, we got good results as shown in Figure 5. In the third question, we asked if our process is a novel contribution to the field (**Q03**), where we obtained mixed answers. The fourth question was related with the impact of our proposed process (**Q04**). With the following question been related with the flexibility of our process to collect information about the team, assign roles and define tasks (**Q05**). For these three question, we obtained a high level of acceptance. For the next two questions, we have asked exclusively about the artifacts documented within our process. The first question was regarding the relevance of a process that provides flexibility for the artifacts used (**Q06**). The second question was if the participants thought that our process is a novel contribution in regards only to its flexibility related with artifacts (**Q07**). Similar to the two questions above, we have asked if, considering the flexibility related with strategies and techniques for feature retrieval, a SPL re-engineering process is a relevant contribution (**Q08**). We have also asked if, regarding the flexibility related with strategies and techniques for feature retrieval, our process is a novel contribution to the SPL re-

engineering field (**Q09**). For these four question, we have obtained another mixed set of answers. The last quantitative question was related with the applicability of our process (**Q10**). For this question, we obtained once more a mixed set of answers.



PS: Some percentages generate a recurring decimal which do not result in a 100% total.

Figure 5. Frequency Diagram of the Quantitative Questions

Results Analysis: By analyzing the results of the quantitative questions we have concluded that our process is on the right direction towards our main goal. We could notice that there is still some lack of flexibility in some points, which we will work to reduce. Despite these problems, we could obtain a high degree of confidence related with the impact, relevance and originality of our proposal. For the first topic, we could see with the results of the questions that a generic process that provides flexibility for SPL re-engineering would make impact in the field. This indicates that the motivation for creating our process makes sense and we have been doing a significant research in the area. Regarding the relevance of our proposal, we could also obtain satisfactory results showing that our proposal is being developed in the right direction. In the last section of the survey, we have placed two open questions to give participants the opportunity to leave comments and suggestions for our process. Most of the suggestions were related to expanding the flexibility of our proposal. The participants think that we should improve some aspects of the documentation to obtain more clarity about how flexible the process really is. Another suggestion was related with the process being specific in some points such as artifacts used by some retrieval techniques. Some participants suggested that we define these techniques in a generic way and provide supporting tools and methods as a recommendation for executing each technique.

5. Conclusion

In this work, we have presented a generic process for software product line re-engineering. Our proposal aims to give enough flexibility for those that may execute it to reduce cost and effort when performing the re-engineering process. This flexibility is made possible through the use of optional and alternative input artifacts, different strategies and techniques for feature extraction, and by making some of the activities optional. With the application of the survey, with quantitative questions, to evaluate our proposed process,

we obtained an early feedback from researchers and professionals of the SPLE and Reengineering field. The results of the survey showed that our proposal may be a relevant contribution in the area. This was the most important information derived from the survey's feedback because we want to have enough confidence before moving forward with our research. We could also see that some improvements have to be made regarding the flexibility of our process, which we plan to improve in future works. We also intend to validate the proposal through a case study in a real development environment.

References

Assunção, W., Lopez-Herrejon, R., Linsbauer, L., Vergilio, S., and Egyed, A. (2017). Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Soft. Engineering*, pages 1–45.

Beuche, D. (2012). Modeling and building software product lines with pure:: variants. In *Proc. of the 16th Inter. SPL Conf. - Volume 2*, pages 255–255. ACM.

Chikofsky, E. and Cross, J. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE software*, 7(1):13–17.

Damaševičius, R., Paškevičius, P., Karčiauskas, E., and Marcinkevičius, R. (2012). Automatic extraction of features and generation of feature models from java programs. *Information Technology And Control*, 41(4):376–384.

Eyal-Salman, H., Seriai, D., Dony, C., and Al-msie'deen, R. (2012). Recovering traceability links between feature models and source code of product variants. In *VARiability for You Workshop: Variability Modeling Made Useful for Everyone*, pages 21–25. ACM.

Kang, K., Cohen, S., Hess, J. A., Novak, W., and Peterson, A. (1990). Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC Document.

Krueger, C. and Clements, P. (2013). Systems and software product line engineering with biglever software gears. In *Proc. of the 17th Inter. SPL Conference co-located workshops*, pages 136–140. ACM.

Martinez, J., Ziadi, T., Klein, J., and Le Traon, Y. (2014). Identifying and visualising commonality and variability in model variants. In *European Conference on Modelling Foundations and Applications*, pages 117–131. Springer.

Pohl, K., Böckle, G., and van Der Linden, F. (2005). *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media.

Rubin, J. and Chechik, M. (2012). Combining related products into product lines. In *Inter. Conf. on Fundamental Approaches to Software Engineering*, pages 285–300. Springer.

Rubin, J., Czarnecki, K., and Chechik, M. (2013). Managing cloned variants: a framework and experience. In *17th Inter. SPL Conference*, pages 101–110. ACM.

Stoermer, C. and O'Brien, L. (2001). Map-mining architectures for product line evaluations. In *Working IEEE/IFIP Conf. on Software Architecture*, pages 35–44. IEEE.

Van der Linden, F., Schmid, K., and Rommes, E. (2007). *Software product lines in action: the best industrial practice in product line engineering*. Springer Science & Business Media.

Atta Method: A Nature Based Approach

Miguel Ecar[1]

[1]Software Engineering – Federal University of Pampa - UNIPAMPA
Alegrete – RS – Brazil

miguel@ecarsm.com

Abstract. *Traditional software engineering approaches were developed based on old manufacturing process, in term of roles, responsibilities, tasks and hierarchy. New approaches, known also as Agile approaches, are based on a fusion of these traditional approaches and the Agile Manifesto, which changes the perspective of main software process objectives. This paper proposes a new software engineering method called Atta, based on a nature process. Furthermore this paper explore the main current software approaches and contrast them with the proposed one. The result is a new software engineering method, which has bases on nature and brings a new vision of organizational processes, responsibilities, roles and task division. Nonetheless this method needs to be validated.*

1. Introduction

This paper presents a new proposal of agile software development method. Atta method is inspired in a nature existent society. Atta sexdens is a type of Leaf Cutter ant genus, which are among the most complex found in all ants [Wilson 1980].

This idea came from the need of a more structured hierarchy, specially when new members are integrated into the team. But even adding some more structured organization and some roles segmentation, is still lightweight, having basis on the agile manifesto and avoiding plan-based approaches.

The inspiration is group of insects to create and propose a new vision for agile software development. The Agile Manifesto was the start point to make a link with the knowledge about these insects organization.

Atta method, is composed by, roles, events, tasks and artifacts, all of them inspired in ants organization and translated to software engineering. It is classified as agile method, due to its principal basis in terms of software development was the Agile Manifesto which have four values, they are (1) Individuals and interactions over processes and tools; (2) Working software over comprehensive documentation; (3) Customer collaboration over contract negotiation; (4) Responding to change over following a plan [Beck et al. 2001].

Taking this into account, Atta method is an approach based on an existent society in nature which, has as common goal to survive and to increase with constant interaction among each member (value 1). They work to find inputs to farm their own food (value 2). They collaborate with the queen, and each caste inside the colony has its own tasks and responsibilities (value 3). Moreover, they adapt themselves to climate changes and attackers, and keep working for colony growth (value 4).

There are several approaches that are called agile methods, which are considered lightweight methods in that they are characterized by short, iterative development cycles, self-organizing teams, simpler designs, code refactoring, test-driven development, frequent customer involvement [Abran et al. 2004]. Agile methods are based on the notion of incremental development and delivery. They share a set of principles, based on the agile manifesto, and so have much in common [Sommerville 2010].

The rest of the paper is organized as follows, section 2 contains the background about leaf cutter ants. Section 3 detail how Atta method is composed. Finally, section 4 presents conclusions and future work.

2. Background

Leaf Cutter ants are among the most complex found in all ants, can be expected to provide an estimate of the upper limits of colonial organization in these insects [Wilson 1980].They are one of the most important groups of insects because they attack plants intensely and constantly at any development stage [ZANETTI et al. 2002]. They are called leaf cutter due to main task is to cutter leaf peaces and take to the colony, these leaf peaces are used to grow a specific fungus which leaf cutter ants feed on [LOECK et al. 2001]. Due to its efficiency in leaf cutting, they are considered among the more important farming pests in Latin America [de Abreu and Delabie 1986, Hernández and Jaffé 1995].

The organization of these ants is composed by 4 castes. The figure 2 shows the organizational structure.

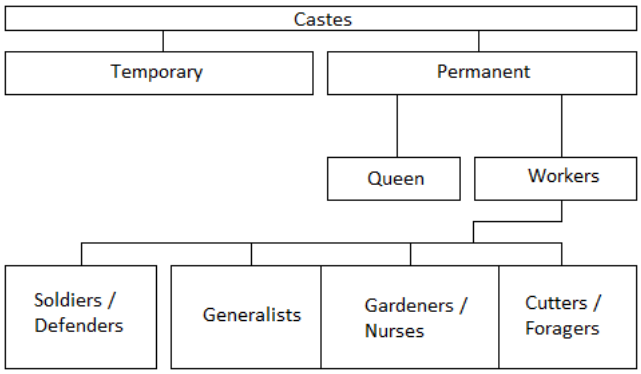


Figure 1. Atta Castes. adapted from [ZANETTI et al. 2002]

- **Gardeners and Nurses:** workers specialized on the care of fungus, final treatment and implantation of substrate particles, brood care, and parasol riding.
- **Generalists:** workers that perform a relatively wide variety of tasks, including the degradation of vegetation prior to its incorporation into the garden, major reconstruction of the fungus combs, disposal of refuse, transport of other workers and queen care.
- **Foragers or cutters:** workers that explore, recruit to new vegetation, cut and retrieve the vegetation, and excavate the nest.
- **Defenders or soldier:** workers that protect colony from invaders and help cutters.

3. Atta Method

Atta method emerged from the need to more structure in a agile software development environment. It is composed by roles, events, tasks and artifacts.

3.1. Roles

Atta Method is composed of five roles, they are: **Queen, Soldier, Cutter Dev. Team, Generalist Dev. Team and Quality Assessment Dev. Team** as shown in table 1

- **Queen:** The Queen is a single person, which is the owner of the business. This person is not part of development team in other words, the Queen do not code but its only responsibility is to feed the product.
- **Soldier Team:** The Soldiers must correspond for about 5% of the whole Atta Team. This team has mainly four responsibilities, coaching, consulting, management and also development. This team must be composed by specialists. Each Atta Team should decide what specialties are demand, for example, DBA, UX Designer, Manager, Tester, Measurer, etc. It is important to note that each specialty must be a single representative in the Soldier Team.
- **Cutter Development Team:** The Cutters is composed of junior developers. The team correspond for about 15% of Atta Team. Its responsibility is to develop a prototype, more precisely an High Quality/Fidelity Functional Prototype which follows the DoD".
- **Generalist Development Team:** Generalists are responsible for make sure that the cutter team is following the DoD'. They correspond of about 30% of Atta Team and it is composed of junior developers and trainees, the team should test, white test script and detailed specification to be followed by QA Team.
- **Quality Assessment Team:** The QA Team should correspond for about 60% of Atta Team and must be filled with senior developers. They are responsible to take the prototype delivered by the Cutter Team and raise it to DoD", following the detailed specification delivered by Generalist Team. Moreover, they are responsible for integration tests and make sure that the build keeps consistent with the new increment.

Table 1. Atta Team Roles

Role	Professional	%
Atta Queen	Business Man	*
Atta Soldiers	Specialists	5%
Atta Cutters	Junior Developers	15%
Atta Generalists	Junior Dev and Trainees	30%
Atta QA's	Senior Developers	60%

*Single Person

3.2. WorkFlow

Atta workflow is presented in figure 3.2. The process starts with the Queen, which is responsible for maintain the product high level requirement specification. It is divided in

small increments that are taken by the Cutters team which develop the increment reaching the DoD'.

After that, the Cutters output is a high quality functional prototype that is taken by Generalists who develop test scripts and detailed requirement specification which is also part of DoD'. This step is iterative and go back to Cutters if specification detail is missing. After that, the responsibility goes to QA team, which increase the prototype developed by Cutters following detailed specification until reach the tests written by Generalist. This is also an iterative step which round in QA team until reaches DoD". The out put from QA team is the increment added to the whole project.

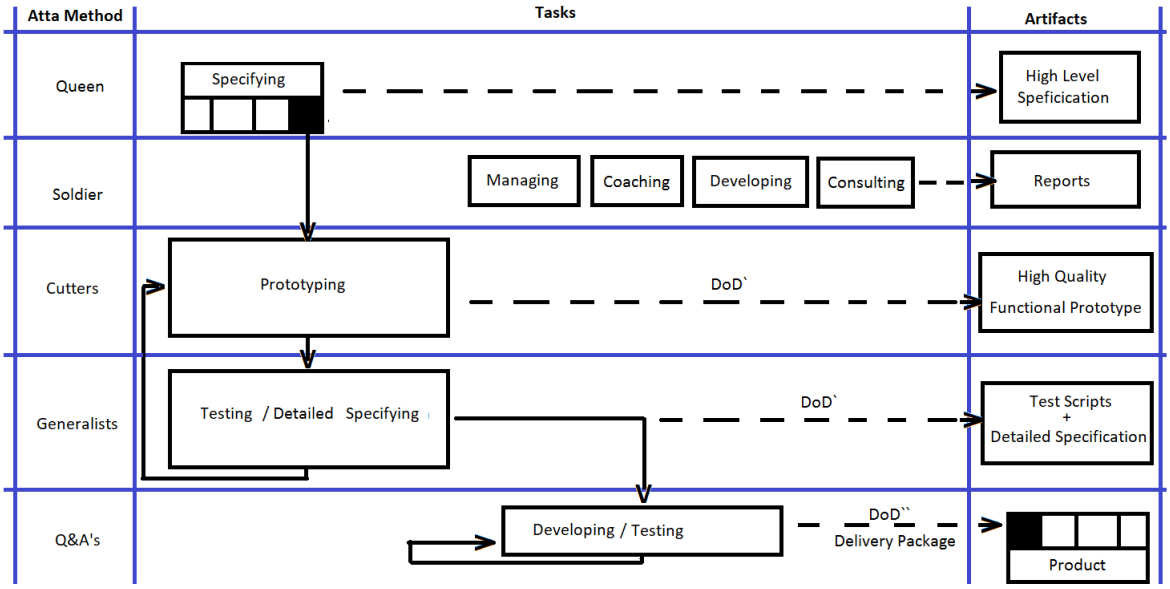


Figure 2. Atta Workflow

3.3. Events

Atta process has two main events, Cutting and Farming. Cutting, is a continuous event, this event is performed by Cutters and Generalists, that are continuously developing high quality functional prototypes. Farming is an event performed by Generalists and QA team, which is time-boxed. Protection is an event performed by Soldiers, which happen transversely in the process. The cycle is shown in figure 3.3.

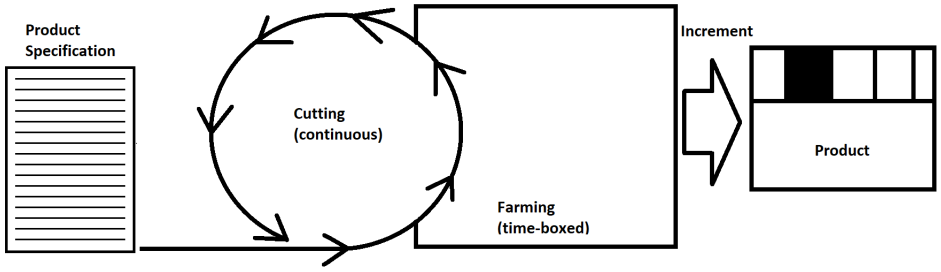


Figure 3. Atta Events Cycle

3.3.1. Cutting

The Cutting event is performed by Cutters and Generalists. It is classified as a continuous event and it is divided in two parts.

The first part a cutter team member (or simply cutter) takes a single part of requirements pile and starts its prototyping. This single part is called Leaf Clipping and it is characterized as the smallest part with value for the user. Moreover, a cutter must be able to prototype the increment by itself.

The second part, is performed by a Generalist. which should write detailed specification, the meaning of "detailed" is in DoD', and test scripts.

It is important to note that cutters and generalists should work together, the one responsibility complement the other one responsibility. Thus, together both team members must reach the DoD'. Once, it is done, cutters and generalists star the cycle again, taking a new leaf clipping.

It is important to always have increments in DoD' tag to supply at least one next Farming.

3.3.2. Farming

The Farming event is performed by QA team. It is classified as time-boxed event, in other words, it has a defined scope and should start and finish in a previously defined time. This time-box is calculated in the project begin and should be agreed by everyone, following the idea of "sprint" in scrum [Schwaber and Sutherland 2011]. The farming scope if calculated based on the amount of software that QA team manage to take from DoD' until reach DoD" in the time-box.

Furthermore, DoD" should contemplate specification for a high quality increment and its integration to the rest of the system. Thus, the delivery at the end of farming is increments added to the whole software.

3.3.3. Protection

Protection is an event performed by the Soldiers team. Soldiers should ensure that the project is healthy, so they can act in any event performing tasks, such as, consulting, managing, coaching, measuring and etc, according to its specialty.

3.4. Artifacts

Atta method has a different number of artifacts. They are, definition of done, high level specification, reports, high quality prototype, test scripts, detailed specification, increment and incremented software.

The term "Definition of Done" was taken from Scrum glossary. When a Product Backlog item or an Increment is described as "Done", everyone must understand what "Done" means. Although this varies significantly per Scrum Team, members must have

a shared understanding of what it means for work to be complete, to ensure transparency [Schwaber and Sutherland 2011].

In Atta Method, there are two DoD's, the one called DoD' (DoD one) that is defined to be the reach point for cutters and generalists, in terms of prototyping, test scripts and detailed requirements. The other one, is called DoD" (DoD two) the is the reach point for QA team and should contemplate the minimum specification for increment integration in whole project.

High level specification is done by the Queen, which is the business person, which knows exactly what the software should do and how it should be done. It is indicated to use, high level specification writing techniques, such as, user story, epics, wire frames, or event simple plain text.

Reports are made by soldiers, according to its specialty, they should have in mind that a report is made when it aggregate value to Atta team as a whole. So, measurement analysis, controlling charts, manuals and etc, are performed by soldiers to make the process more transparent.

High quality prototype is expected from cutters and it should agree with DoD'. test scripts and detailed specification, are expected from generalists, they should also agree with DoD' and detailed specification must be done, also, when it is valuable for the whole Atta team, any writing technique may be used since, it is agreed with all members.

The increment and incremented software are expected from QA team at the end of farming, They should agree with DoD" and are in fact the software expected by the Queen and final users.

3.5. Comparative Analysis

The Table 2 shows a comparative analysis among Atta and other agile approaches. The agile approaches compared are eXtreme Programming (XP) [Maurer and Martel 2002], Scrum [Schwaber and Sutherland 2011], Feature-Driven Development (FDD) [Palmer and Felsing 2001], Adaptive Software Development (ASD) [Highsmith 2013], Dynamic Systems Development Method (DSDM) [Stapleton 1997].

Based on Table 2, it is possible to note that Atta method has significant differences in terms of project size, team size, development style, environment and business culture, from other agile approaches.

4. Conclusion and Future Work

This paper presents a new software development method, called Atta method. It is inspired in a nature process, the leaf cutter ants organization.

The basis for this method development was the need for a more structured software development process. Current agile approaches left lacks of segmentation, and it may be difficult to implement in an existent company with huge number of developers.

Thereby, the raised hypothesis is that Atta method may feet in this type of organization bringing a segmented environment at the same time as an agile culture and thinking.

Table 2. Agile Approaches Comparative Analysis.

Criteria	XP	Scrum	FDD	ASD	DSDM	Atta
Scope Project Size	Small, medium	Small, medium, and scalable for large	Small, medium, and large (business projects/ applications)	Large and Complex projects	Small and large projects (Business Applications)	Medium, Large and Complex projects
Team Size	<10	<10 and multiple teams	No limit, scalable from small to large teams	Not mentioned	Minimum 2 and Maximum 6 (Multiple teams)	No limit, scalable from small to large teams
Dev. Style	Iterative, rapid	Iterative, rapid	Iterative design and construction	Iterative and rapid development, distributed development	Iterative, rapid development and cooperative	Iterative, rapid development and cooperative
Physical Environment	Co-located teams and distributed teams (limited-interaction)	Not specified	Not specified	Co-located teams and distributed teams	Not mentioned	Co-located teams and distributed teams
Business Culture	Collaborative and cooperative	Not specified	Not specified	Not specified	Collaborative and cooperative	Collaborative and cooperative

*Adapted from [Qumer and Henderson-Sellers 2008]

Next step for this work is an empirical validation to serve as concept proof and refinement of this proposal. The plan is expose the Atta method idea and get feedback from community. It should be noted that Atta method is currently only an idea, but with refinement and enrichment, it may have a good contribution for agile software community.

References

Abran, A., Moore, J. W., Bourque, P., Dupuis, R., and Tripp, L. L. (2004). *Guide to the software engineering body of knowledge: 2004 version SWEBOK*. IEEE Computer Society.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). The agile manifesto.

de Abreu, J. and Delabie, J. (1986). Controle das formigas cortadeiras em plantios de cacau. *Revista Theobroma*, 16:4.

Hernández, J. V. and Jaffé, K. (1995). Dano econômico causado por populações de formigas atta laevigata (f. smith) em plantações de pinus caribaea mor. e elementos para o manejo da praga. *Anais da Sociedade Entomológica do Brasil*, 24(2):287–298.

Highsmith, J. (2013). *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley.

LOECK, A. E., Grützmacher, D., and Storch, G. (2001). Distribuição geográfica de atta sexdens piriventris santschi, 1919, nas principais regiões agropecuárias do estado do rio grande do sul. *Current Agricultural Science and Technology*, 7(1).

Maurer, F. and Martel, S. (2002). Extreme programming. rapid development for web-based applications. *IEEE Internet computing*, 6(1):86–90.

Palmer, S. R. and Felsing, M. (2001). *A practical guide to feature-driven development*. Pearson Education.

Qumer, A. and Henderson-Sellers, B. (2008). An evaluation of the degree of agility in six agile methods and its applicability for method engineering. *Information and software technology*, 50(4):280–295.

Schwaber, K. and Sutherland, J. (2011). The scrum guide. *Scrum Alliance*, 21.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley Publishing Company, USA, 9th edition.

Stapleton, J. (1997). *DSDM, dynamic systems development method: the method in practice*. Cambridge University Press.

Wilson, E. O. (1980). Caste and division of labor in leaf-cutter ants (hymenoptera: Formicidae: Atta). *Behavioral ecology and sociobiology*, 7(2):143–156.

ZANETTI, R., Carvalho, G. A., Santos, A., Souza-Silva, A., and Godoy, M. S. (2002). Manejo integrado de formigas cortadeiras. *Lavras: UFLA*, page 16.

Integrando Engenharia de Software e Interação Humano-Computador em um Processo de Desenvolvimento de Software para/com Crianças

Jéssica C. B. Pereira, Amanda Meincke Melo

Universidade Federal do Pampa (Unipampa) – Campus Alegrete
Av. Tiarajú, 810 – Ibirapuitã – 97.546-550 – Alegrete – RS – Brasil

jcbp54@gmail.com, amanda.melo@unipampa.edu.br

Abstract. *Children are potential users of digital technologies as much as adults. Though there are several experiences described in the literature of children's involvement in software development to their use. However, these experiences do not show any Software Engineering approach. This work aims at proposing a software development process for/with children, integrating Software Engineering and Human-Computer Interaction methods, techniques and practices, to support researchers and software developers. This process, described in BPMN, can be refined and adopted as a reference for the development of software with children in new projects.*

Resumo. *Crianças são potenciais usuárias de tecnologias digitais tanto quanto os adultos. Embora haja várias experiências relatadas na literatura do envolvimento da criança no desenvolvimento de software para seu uso, essas não apresentam uma abordagem de Engenharia de Software. Este trabalho tem como objetivo, portanto, propor um processo de desenvolvimento de software para/com crianças, integrando métodos, técnicas e práticas de Engenharia de Software e Interação Humano-Computador, para auxiliar engenheiros de software e pesquisadores. Esse processo, descrito em BPMN, pode vir a ser aprimorado e adotado como referência para o desenvolvimento de software com crianças.*

1. Introdução

Crianças são usuárias de tecnologias digitais e, assim como os adultos, podem contribuir efetivamente para o seu desenvolvimento [Druin 2002]. Embora essas tecnologias adotem soluções em software, processos e métodos da Engenharia de Software (ES), diferentemente da Interação Humano-Computador (IHC), não consideram explicitamente o público infantil como possível parte interessada em seu processo de desenvolvimento.

Segundo Barbosa e Silva (2010), ES e IHC possuem enfoques diferentes no desenvolvimento de software, havendo alternativas à integração de ambas as áreas. Uma abordagem sistemática e disciplinada ao desenvolvimento de software para/com crianças, envolvendo a integração entre ES e IHC, tem o potencial de contribuir ao desenvolvimento de um sistema de software de qualidade ao mesmo tempo em que considera explicitamente as experiências, as necessidades e os anseios dos usuários envolvidos.

Propõe-se, neste artigo, apresentar um processo de desenvolvimento de software

para/com crianças, em BPMN (do inglês, *Business Process Model Notation*), organizado a partir do uso de técnicas como observação participante, reuniões, entrevistas e análise documental no contexto do projeto de pesquisa Design Inclusivo para/com Crianças e Adolescentes de Tecnologias Digitais de Informação e Comunicação (DICA-TDIC).

O texto está estruturado como segue. A Seção 2 apresenta o resultado de uma revisão de literatura realizada nas bases de dois eventos científicos nacionais, o Simpósio Brasileiro de Engenharia de Software (SBES) e o Simpósio sobre Fatores Humanos em Sistemas Computacionais (IHC), além de uma conferência internacional dedicada ao tema do design da interação e crianças: *Interaction Design and Children* (IDC). A Seção 3 descreve a metodologia adotada no trabalho. A Seção 4 sumariza os resultados obtidos. Finalmente, a Seção 5, apresenta algumas considerações finais e trabalhos futuros.

2. Revisão de Literatura

A revisão de literatura foi realizada, no período [2012, 2016], a partir de pesquisas nos anais de dois eventos nacionais de referência, um na área de Engenharia de Software – o SBES – e outro na área de Interação Humano-Computador – o IHC –, e de um evento internacional de referência para o design da interação e crianças – o IDC. No SBES, foram encontrados cinco trabalhos que abordam a integração entre ES e IHC e um que apresenta um estudo sobre gestão de processo. No IHC, foram encontrados três trabalhos que apresentam formas de integração entre IHC e ES. Já no IDC, foram encontrados três trabalhos que relatam o desenvolvimento de tecnologias para/com crianças.

Os trabalhos publicados no SBES têm considerado a integração entre ES e IHC em diferentes perspectivas. Contudo, não consideram o domínio de desenvolvimento de software para/com crianças. Propõem técnicas de inspeção de usabilidade para aplicações *web* [Fernandes *et al.* 2012] e *mobile* [Rivero *et al.* 2012], a avaliação de técnicas de avaliação [Rivero *et al.* 2014][Valentim *et al.* 2014], a investigação do impacto da adoção do Scrum na satisfação do cliente [Cartaxo *et al.* 2013], além de propor uma técnica para a criação de *personas* para apoiar no desenvolvimento de aplicações *web* para dispositivos móveis [Ferreira *et al.* 2015].

Trabalhos publicados no evento IHC observam requisitos como acessibilidade e usabilidade no *design* e na avaliação de tecnologias computacionais interativas [Borges *et al.* 2012][Braz *et al.* 2014][Boscarioli 2012]. Sua ênfase, entretanto, não está no envolvimento de crianças como parceiras de *design*, onde elas possam contribuir diretamente no desenvolvimento de um produto, expressando suas ideias, gostos e interesses.

No evento IDC, embora sejam reportadas várias experiências do envolvimento de crianças no desenvolvimento de tecnologias digitais, não foram identificadas experiências que adotem uma abordagem de ES, portanto, sistemática e disciplinada ao desenvolvimento de software [Pressman 2006]. Entre os trabalhos identificados, têm-se um estudo sobre a motivação de crianças para se envolverem com o Design Participativo [Hansen & Iversen 2013]; a apresentação de uma tecnologia desenvolvida com o envolvimento de crianças após a etapa de concepção e cujo desenvolvimento, conduzido de forma interdisciplinar, adotou o Design Centrado no Usuário [Gray *et al.*

2015] e a proposta de princípios de design multisensoriais para o Desenho Universal para crianças [Puccini *et al.* 2013].

3. Metodologia

Além da revisão de literatura, para se chegar ao modelo de processo proposto, em notação BPMN, adotaram-se técnicas como observação participante, reuniões, entrevistas e análise documental.

Realizou-se observação participante de dois estudos exploratórios envolvendo a produção de tecnologias digitais com crianças [Lôbo 2016] e em reuniões semanais do projeto de pesquisa DICA-TDIC. A observação participante é definida por Appolinário (2004, p. 144) como "Processo de observação no qual o pesquisador participa ativamente como membro do grupo que ele está estudando, utilizando esta posição privilegiada para obter informações acerca desse grupo".

A análise documental envolveu a verificação das agendas de trabalho de dois estudos exploratórios de Engenharia de Requisitos (ER) [Lôbo 2016], além da análise da descrição de dois processos de desenvolvimento de software para/com crianças no escopo do projeto de pesquisa DICA-TDIC [Ribeiro 2016][Souza 2016]. De acordo com Marconi e Lakatos (2009, p. 29), essa técnica consiste na representação condensada da informação do conteúdo de um documento.

Em complemento às reuniões semanais da equipe do projeto de pesquisa DICA-TDIC, foram realizadas reuniões com os colegas do grupo que produziam tecnologias com crianças, visando a conhecer seus processos de trabalho [Ribeiro 2016][Souza 2016]. Essas reuniões, no segundo semestre de 2016, colaboraram à dissolução de dúvidas, tomadas de decisões e à validação de modelos.

Entrevistas foram organizadas visando a orientar as reuniões para conhecer os processos de software utilizados pelos colegas do grupo DICA-TDIC. Segundo Barbosa e Silva (2010, p. 145), entrevista é "uma das técnicas mais utilizadas de coleta de dados e levantamento de requisitos. Tratando-se de uma conversa guiada por um roteiro de perguntas ou tópicos." A entrevista utilizada foi a não estruturada, que é uma entrevista flexível, onde são realizadas perguntas abertas para se aprofundar nos pontos desejados.

A abordagem adotada para o mapeamento e a modelagem do processo é a proposta por Tolfo (2013) *apud* Segui (2015), organizada em três etapas:

1. **Elaborar um mapa mental:** neste mapa são descritas as principais atividades do processo, seus atores e suas entradas e saídas;

2. **Elaborar fluxo de processo:** as atividades descritas no mapa mental são organizadas em ordem de ocorrência no processo;

3. **Modelar estado atual do processo:** adota-se a notação *Business Process Model Notation* para representar o processo em análise.

Essas etapas geram um ciclo, como ilustrado na Figura 1. Ao término deste ciclo, os responsáveis pelo modelo refinam a modelagem, analisam e discutem o fluxo e o mapa do processo.

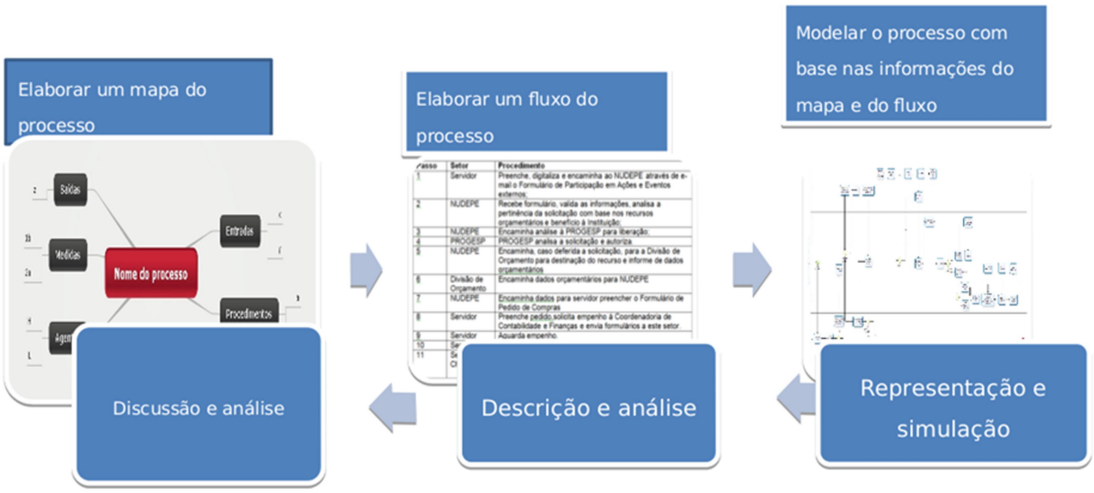


Figura 1. Abordagem para o mapeamento e a modelagem de um processo

Esse ciclo foi realizado para cada processo adotado no contexto do projeto de pesquisa DICA-TDIC, iniciando pelo processo de Engenharia de Requisitos aplicado em dois estudos exploratórios [Lôbo 2016]. Em seguida, foi realizado para apoiar a abstração de outros dois processos de desenvolvimento de *software* envolvendo crianças [Ribeiro 2016][Souza 2016].

4. Resultados

Como resultado, tem-se documentado, em notação BPMN, um processo de desenvolvimento de software que contempla a participação de crianças [Pereira 2016]. Seus níveis de abstração mais altos são apresentados nas Figuras 1, 2 e 3.

A Figura 2 evidencia as atividades realizadas pelas crianças – contidas nos subprocessos conduzidos pelos engenheiros de software. Na troca das mensagens são explicitadas as contribuições das crianças, o que inclui ideias, desenhos, histórias de usuários e sua priorização, *storyboards*, sugestões, identificação de problemas e impressões. Esses subprodutos, em níveis de abstração mais baixos, são modelados como artefatos [Pereira 2016]. Além disso, podem colaborar ativamente no desenvolvimento iterativo de protótipos e na codificação de software.

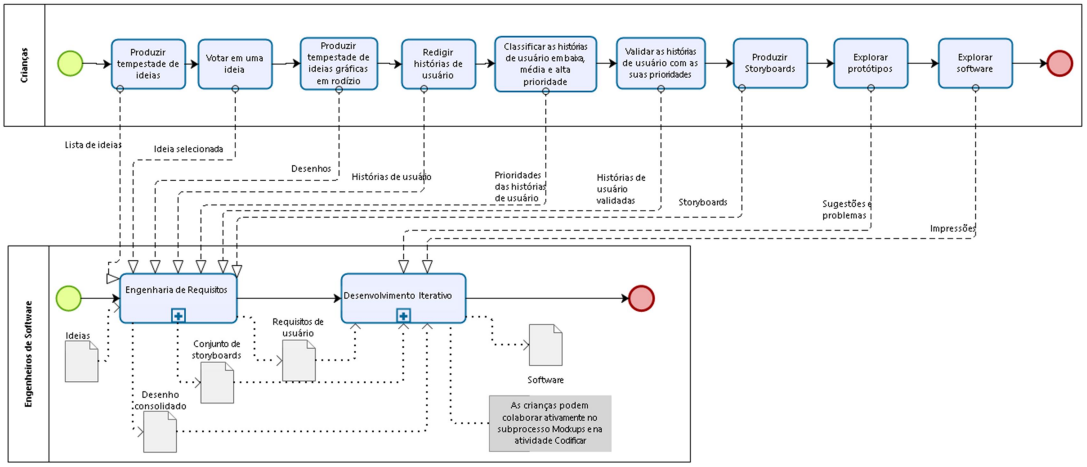


Figura 2. Modelo do processo Desenvolvimento de Software para/com crianças

Os subprocessos Engenharia de Requisitos e Desenvolvimento Iterativo são apresentados a seguir.

4.1 Engenharia de Requisitos

A Figura 3 representa o subprocesso de Engenharia de Requisitos em seu nível de abstração mais alto, com seu ciclo de atividades baseado em Sommerville (2007).

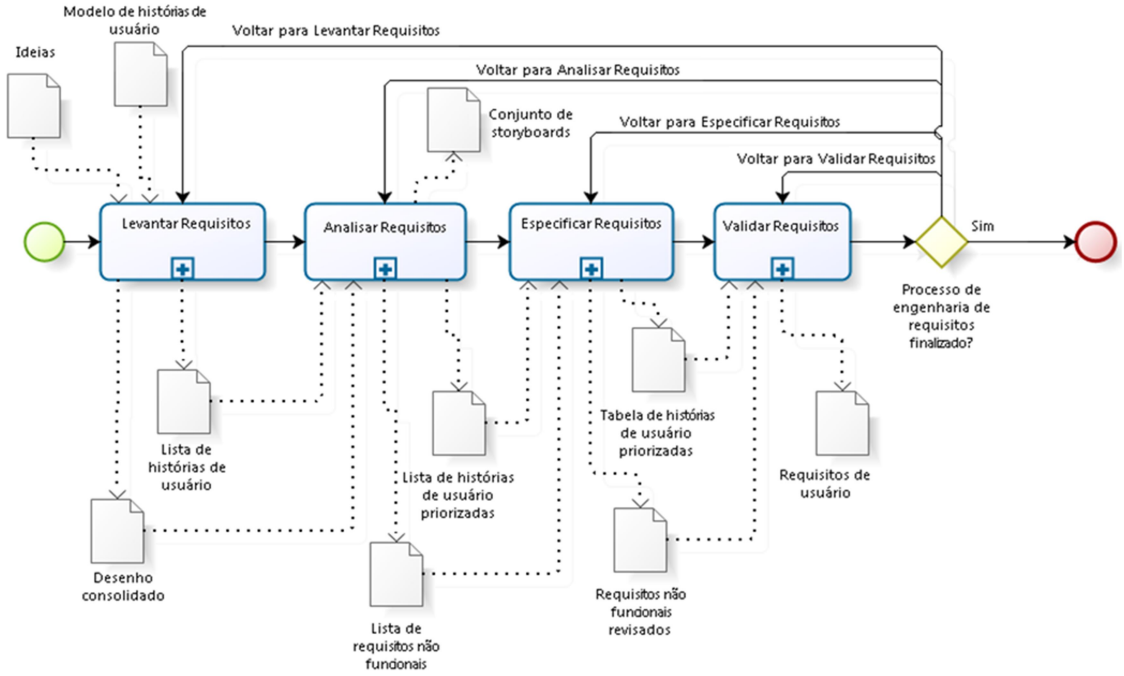


Figura 3. Modelo do subprocesso Engenharia de Requisitos

O subprocesso ER apresenta os subprocessos “Levantar Requisitos”, “Analisar Requisitos”, “Especificar Requisitos” e “Validar Requisitos”. Como entrada do subprocesso “Levantar Requisitos”, têm-se ideias para o desenvolvimento de uma tecnologia com as crianças e um modelo de histórias de usuário. Como saída, têm-se um desenho consolidado, que representa um esboço preliminar da interface de usuário, e uma lista de histórias de usuários, que representam os requisitos funcionais a serem desenvolvidos.

Esses artefatos são entradas para o subprocesso “Analisar Requisitos”. Como saídas do processo “Analisar Requisitos”, têm-se a lista de histórias de usuário priorizadas, a lista de requisitos não funcionais e um conjunto de *storyboards*, que descrevem algumas das interações do usuário com o sistema.

A lista de histórias de usuário priorizadas e a lista de requisitos não funcionais servem de entradas para o subprocesso “Especificar Requisitos”, que apresenta como saída uma tabela de histórias de usuário priorizadas e uma lista de requisitos não funcionais revisada.

A tabela de histórias de usuário priorizadas e a lista de requisitos não funcionais revisada são entradas do subprocesso “Validar Requisitos”, que gera como saída os requisitos de usuário. Antes de finalizar o processo, ocorre um evento de decisão, que determina se o subprocesso de Engenharia de Requisitos foi efetivamente finalizado ou

se ainda é necessário retornar a algum dos subprocessos de ER.

Durante o processo de Engenharia de Requisitos, as crianças participam ativamente dos subprocessos "Levantar Requisitos", "Analisar Requisitos" e "Validar Requisitos", gerando ideias, desenhos, histórias de usuário e *storyboards*. Assim, contribuem diretamente à identificação dos requisitos de usuário. Pereira (2016) detalha o subprocesso ER em mais dois níveis de abstração, explicitando a colaboração das crianças nas atividades evidenciadas na Figura 2, integradas aos subprocessos de ER.

4.2. Desenvolvimento Iterativo

A Figura 4 descreve as atividades do subprocesso Desenvolvimento Iterativo.

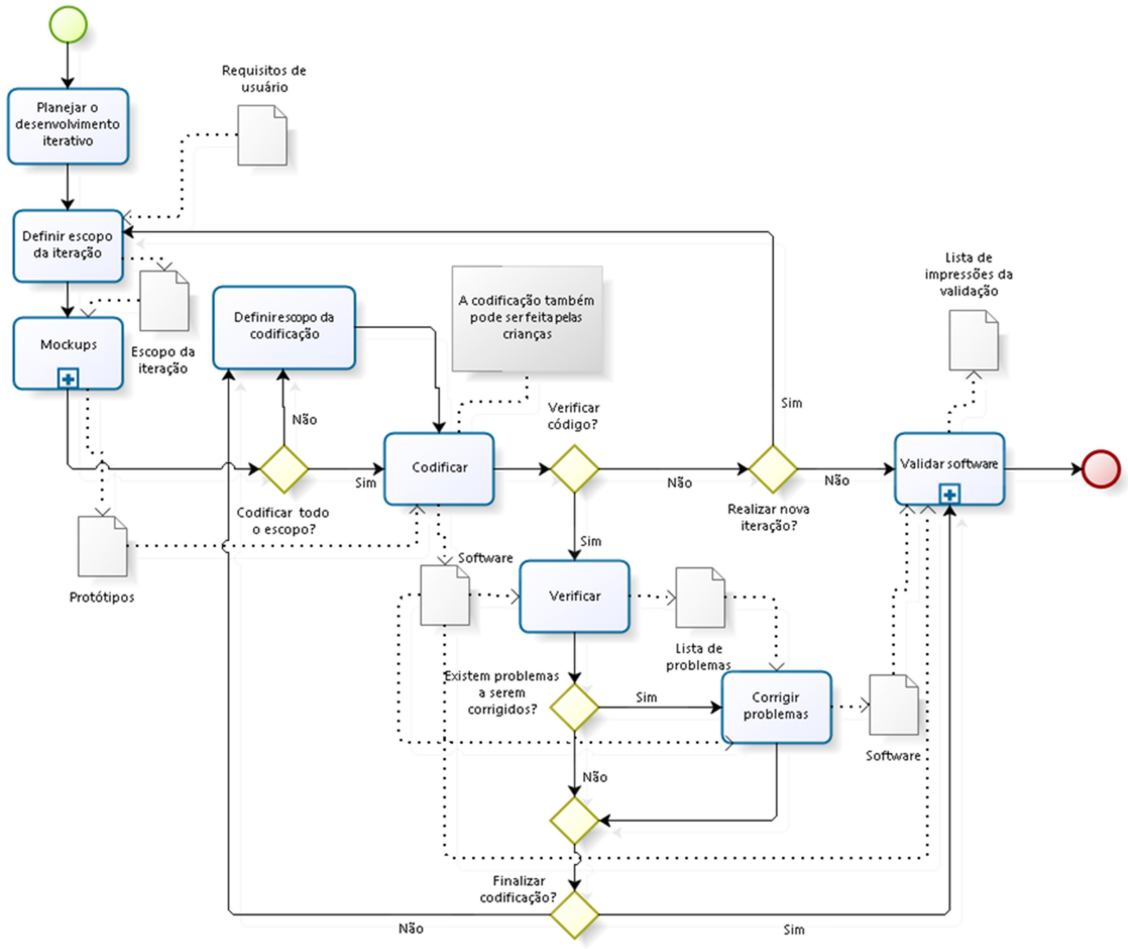


Figura 4. Modelo do subprocesso Desenvolvimento Iterativo

O subprocesso DI é iniciado com a atividade "Planejar o desenvolvimento iterativo", na qual ocorrem decisões como, por exemplo, se o desenvolvimento será iterativo e/ou incremental, métodos e técnicas de prototipação e de avaliação de interfaces de usuário a serem adotados, entre outras decisões relativas ao processo de desenvolvimento.

Na atividade "Definir escopo da iteração", que recebe os requisitos de usuário como entrada, define-se como parte do escopo todos os requisitos ou parte deles. Neste último caso, assume-se que o processo como um todo é iterativo. Sobre o escopo definido, no subprocesso "Mockups" [Muller *et al.* 1997], desenvolvem-se, com a

participação das crianças, protótipos da interface de usuário, em ciclos iterativos de prototipação e avaliação. Ao finalizar o subprocesso "Mockups", antes da atividade "Codificar", ocorre um evento de decisão para definir se todo o escopo será codificado como uma única atividade ou se a codificação será realizada de forma iterativa. A atividade "Codificar" gera software como saída. Essa atividade pode ser realizada pelas próprias crianças, por exemplo, com a adoção do ambiente Scratch [Ribeiro 2016]. O software produzido serve de insumo para as atividades "Verificar", "Corrigir problemas" e "Validar software". Uma vez que todo o software tenha sido codificado, prossegue-se ao subprocesso "Validar software". Os engenheiros de software, então, documentam as impressões da validação em uma lista que se torna saída do subprocesso.

Durante o processo DI, além de as crianças participarem do subprocesso "Validar software", podem atuar ativamente do subprocesso "Mockups" e da atividade "Codificar". Os subprocessos "Mockups" e "Validar software" são detalhados em Pereira (2016).

5. Conclusão

O modelo de processo proposto, em notação BPMN, provê apoio a engenheiros de software e pesquisadores para desenvolverem software para/com crianças, que contemplem a participação, os interesses, os gostos, as necessidades e as experiências desses usuários em uma perspectiva de ES. O modelo integra atividades de IHC a processos de Engenharia de Software, explicitando possibilidades de colaboração de crianças. Assim, dá oportunidade para que informem o que, como e porque querem.

Destaca-se como limitação do trabalho a ausência de documentação para o subprocesso Evolução de Software, que envolve acompanhar o software após sua entrega. Essa experiência, entretanto, não ocorreu no escopo do projeto de pesquisa DICA-TDIC.

Entre os trabalhos futuros, o modelo pode subsidiar o desenvolvimento de software com crianças em novos projetos. Além disso, pode ser revisado e aprimorado, por exemplo, contemplando o subprocesso Evolução.

6. Referências

Appolinário, F. (2004) Dicionário de Metodologia Científica, Atlas.

Barbosa, S. D. J., Silva, B. S. (2010), Interação Humano-Computador, Elsevier.

Borges, L. C. L. F., Filgueiras, L. V. L., Maciel, C., Pereira, V. C. (2012) Customizing a communication device for a child with cerebral palsy using participatory design practices: contributions towards the PD4CAT method. In: XI IHC, p. 57-66. SBC.

Boscarioli, C., Baqueta, J. J., Salles, C. G., Colling, J. P. (2012) Avaliação e design de interação de jogos voltados ao aprendizado de crianças surdas. In: XI IHC, p. 25-26. SBC.

Braz, P., Raposo, A., Souza, C. S. (2014) Uso de design probes no design de tecnologias para terapeutas de crianças com autismo. In: XIII IHC, p. 140-149. SBC.

Cartaxo, B., Araujo, A., Barreto, A. S. (2013) The Impact of Scrum on Customer Satisfaction: an Empirical Study. In: XXVII SBES, p. 129-136. SBC.

Druin, A. (2002) The role of children in the design of new technology. *Behaviour and information technology*, v. 21, n. 1, p. 1-25. Taylor & Francis.

Fernandes, P., Conte, T., Bonifácio, B. (2012) WE-QT: A Web Usability Inspection Technique to Support Novice Inspectors. In: XXVI SBES, p. 11-20. SBC.

Ferreira, B., Conte, T., Barbosa, S. D. J. (2015) Eliciting Requirements using Personas and Empathy Map to Enhance the User Experience. In: XXIX SBES, p. 80-89. SBC.

Gray, S., Robertson, J., Rajendran, G. (2015) BrainQuest: an active smart phone game to enhance executive function. In: XIV IDC, p. 59-68. ACM.

Hansen, E. I. K., Iversen, O. S. (2013) You are the real experts: studying teenagers' motivation in participatory design. In: XII IDC, p. 328-331. ACM.

Lôbo, K. L. S. (2016) Uma abordagem ética e responsável ao design para/com crianças: investigando a integração de práticas de interação humano-computador à engenharia de requisitos, Unipampa.

Marconi, M. A., Lakatos, E. M. (2009) Metodologia Científica, Atlas, 7ª. ed.

Muller, M. J., Haslwanter, J. H., Dayton, T. (1997) "Participatory Practices in the Software Lifecycle", Handbook of Human-Computer Interaction, Elsevier Science.

Pereira, C. J. B. (2016) Um Processo de Desenvolvimento de Software para/com Crianças: integração entre Engenharia de Software e Interação Humano-Computador, Unipampa.

Pressaman, R. S. (2006) Engenharia de Software, Mc Graw-Hill, 6ª. ed.

Puccini, A. M., Puccini, M., Chang, A. (2013) Acquiring educational access for neurodiverse learners through multisensory design principles. In: XII IDC, p. 455-458. ACM.

Ribeiro, S. S. (2016) Um método para o desenvolvimento de software por crianças com o ambiente Scratch, Unipampa.

Rivero, L., Conte, T. (2012) Using an Empirical Study to Evaluate the Feasibility of a New Usability Inspection Technique for Paper Based Prototypes of Web Applications. In: XXVI SBES, p. 81-90. SBC.

Rivero, L., Kawakami, G., Conte, T. (2014) Using a Controlled Experiment to Evaluate Usability Inspection Technologies for Improving the Quality of Mobile Web Applications Earlier in their Design. In: XXVIII SBES, p. 161-170. SBC.

Tolfo, C. (2013) Metodologia de mapeamento e modelagem de processos. p. 20-55 *apud* Segui, B. S. (2015) Desenvolvimento de sistema para solicitação de registro de marca de gado, Unipampa.

Sommerville, I. (2007) Engenharia de Software. Pearson Education, 8ª. ed.

Souza, A. T. (2016) Requisitos de segurança e privacidade no desenvolvimento de tecnologia web para/com crianças: integrando técnicas e práticas de Interação Humano-Computador e Engenharia de Software, Unipampa.

Valentim, N. M. C., Conte, T. (2014) Improving a usability inspection technique based on quantitative and qualitative analysis. In: XXVIII SBES, p. 171-180. SBC.

Desenvolvimento de Jogos Digitais do Gênero RPG com Adolescentes

Leonardo Anschau, Amanda Meincke Melo

Universidade Federal do Pampa (Unipampa) – Campus Alegrete
Av. Tiarajú, 810 – Ibirapuitã – 97.546-550 – Alegrete – RS – Brasil

leonardoanschau@gmail.com, amanda.melo@unipampa.edu.br

Abstract. *Teenagers are an important audience for digital games and they can also develop their own games. In a literature review on the development of digital games of the RPG genre, it was verified the lack of processes that considers its co-authorship by teens. This work, therefore, proposes a process to assist in the development of digital games of the RPG genre by teens, integrating methods and techniques from Software Engineering and Participatory Design. A qualitative research approach is adopted, by the accomplishment of an exploratory study in the Social Living and Strengthening of Bonds Service in Alegrete, south of Brazil.*

Resumo. *Adolescentes constituem um importante público de jogos digitais e podem, inclusive, desenvolver seus próprios jogos. Em uma revisão de literatura sobre o desenvolvimento de jogos digitais do gênero RPG, constatou-se a carência de processos que contemplem sua coautoria por adolescentes. Este trabalho, portanto, propõe um processo para auxiliar no desenvolvimento de jogos digitais do gênero RPG com a coautoria de adolescentes, integrando métodos e técnicas da Engenharia de Software com o Design Participativo. Adota-se uma abordagem qualitativa de pesquisa, pela realização de um estudo exploratório no Serviço de Convivência e Fortalecimento de Vínculos de Alegrete/RS.*

1. Introdução

Adolescentes, além de importantes usuários de jogos digitais, podem desenvolver seus próprios jogos e, assim, exercitar sua imaginação e expor suas ideias. A ferramenta RPG Maker [Dias 2017], em particular, facilita o desenvolvimento de jogos do gênero RPG (do inglês, *Role Playing Games*), sem exigir conhecimentos em linguagem de programação.

Através do Design Participativo, pode-se envolver adolescentes nesse desenvolvimento como parceiras de *design* [Druin 2002][Poole e Peyton 2013]. A adoção de um processo sistemático tem o potencial de qualificar o desenvolvimento de jogos digitais do gênero RPG com a colaboração de adolescentes. Este trabalho propõe a definição de um processo para o desenvolvimento de jogos digitais do gênero RPG com adolescentes, integrando métodos e técnicas da Engenharia de Software e do Design Participativo.

O texto está organizado como segue. A Seção 2 apresenta uma síntese da revisão de literatura. A Seção 3 apresenta o referencial teórico-metodológico da pesquisa. A Seção 4 sumariza seus aspectos metodológicos, além de apresentar o estudo realizado. A Seção 5 descreve a proposta de processo para o desenvolvimento de jogos digitais do

gênero RPG com a colaboração de adolescentes. A Seção 6 organiza algumas considerações finais sobre o desenvolvimento deste trabalho.

2. Revisão de Literatura

A fim de encontrar contribuições ao desenvolvimento de jogos digitais do gênero RPG com adolescentes, realizou-se uma revisão narrativa da literatura. Conduziram-se dois tipos de buscas: busca automática na biblioteca digital da ACM; e busca manual nos anais do Simpósio Brasileiro de Games e Entretenimento Digital (SBGames), publicados entre os anos 2010 a 2015. Para identificar e selecionar os artigos, levaram-se em conta as seguintes questões: Como jogos digitais, do gênero RPG, têm sido desenvolvidos? Há propostas de processos de Engenharia de Software para desenvolvê-los? Há experiências que envolvam a participação de crianças ou adolescentes em sua produção?

Ao todo, identificaram-se 22 trabalhos: quatorze na biblioteca digital da ACM e oito nos anais do SBGames. Esses tiveram seus títulos e resumos lidos. Então, selecionaram-se sete artigos que abordam o desenvolvimento de jogos do gênero RPG e/ou o desenvolvimento de jogos com a participação de crianças e/ou adolescentes. Esses, embora não apresentem como é realizado o desenvolvimento de um jogo do gênero RPG com adolescentes, contribuem ao desenvolvimento deste trabalho.

2.1. Desenvolvimento de jogos digitais do gênero RPG

Marques *et al.* (2012) observam conceitos que os jogos do gênero RPG possuem, independente das tecnologias utilizadas para desenvolvê-los, como personagens, diálogos, mapas e missões. Em seu trabalho, propuseram um *framework* aplicando o *Model-Driven Development* para criar jogos do gênero RPG reutilizando código. Os autores também destacam quatro características que todo jogo RPG possui: agentes, herói, espaço e objetivos. Agentes são os personagens do jogo. Herói é um personagem controlado pelo jogador e que possui ações. Espaço se refere ao ambiente do jogo, como mundo e cenários. Os objetivos são divididos em principal e secundário, sendo que o principal é o objetivo do jogo e os secundários são pequenos objetivos que auxiliam o jogador a alcançar o objetivo principal.

Meante e Silva (2014) apresentam um jogo RPG multijogador social denominado *Mychar*. Descrevem seus personagens, o ambiente do jogo, ações que podem ser realizadas, o mundo do jogo, além de ressaltarem a interação social do jogo, onde os personagens podem dialogar quando estão pertos uns dos outros. O jogo propõe uma dinâmica onde é possível conhecer novos lugares, realizar desafios, desbloquear locais secretos e conseguir equipamentos únicos.

Horta e Almeida (2012) propõem um jogo RPG para o ensino de Biologia. Adotam três abordagens distintas: ensino explícito, elemento dramático do “mestre” e história. O ensino explícito ocorre através de desafios (*quests*) e *quizzes*. O elemento dramático equivale ao “mestre” dos jogos RPG de mesa, responsável por conduzir e ditar as regras do jogo. Nesse jogo, o NPC (do inglês, *Non-Player-Character*) assume o papel de mestre e continua a história. A história do jogo é o contexto no qual o jogo acontece. Nesse caso, é o combate contra a organização LIX, que prejudica o meio ambiente.

Ao apresentar o desenvolvimento de um jogo RPG chamado *Hero Training*, Lindemann e Malheiros (2014) adotam a realidade virtual aumentada em dispositivos

móveis. Para os autores, alguns aspectos devem ser considerados por se tratar de um jogo do gênero RPG, como atributos de personagens (ex.: experiência, nível, vida, pontos de magia, força, velocidade, defesa, ataque mágico, defesa mágica e evolução no decorrer das batalhas). No decorrer do jogo, o personagem ganha experiência e, ao acumulá-la, sobe de nível e, consequentemente, seus atributos aumentam.

2.2. Desenvolvimento de jogos com crianças

Alves e Borges (2015) relatam em seu trabalho o jogo "Em busca da flor", que contou com a participação de uma criança de sete anos em seu processo de desenvolvimento. Nesse processo, aplicaram o Design Participativo através de um estudo de caso no qual a criança e o *designer* trabalharam colaborativamente. Inicialmente, a criança elaborou um jogo através de desenhos em papel. A partir disso, foram realizadas três atividades: descobrimento, avaliação e prototipação. O descobrimento permitiu ao *designer* coletar as principais características do jogo. Na avaliação, realizou-se uma entrevista aberta com a criança, procurando identificar suas ideias para o jogo. Por último, com a prototipação, foi elaborado o *design* do personagem principal, baseado no desenho da criança.

Good e Robertson (2004a) procuraram relacionar a criação de um jogo com o desenvolvimento de habilidades narrativas em crianças. Conduziram entrevistas com dois pares de crianças com dez anos de idade, sendo um par de meninos e outro par de meninas. Durante as entrevistas, conseguiram identificar as principais características que essas crianças valorizam em um jogo: propósito, metas, escolha de personagem, gráficos de boa qualidade, diálogos, narrações ao invés de grandes textos na tela, emoções e aventura, customização do personagem.

Procurando auxiliar crianças no desenvolvimento da narrativa, planejamento do enredo e criação de personagem em um jogo, os Good e Robertson (2004b) realizaram o *Workshop Game Maker*, no qual utilizaram ferramentas de desenvolvimento de jogos de nome *Neverwinter Nights*, que explora benefícios educacionais na autoria de jogos. Os autores realizaram atividades como discussão sobre jogos, *design* de personagem, *storyboards* com câmeras digitais e planejamento.

3. Referencial Teórico-Metodológico

Segundo Petrillo (2008), o processo mais utilizado na indústria de jogos digitais é baseado no modelo cascata do desenvolvimento de software tradicional. Ainda que seja bastante utilizado, esse modelo não lida bem com gestão de mudanças, devido ao seu fluxo linear, que pode ocasionar problemas no desenvolvimento [Barros 2007]. Por outro lado, o *Extreme Game Development* (XGD) é uma metodologia ágil proposta para o desenvolvimento de jogos digitais, que apoia a gestão de mudanças [Barros 2007]. O XGD, fundamentado no *Extreme Programming* (XP), tem como princípios: toda a equipe, *design* incremental, histórias de usuário, ciclo seminal (pequenos ciclos de execução), integração contínua, código compartilhado e reuniões em pé.

Barros (2007), ao discutir o trabalho de Sloper (2002), menciona cinco fases para o desenvolvimento de jogos digitais: Concepção, Pré-produção, Produção, Pós-produção e Pós-lançamento. É na etapa de Pré-produção que o *Game Design Document* (GDD) é gerado. Esse documento, segundo Petrillo (2008), apresenta as principais características para o desenvolvimento de um jogo digital. No GDD, detalham-se a mecânica do jogo, sua história, elementos importantes, cenários em que o jogo ocorre, ser-

vindo como referência para o desenvolvimento. Schuytema (2008) propõe que um GDD apresente os seguintes capítulos: visão geral, contexto do jogo, objetos essenciais, conflitos e soluções, inteligência artificial, fluxo do jogo, definições e referências.

Sommerville (2007) apresenta quatro etapas fundamentais a todos os processos de desenvolvimento de software: especificação, desenvolvimento, validação e evolução de software. A especificação reflete a Pré-produção, com a geração do *Game Design Document*. O desenvolvimento corresponde à fase de Produção, a validação à Pós-produção e a evolução ao Pós-lançamento.

O Design Participativo envolve a participação ativa de usuários no desenvolvimento de um determinado produto [Muller *et al.* 1997]. Adolescentes, assim como crianças, podem atuar como parceiras de *design* e, desse modo, contribuir diretamente no processo de desenvolvimento de tecnologias digitais [Alves e Borges 2015][Druin 2002][Poole e Peyton 2013].

Poole e Peyton (2013) discutem o *design* da interação com adolescentes, contemplando aspectos éticos. Chamam atenção para a necessidade de consentimento e concordância, além de possíveis desequilíbrios de poder entre adolescentes e adultos, que precisam ser amortizados. Nessa perspectiva, McNally *et al.* (2016) problematizam questões éticas na realização de pesquisas com crianças, entre elas: a necessidade de consentimento dos pais e concordância das próprias crianças a partir da compreensão da proposta, a substituição de nomes por pseudônimos e a realização do registro fotográfico pelas costas.

4. Metodologia

Optou-se por uma abordagem qualitativa de pesquisa, pela realização de um estudo exploratório, visando a aumentar a experiência sobre o desenvolvimento de jogos do gênero RPG com adolescentes. A pesquisa teve início com uma revisão narrativa de literatura, buscando-se encontrar contribuições para o seu desenvolvimento e delimitar melhor as contribuições da pesquisa para a produção de jogos do gênero RPG.

A partir da revisão de literatura, esboçou-se um método para o desenvolvimento de um jogo do gênero do RPG com a participação de adolescentes. Esse método foi aplicado e ajustado conforme o andamento do estudo, desenvolvido no Centro de Convivência e Fortalecimento de Vínculos, em Alegrete/RS. Para o desenvolvimento do jogo, adotou-se a ferramenta RPG Maker [Dias 2017], voltada à criação de jogos do gênero RPG. Essa ferramenta não exige conhecimentos de programação de computadores, possibilitando enfatizar a investigação no processo de desenvolvimento.

4.1. Ambiente de pesquisa e participantes

O jogo foi desenvolvido no Centro de Convivência e Fortalecimento de Vínculos, em Alegrete/RS. Trata-se de um programa do Governo Federal, de caráter socioeducativo, que contempla adolescentes, entre outros públicos.

Considerando a recomendação de Mazzone *et al.* (2008), a equipe de participantes foi formada por quatro adolescentes, juntamente com o pesquisador. Segundo os autores, um grupo pequeno é melhor para os participantes expressarem suas ideias e, assim, os participantes se sentem valorizados em desempenhar suas atividades [Mazzo-

ne *et al.* 2008]. Os adolescentes que participaram do projeto tinham entre de 12 e 17 anos de idade. Esses possuíam alguma experiência com informática básica.

4.2. Desenvolvimento de um jogo digital do gênero RPG com adolescentes

Para desenvolver um jogo digital do gênero RPG com adolescentes, organizou-se uma série de atividades em quatro etapas: Ambientação, Engenharia de Requisitos, Desenvolvimento e Evolução. A cada encontro, os participantes tinham a oportunidade de avaliar as atividades realizadas.

Na etapa "Ambientação", a proposta foi apresentada aos adolescentes, o consentimento de participação foi obtido dos responsáveis e os adolescentes, que concordaram em colaborar, participaram de uma oficina de construção de RPG digital com a ferramenta RPG Maker. No período de três semanas, em oito encontros de 2h cada, os recursos da ferramenta foram apresentados aos adolescentes, que também tiveram a oportunidade de utilizá-la, seguindo as instruções do pesquisador.

A etapa "Engenharia de Requisitos" teve início com uma tempestade de ideias, com o objetivo de explorar temas para o desenvolvimento do jogo. Com o apoio de *post-its*, foram gerados vinte e dois temas. A partir de uma votação, escolheu-se o tema "Guerra medieval: o reino perdido". A partir desse tema, em três encontros de 2h a 3h cada, foram definidas as características para o jogo RPG, com auxílio de um documento gerado com características identificadas na revisão de literatura, como agentes e heróis, espaço, objetivos, entre outras. Essas características contribuíram à organização do GDD, elaborado com apoio de uma adaptação do modelo proposto por Schuytema (2008), contendo os seguintes capítulos: visão geral, contexto do jogo, elementos essenciais, conflitos e soluções, *quests*, locais secretos e referências.

O GDD foi construído em cinco encontros de 1h30min a 3h cada, com intensa colaboração entre os participantes. Para levantar a maioria das informações, realizaram-se tempestades de ideias com os envolvidos dispostos em um círculo em torno do pesquisador. Na elaboração da história do jogo, em particular, cada participante escreveu a sua história, respondendo à questão "Com base nos objetivos secundários, como você imagina a história? Conte-a.". A partir dessas histórias, o pesquisador elaborou a história do jogo que foi incorporada ao GDD.

Então, o documento gerado foi validado em três encontros com duração de 3h a 3h30min. A validação, além da leitura de todo o GDD, contou com o apoio de um questionário, contendo as seguintes questões: (1) O GDD reflete o que foi idealizado pelo grupo? (2) Tudo o que é desejado está documentado? (3) Todos os requisitos podem ser implementados com a ferramenta escolhida? (4) O documento apresenta as informações de forma objetiva, de modo que possam ser verificadas se foram implementadas?

Na etapa "Desenvolvimento", com auxílio da ferramenta RPG Maker, o jogo propriamente dito foi criado. Essa etapa foi realizada em 20 encontros com os adolescentes, com a duração variando de 1h a 4h, além de atividades desempenhadas pelo pesquisador. Iniciou-se pela criação dos personagens na ferramenta RPG Maker. Então, foram criados o mapa do mundo e os cenários, a partir de ideias geradas com apoio da técnica BrainDraw [Muller *et al.* 1997]. Inicialmente, pretendia-se criar os cenários iterativamente, mas optou-se por criá-los todos com auxílio da técnica BrainDraw para, então, desenvolvê-los iterativamente na ferramenta. Desse modo, seria possível ter uma visão geral dos cenários, além de favorecer a coesão entre eles.

Definiram-se três iterações para o desenvolvimento e verificação dos cenários – as duas primeiras envolvendo cada qual o desenvolvimento de dois cenários e a última envolvendo o desenvolvimento de três cenários. Na primeira iteração, os cenários foram desenvolvidos na ferramenta pelos adolescentes e verificados pelo pesquisador. Na segunda e na terceira iterações, os cenários foram desenvolvidos pelo pesquisador e verificados pelos adolescentes. Quando identificadas inconsistências, os adolescentes tinham liberdade para corrigi-las com a supervisão do pesquisador. Para encerrar cada iteração, era realizada uma reunião de acompanhamento.

Ao final de todas as iterações, o jogo foi verificado, corrigido e disponibilizado para uso por outros adolescentes que frequentavam o Centro de Convivência e Fortalecimento de Vínculos. Esse uso foi observado pelo grupo que o desenvolveu. Finalmente, foi publicado em uma rede de jogos RPG. Seu uso, então, pode ser acompanhado em trabalhos futuros.

5. Resultados

Esta seção apresenta um modelo de processo para o desenvolvimento de jogos do gênero RPG com a colaboração de adolescentes, proposto a partir da experiência desenvolvida no estudo exploratório.

5.1. Ambientação

Nesta etapa, no passo “Conhecer o Projeto”, apresenta-se o projeto aos participantes, para que tenham conhecimento de seu objetivo e a natureza de sua colaboração, deixando-se clara a natureza voluntária da participação.

No passo “Solicitar consentimento ao responsável”, entrega-se um Termo de Consentimento Livre e Esclarecido aos responsáveis para que tomem ciência do projeto, saibam da natureza voluntária da participação e assinem consentindo a participação do adolescente sob sua responsabilidade. No passo “Concordar com a participação”, os próprios adolescentes indicam concordância em participar do projeto.

No passo “Realizar oficina de construção de RPG digital”, realiza-se uma oficina de RPG para que os participantes compreendam o que pode ser feito com a ferramenta escolhida e como utilizá-la para criar cenários, personagens e tudo o que for necessário para o desenvolvimento do jogo RPG.

5.2. Engenharia de Requisitos

No passo “Brainstorming”, escolhe-se o tema do jogo a partir da livre geração de ideias entre os participantes, sem julgamentos ou críticas. Com a mediação do Engenheiro de Software, chega-se em um consenso.

Escolhido o tema, no passo “Definir Característica RPG”, deve-se definir e registrar as características do jogo como agentes e herói, espaço (ex.: mundo e cenários), objetivo principal e objetivos secundários, entre outras. Então, inicia-se o passo “Elaborar GDD”, gerando o artefato que serve de guia para o desenvolvimento do jogo. Nesse passo, o GDD deve ser preenchido pelo Engenheiro de Software com a colaboração dos adolescentes.

Uma vez elaborado o GDD, no passo “Validar GDD”, o artefato deve ser validado com todo o grupo, a fim de identificar se está tudo conforme o planejado. Caso

esteja, segue-se à etapa “Desenvolvimento”. Caso contrário, retorna-se ao passo “Elaborar GDD”.

5.3. Desenvolvimento

Baseando-se no *Extreme Game Development*, esta etapa considera princípios como: comunicação acima de tudo, execução do projeto em pequenos ciclos, integração contínua e pequenas reuniões.

No passo “Elaborar personagens na ferramenta”, elaboram-se todos os personagens documentados no GDD. Em seguida, no passo “Braindrawing do mundo”, elabora-se um protótipo de baixa fidelidade do mundo. Então, no passo “Elaborar mundo na ferramenta”, o mundo deve ser desenhado em alta fidelidade.

No passo “Braindrawing de todos os cenários”, geram-se protótipos de baixa fidelidade dos cenários identificados no mundo. Esses cenários, no passo “Definir iterações”, são alocados em ciclos iterativos de desenvolvimento, que envolve os passos “Desenvolver cenários”, “Verificar e corrigir cenários” e “Realizar reunião de acompanhamento”.

Uma vez que todas as iterações tenham sido realizadas, parte-se para o passo “Verificar e validar jogo”. Nesse passo, o jogo é verificado como um todo e validado para conferir se está de acordo com o que foi descrito no GDD. Se necessário, correções são realizadas no passo “Realizar adequações”. Se correções forem realizadas, retorna-se ao passo “Verificar e validar jogo”. Senão, segue-se para o passo seguinte.

Finalmente, no passo “Observar experiência de uso”, libera-se uma versão estável do jogo para um grupo de pessoas que não participaram de seu desenvolvimento. Seu uso é observado.

5.4. Evolução

No passo “Publicar o jogo”, deve-se publicar uma versão do jogo. A partir disso, acompanha-se seu uso através do passo “Acompanhar uso do jogo”, analisando os *feedbacks* dos jogadores para possíveis melhorias.

6. Conclusão

Através da abordagem do Design Participativo no desenvolvimento de um jogo digital do gênero RPG com adolescentes, os participantes se mostraram bastante motivados a colaborar ao perceberem que suas contribuições eram refletidas de forma direta nos produtos gerados a cada atividade do processo de desenvolvimento delineado.

A organização das atividades em etapas e a adoção de ciclos iterativos de desenvolvimento dos cenários facilitaram o planejamento e a gestão de riscos – estes provocados por ausências em função de imprevistos e problemas de saúde reportados pelos participantes. Favoreceram a redistribuição de tarefas entre pesquisador e participantes. A documentação elaborada durante a etapa “Engenharia de Requisitos”, embora bastante extensa, orientou o processo de desenvolvimento do jogo.

Tem-se como principal contribuição um processo que pode servir de referência para novas experiências de desenvolvimento de jogos do gênero RPG com o envolvimento direto de adolescentes. Como sugestão de trabalho futuro está a abordagem da evolução de jogos digitais do gênero RPG com adolescentes.

7. Referências

Alves, A. G., Borges, T. L. (2015) Criança co-criadora de jogos digitais: um estudo de caso com aplicação da abordagem do Design Participativo. In: SBGames'2015, p. 396-403, SBC.

Barros, R. L. B. (2007) Análise de Metodologias de Desenvolvimento de Software aplicadas ao Desenvolvimento de Jogos Eletrônicos, UFPE.

Dias, R. RPG Maker: O Guia Completo – Produção de Jogos, http://www.producaodejogos.com/rpg-maker/, ago. 2017.

Druin, A. (2002) The role of children in the design of new technology. *Behaviour and information technology*, v. 21, n. 1, p. 1-25. Taylor & Francis.

Good, J., Robertson, J. (2004a) Computer Games Authored by Children: a multi-perspective evaluation. In: IDC'04, p. 123–124, ACM Press.

Good, J. Good, Robertson, J. (2004b) Children's narrative development through computer game authoring. In: IDC'04, p. 57–64, ACM Press.

Horta, A., Almeida, L. F. (2012) Projeto Alius Educare: RPG para educação e conscientização ambiental. In: SBGames'2012, p. 63–69, SBC.

Lindemann, J. L., Malheiros, M. G. (2014) Desenvolvimento de RPG Usando Realidade Aumentada em Dispositivos Móveis. In: SBGames'2014, p. 993–996, SBC.

Marques, E., Balegas, V., Barroca, B. F. *et al.* (2012) The RPG DSL: a case study of language engineering using MDD for generating RPG games for mobile phones. In: DSM'12, p. 13–18, ACM Press.

McNally, B., Guha, M. L., Mauriello, M. L. *et al.* (2016) Children's Perspectives on Ethical Issues Surrounding Their Past Involvement on a Participatory Design Team. In: CHI'16, p. 3595–3606, ACM Press.

Mazzone, E., Read, J. C., Beale, R. (2008) Design with and for disaffected teenagers. In: NordiCHI'08, p. 290–297, ACM Press.

Meante, E., Silva, C. F. (2014) MyChar: um jogo para RPG social. In: SBGames'2014, p. 312–315, SBC.

Muller, M. J., Haslwanter, J. H., Dayton, T. (1997) "Participatory Practices in the Software Lifecycle", Handbook of Human-Computer Interaction, Elsevier Science.

Petrillo, F. S. (2008) Práticas Ágeis no Processo de Desenvolvimento de Jogos Eletrônicos, UFRGS.

Poole, E. S., Peyton, T (2013). Interaction design research with adolescents: methodological challenges and best practices. In: IDC'13, p. 211–217, ACM Press.

Schuytema, P. (2008) "O documento de design", Design de Games: uma abordagem prática, Cengage Learning.

Sloper, T. (2002) "Following Up After the Game in Released: It's not Over when it's Over", Game Design Perspectives, Cengage Learning. *apud* Barros (2007).

Sommerville, I. (2007) Engenharia de Software. Pearson Education, 8ª. ed.

Um Editor para a Linguagem de Especificação de Requisitos RELAX

Luiz Paulo Franz[1], Gabriel B. Moro[2], Rafael Torres[1], João Pablo S. da Silva[1]

[1]Universidade Federal do Pampa (UNIPAMPA)
Av.Tiarajú, 810, Alegrete, RS, Brasil

[2]Universidade Federal do Rio Grande do Sul (UFRGS)
Av. Bento Gonçalves, 9500, Porto Alegre, RS, Brasil

{luizpaulofranz, gabrielbronzattimoro.es, torresrafa22, jpabloss}@gmail.com

Abstract. *Self-adaptive systems have unique characteristics that differentiate them from other types of systems. As the main one, we can highlight the situations of uncertainty that such systems must face. Because of this, the usual requirements engineering processes are not satisfactory in this context. One of the proposals to assist in the engineering of requirements for self-adaptive systems is the RELAX language, which aims to highlight the uncertainties inherent in this type of system. In this article, we present the development of a tool that supports the writing and editing of RELAX requirements, exploring its syntax and semantics, and facilitating the use of the language. As a result, we get a functional editor based on the eclipse platform.*

Resumo. *Sistemas autoadaptativos apresentam características únicas, que os diferenciam de outros tipos de sistemas. Como a principal delas, podemos destacar as situações de incerteza que tais sistemas devem enfrentar. Por conta disso, os processos usuais de engenharia de requisitos não são satisfatórios nesse contexto. Uma das propostas para auxiliar na engenharia de requisitos para sistemas autoadaptativos é a linguagem RELAX, que tem como objetivo evidenciar as incertezas inerentes à esse tipo de sistema. Neste artigo, apresentamos o desenvolvimento de uma ferramenta que da suporte à escrita e edição de requisitos RELAX, explorando sua sintaxe e semântica e facilitando o uso da linguagem. Como resultado, obtemos um editor funcional baseado na plataforma eclipse.*

1. Introdução

Sistemas autoadaptativos podem ser definidos como softwares capazes avaliar e mudar seu comportamento em tempo de execução com base nos estímulos que recebem do ambiente e seu estado atual [Macías-Escrivá et al. 2013]. [Whittle et al. 2010] comentam que esses sistemas atuam em condições e contextos diversos, o que dificulta a previsão dos estados pelos quais esse software vai passar ao longo de seu ciclo de vida.

Engenharia de requisitos é um conjunto de processos que tem como objetivo elicitar e avaliar os requisitos que motivam um projeto. De acordo com

[Sommerville et al. 2007], através da engenharia de requisitos, é possivel manter um artefato de documentalção de requisitos, que deve ser seguido pela equipe de desenvolvimento durante a implantação do sistema.

Considerando as situações de imprevisibilidade que os sistemas autoadaptativos enfrentam, fica claro que as abordagens convencionais de engenharia de requisitos não são adequadas para lidar com tais sistemas. Tendo isso em vista, [Whittle et al. 2010] propuseram a linguagem RELAX, com o objetivo de elicitar os requisitos de um sistema autoadaptativo evidenciando suas características de incerteza.

Neste trabalho, temos como objetivo apresentar uma ferramenta para especificação de requisitos que dê suporte a linguagem RELAX, explorando seus aspectos sintáticos e semânticos, tornando seu uso mais transparente.

Este artigo está organizado da seguinte maneira: no capítulo 2 apresentamos os trabalhos relacionados, no capítulo 3 apresentamos em detalhes a linguagem RELAX; no capítulo 4 realizamos a descrição do desenvolvimento da ferramenta; no capítulo 5, a validação da ferramenta, e por fim, no capítulo 6, a conclusão do trabalho.

2. Trabalhos Relacionados

Nesta seção, falaremos de trabalhos que descrevem ferramentas de apoio à engenharia de requisitos para sistemas autoadaptativos propostas por outros autores.

[Ali et al. 2013] propõem um mecanismo de mapeamento da especificação de requisitos em um modelo de metas para identificar inconsistências e requisitos conflitantes de acordo com contexto. Esse mecanismo é acrescentado como funcionalidade para a ferramenta CASE RE-*Context*, desenvolvida pelos autores em um trabalho anterior.

[Hussein et al. 2013] apresentam uma ferramenta de validação de requisitos automatizada baseada em cenários através do processamento da especificação das variantes dos requisitos levantados (características, estado, e assim por diante). A partir disso, a ferramenta gera automaticamente scripts de acordo com o cenário de entrada e os verifica. Após isso, as inconscistências encontradas são apresentadas.

[Fredericks et al. 2014] relatam uma abordagem que automatiza a análise de modelos objetivos e utiliza como entrada modelos de metas construído em KAOS (Keep All Objectives Satisfied). A partir disso a ferramenta desenvolvida pelos autores gera possíveis soluções em modelos Relax, onde contém operadores da própria linguagem como também funções que caracterizam o objetivo definido no modelo de entrada.

[Moro 2015] apresenta a primeira versão da RelaxEditor, com a proposta de uma ferramenta que dê suporte à requisitos RELAX. No entanto, optamos por re-desenvolver a ferramenta utilizando uma versão mais atual da plataforma Eclipse, que oferece uma série de recursos: *syntax high-light*, *auto-complete*, *hovering*, entre outros.

3. Linguagem RELAX

Apresentada por [Whittle et al. 2010], a linguagem RELAX é uma linguagem natural estruturada que tem como objetivo evidenciar características de incerteza, que por sua vez, são inerentes à sistemas autoadaptativos. A RELAX utiliza operadores para organizar o requisito de forma mais formal, porém ainda com trechos em linguagem natural. A

classificamos, portanto, como linguagem semi-formal. Os operadores RELAX são apresentados na Tabela 1 e a semântica na Tabela 2.

Table 1. O vocabulário da Linguagem RELAX.

Operadores	Descrição
Operadores Modais	
SHALL	Um requisito deve ser fornecido/mantido.
MAY... OR MAY	Um requisito especifica uma ou mais alternativas.
Operadores Temporais	
EVENTUALLY	Um requisito deve ser fornecido/mantido eventualmente.
UNTIL	Um requisito deve ser fornecido/mantido até uma posição futura.
[BEFORE — AFTER]	Um requisito deve ser fornecido/mantido antes ou depois de um evento em particular.
IN	Um requisito deve ser fornecido/mantido em/durante um particular intervalo de tempo.
AS [EARLY — LATE] AS POSSIBLE	Um requisito especifica algo que deveria ser fornecido/mantido tão cedo quanto possível ou deveria ser postergado por tanto tempo quanto possível.
AS CLOSE AS POSSIBLE [frequência]	Um requisito especifica algo que acontece repetidamente mas a frequência pode ser relaxada tão próxima quanto possível de uma frequência.
Operadores Ordinais	
AS CLOSE AS POSSIBLE [quantidade]	Um requisito especifica uma quantidade contável, mas o número exato pode ser relaxado tão próxima quanto possível de uma quantidade.
AS [MANY — FEW] AS POSSIBLE	Um requisito especifica uma quantidade contável, mas o número exato pode ser relaxado o maior número possível ou o menor número possível.
Fatores de Incerteza	
ENV	Define um conjunto de propriedades que definem o ambiente do sistema.
MON	Define um conjunto de propriedades a serem monitoradas pelo sistema.
REL	Define a relação entre as propriedades de ENV e MON.
DEP	Identifica as dependências entre os requisitos relaxados e invariantes.

O exemplo apresentado na Tabela 3 ilustra um requisito em RELAX. Podemos perceber nele as características básicas da linguagem, tais como operadores e fatores de incerteza. Também podemos perceber que a especificação do requisito se torna mais expressiva, mesmo sendo escrita em uma estrutura declarativa e em linguagem natural.

4. Desenvolvimento do RelaxEditor

Na Figura 1, apresentamos um *screenshot* da tela do RelaxEditor. Por ter sido desenvolvido na plataforma Eclipse, a interface se assemelha à IDE Eclipse. Na imagem temos os arquivos no formato *.relax* no canto esquerdo, onde são escritos os requisitos, no centro o conteúdo do arquivo (editor), e a direita, a expressão resultante do requisito *tractor01*.

4.1. Xtext

Para realizarmos o desenvolvimento da ferramenta, optamos pela utilização do Xtext. Xtext é um framework pertencente à plataforma Eclipse para o desenvolvimento de *Domain Specific Language* (DSL), bem como linguagens de propósito geral

Table 2. A semântica da Linguagem RELAX.

Expressão RELAX	Formalização FBTL
SHALL ϕ	AG ϕ
MAY ϕ1 OR MAY ϕ2	AG (ϕ_1 ou ϕ_2)
EVENTUALLY ϕ	A F ϕ
ϕ_1 UNTIL ϕ_2	A(ϕ_1 UNTIL ϕ_2)
BEFORE *e* ϕ	A$\chi <_{e_d}\phi$ onde e_d é a duração até a próxima ocorrência de e
AFTER *e* ϕ	A$\chi >_{e_d}\phi$
IN t ϕ	(AFTER $t_{começo}$ ϕ $\bigwedge$ BEFORE t_{fim} ϕ) onde $t_{começo}$, t_{fim}, são eventos que denotam o início e o fim do intervalo *t*, respectivamente
AS EARLY AS POSSIBLE ϕ	A$\chi_{\geq_d}\phi$ onde *d* é a duração difusa definida de tal forma que seus membros de função tem o seu máximo em 0 (ou seja, M(0)= 1) e diminui continuamente para os valores >0
AS LATE AS POSSIBLE ϕ	A$\chi_{\geq_d}\phi$ onde *d* é a duração difusa definida de tal forma que seus membros de função tem o seu valor mínimo em 0 (ou seja, M(0)= 0) e aumenta continuamente para os valores >0
AS CLOSE AS POSSIBLE TO *f* ϕ	A ($\chi=_d\phi \bigwedge \chi=_{2d}\phi \bigwedge \chi=_{3d}\phi \bigwedge$...) onde *d* é a duração difusa definida de tal modo que a sua função de adesão tem o seu valor máximo, no período definido por *f* (ou seja, M(d)= M(2d)=...=1) e diminui continuamente para valores inferiores e superiores a *d* (e 2*d*, ...)
AS CLOSE AS POSSIBLE TO *q* ϕ	AF(($\Delta(\phi)$-q) ϵ S) pertence a S, onde S é um conjunto fuzzy cuja função de composição tem um valor no zero (M(0)= 1) e diminui continuamente em torno de zero. $\Delta(\phi)$ "conta"o quantificável, que será comparado com *q*
AS MANY AS POSSIBLE ϕ	AF ($\Delta(\phi)$ ϵ S), onde S é um conjunto fuzzy cuja função de pertinência tem 0 valor no zero(M(0)= 0) e aumenta continuamente em torno de zero
AS FEW AS POSSIBLE ϕ	AF ($\Delta(\phi)$ ϵ S), onde S é um conjunto fuzzy cuja função de pertinência tem um valor no zero(M(0)= 1) e diminui continuamente em torno de zero

Table 3. Exemplo de requisito escrito usando a Linguagem RELAX

O refrigerador DEVE gerenciar o plano de dieta de forma que esse mantenha o consumo de calorias TÃO PRÓXIMO QUANTO POSSÍVEL do consumo diário ideal.

ENV:	Alimentos.
MON:	Leitores RFID; Sensores de Peso.
REL:	Leitores RFID monitoram os Alimentos para prover informações sobre propriedades calóricas; Sensores de Peso monitoram os Alimentos para prover informação sobre consumo de calorias.
DEP:	Não se aplica.

[Foundation 2017]. Além da facilidade que o framework oferece para desenvolver linguagens, ele ainda oferece recursos avançados de forma simples, sendo alguns deles: Syntax Highlight, evidenciando palavras reservadas, comentários e erros de sintaxe; Con-

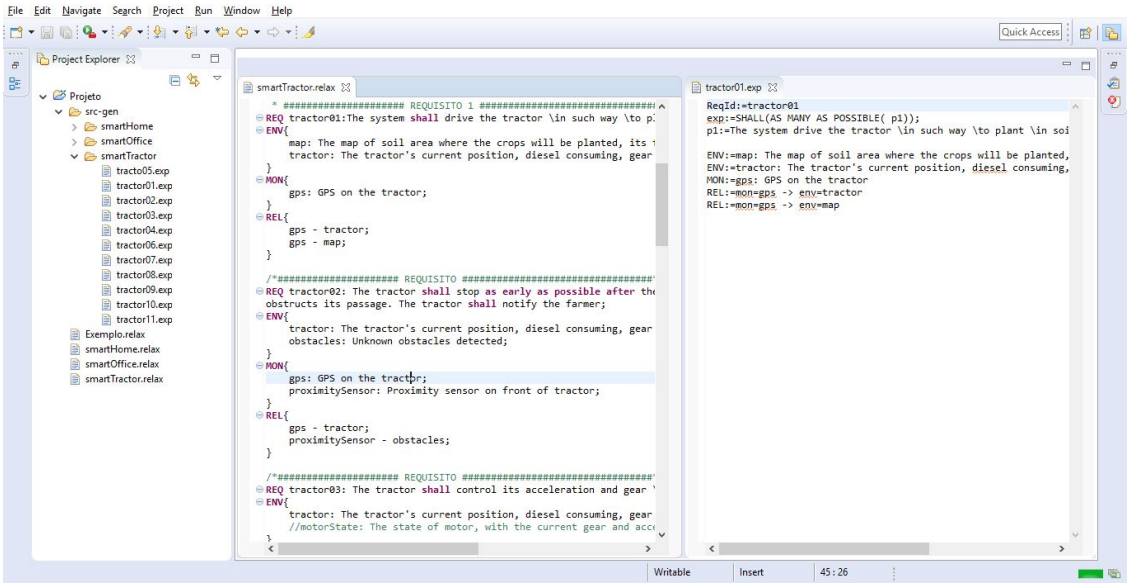


Figure 1. Screenshot da tela do RelaxEditor

tent assist, opção de auto-complete e Hyperlinking, possibilitando saltar para a declaração de algum bloco ou variável.

4.2. Desenvolvimento da gramática

Para o desenvolvimento da ferramenta, estendemos a gramática original da Linguagem RELAX e adicionamos alguns operadores: o operador REQ para demarcar o início de um novo requisito; os operadores ENV, MON, REL e DEP que, por mais que tenham sido apresentadas no artigo de [Whittle et al. 2010], não haviam sido formalmente definidos.

Como terminal da Linguagem RELAX utilizado para o desenvolvimento da ferramenta, temos o conjunto T. Nele, temos os operadores da linguagem definidos, o terminal ID, que representa os caracteres alfanuméricos para os trechos em linguagem natural, e o simbolo ϕ representando uma expressão RELAX.

T = {REQ, ID, ϕ, true, false, SHALL, MAY... OR MAY..., EVENTUALLY, UNTIL, BEFORE, AFTER, IN, AS CLOSE AS POSSIBLE TO, AS CLOSE AS POSSIBLE TO, AS {EARLY; LATE; MANY; FEW} AS POSSIBLE, ENV, MONS, RELS, DEP}

Como variáveis da gramática, temos V = {ϕ, ϕ1, ϕ2, e, f, q, t, p}, onde ϕ representa uma expressão RELAX (ou seja, tratando a recursividade da linguagem), *e* representa um evento, *f* representa frequência, *q* representa quantidade, *t* representa tempo e *p* representa uma frase em linguagem natural.

Para a regra maior da linguagem, definimos que ela será composta pelo operador REQ (ou req), um identificador único para o requisito, uma sentença RELAX, suas propriedades, e encerrará com ponto e virgula. Temos, portanto, a seguinte regra de construção:
P = {S→REQ p: ϕ ; Prop | true | false,
Prop→ ENV | MON | ENV | ENV MON REL | ENV MON REL DEP | DEP | ENV DEP | MON DEP | ε,
ENV→ ENV { (p : p;)$^+$ } ,

MON→ MON { (p : p;)$^+$ } ,
DEP→ DEP: p; | (p,)$^+$ p; ,
REL→ REL { (p : p;)$^+$ }}

4.3. Geração de expressões

Segue um trecho de código para exemplificar como o Xtext nos permite manipular informações:

```
def dispatch compileRest(ShallOperator s) {
this.operators += "SHALL("
s.elements.compileGeneral
}

def dispatch compileRest(EventuallyOperator e) {
this.operators += "EVENTUALLY("
e.element.compileGeneral
}

def dispatch compileRest(BeforeOperator b) {
this.operators += "BEFORE("
this.e = b.event
}
```

No trecho acima, os atributos *elements* e *event* que foram definidos nas regras da gramática representam o texto em linguagem natural que vem logo após os operadores, e a partir daí, montamos as expressões da maneira que desejamos.

A seguir, apresentamos um exemplo de requisito escrito utilizando a Linguagem RELAX e depois a expressão que é gerada a partir dele.

```
REQ prenat01 : The smartphone SHALL send a help message \to
another authorized user when the pregnant woman is \in a
hospital ;
ENV { currentPosition : Current position of the pregnant woman ; }
MON {gps: The pregnant woman 's smartphone GPS ;}
REL {gps - currentPosition ;}
```

Expressão gerada:

```
ReqId := prenat01
exp := SHALL ( p1);
p1 := The smartphone send a help message \to another authorized
user when the pregnant woman is \in a hospital
ENV := currentPosition : Current position of the pregnant woman
MON := gps: The pregnant woman 's smartphone GPS
REL := mon=gps -> env= currentPosition
```

5. Testes

Para realizarmos os testes da ferramenta, desenvolvemos testes baseados em contextos distintos e cobrindo várias das possibilidades oferecidas pela linguagem RELAX, bem

Table 4. Resultado dos testes

Cenário de teste	Testes	Erros
SmartCampus	12	1
SmartCar	12	1
SmartHome	6	0
SmartTractor	11	0
Total	41	2

como todos os operadores que ela apresenta junto às suas propriedades. Na Tabela 4 podemos observar os resultados dos testes elaborados.

Levando em conta os casos testados, obtivemos uma baixa quantidade de erros, sendo que todos eles estão relacionados ao operador *MAY... OR MAY...* .

Os requisitos que apresentaram mau funcionamento são apresentados abaixo.

```
REQ campus02: The app MAY notify students and teachers when
a class \or meeting time approaches OR MAY register
the student attendance \in case he's already \in
the classroom;

ENV{smartphone : The student's smartphone that is running
the app;
people : Students and teachers;
}
MON{mainSystem : the SmartCampus main system;
roomSensor : The sensor that is going \to monitor \and
    identify people;
}
REL{mainSystem - smartphone;
roomSensor - people;
}

REQ car02: The Smartcar MAY turn on air conditioner OR MAY
open the windows;

ENV{temperature: Smartcar's internal temperature;}

MON{temperatureSensor: Thermometer \to monitoring the
Smartcar's temperature;}

REL{temperatureSensor-temperature;}
```

Ambos os erros ocorrem por erro de sintaxe, ressaltados pelo recurso de *syntax highlight*, como pode ser visto na Figura 2.

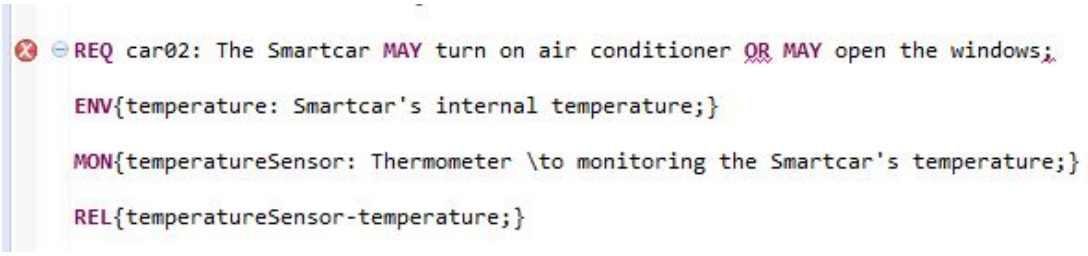


Figure 2. Screenshot de um erro de sintaxe sendo apontado pela ferramenta.

6. Conclusão

O principal objetivo deste trabalho é criar uma ferramenta para elicitação de requisitos com suporte à linguagem RELAX proposta por [Whittle et al. 2010]. Baseado no que foi apresentado, podemos concluir que cumprimos o objetivo de produzir tal ferramenta. Com ela podemos escrever sentenças complexas utilizando os operadores RELAX e extrair as expressões formais delas. Entretanto, a ferramenta ainda apresenta limitações relacionadas ao operador *MAY... OR MAY...*. Como trabalhos futuros, propomos a melhoria da ferramenta em relação ao operador que apresenta problema e ampliar ainda mais os cenários de teste para aumentar a confiabilidade na ferramenta.

References

[Ali et al. 2013] Ali, R., Dalpiaz, F., and Giorgini, P. (2013). Reasoning with contextual requirements: Detecting inconsistency and conflicts. *Information and Software Technology*, 55(1):35–57.

[Foundation 2017] Foundation, E. (2017). Xtext - reference documentation. Disponível em: `https://eclipse.org/Xtext/documentation/index.html`. Acesso em: 02 de junho de 2017.

[Fredericks et al. 2014] Fredericks, E. M., DeVries, B., and Cheng, B. H. (2014). Autorelax: automatically relaxing a goal model to address uncertainty. *Empirical Software Engineering*, 19(5):1466–1501.

[Hussein et al. 2013] Hussein, M., Han, J., Yu, J., and Colman, A. (2013). Scenario-based validation of requirements for context-aware adaptive services. In *Web Services (ICWS), 2013 IEEE 20th International Conference on*, pages 348–355. IEEE.

[Macías-Escrivá et al. 2013] Macías-Escrivá, F. D., Haber, R., Del Toro, R., and Hernandez, V. (2013). Self-adaptive systems: A survey of current approaches, research challenges and applications. *Expert Systems with Applications*, 40(18):7267–7279.

[Moro 2015] Moro, G. B. (2015). Uma ferramenta de apoio à especificação de requisitos para sistemas autoadaptativos.

[Sommerville et al. 2007] Sommerville, I. et al. (2007). *Engenharia de software*, volume 8. Addison Wesley São Paulo.

[Whittle et al. 2010] Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H., and Bruel, J.-M. (2010). Relax: a language to address uncertainty in self-adaptive systems requirement. *Requirements Engineering*, 15(2):177–196.

Multivocal Literature Review on User Story Models for COSMIC Sizing

Miguel Ecar[1], **João Pablo Silva da Silva**[1]

[1]Software Engineering – Federal University of Pampa - UNIPAMPA
Alegrete – RS – Brazil

miguel@ecarsm.com, joaosilva@unipampa.edu.br

Abstract. *User Story is a technique widely used in Agile development. It is characterized as short and high level description of required functionality, written in customer language during very early stage of requirement gathering. COSMIC method is a second generation technique of function size measurement. The requirement estimation precision in COSMIC is directly proportional to requirement detailing level. The current user story writing models may have lacks of important information for COSMIC measurement purposes. Considering this, the paper presents a multivocal literature review which presents current user story models found in traditional and gray literature. Furthermore, we analyze the effectiveness of this models for COSMIC sizing purposes.*

1. Introduction

Agile methods were born from the need to smooth the heavyweight plan-based methods used in large-scale software-development projects [Abran et al. 2004]. Many agile methods are available in the literature, popular approaches are Rapid Application Development (RAD)[Martin 1991], Scrum[Schwaber and Sutherland 2011], Feature-Driven Development (FDD)[Palmer and Felsing 2001], and eXtreme Programming (XP) [Maurer and Martel 2002]. This last in particular introduced the idea of User Story (US) that is basically use cases, which briefly capture functional requirements [Maurer and Martel 2002]. Ideally, it should take a team one to five engineering weeks of effort to implement the tasks each story implies. Stories should be testable [Maurer and Martel 2002].

User Stories have gained popularity among agile approaches being one of the main techniques used when the subject is requirements engineering in agile environments. There is a common spread template to write US, however a number of different extension have been appeared adding or suppressing information depending on the application context.

One of the characteristic of US is the, besides being testable, user stories should be measurable. In this work, we considered COSMIC method as Functional Size Measurement (FSM) technique for sizing estimation of US. COSMIC method has being called by its owners as second generation of FSM techniques [Abran et al. 2015]. The COSMIC method is perfectly suited for measuring software evolving through iterations and increments as typically found in Agile development without any adaptation [Berardi et al. 2011].

COSMIC method must be used to estimate user stories with no difficulties. If there is any difficulty, then this is almost certainly due to weaknesses (ambiguities or omissions) in the User Stories [Berardi et al. 2011].

Traditional User Story template, even expressing basic information that is enough for COSMIC sizing, it does not support some key points for a more precise estimation. For example, connections between data groups in a functional process and feedback to functional user clearly.

Considering that beyond traditional template, there are several other ones for US writing, we conducted a Multivocal Literature Review (MLR) to find the biggest number of proposed templates then we analyzed all of them in terms of cosmic sizing.

Multivocal Literature Review is a form of a Systematic Literature Review (SLR) which includes not just formal literature (e.g., journal and conference papers), but also grey literature, such as, blog posts and white papers[Garousi et al. 2017]. MLRs are useful for researchers and practitioners since they provide summaries the state-of-the art and practice in a given area [Garousi et al. 2017].

The rest of the paper is organized as follows. The section 2 present a background of User Story and COSMIC Method. Section 3 present the MLR protocol. The section 4 shows the analysis of MLR result in terms of COSMIC sizing. The section 5 is threats do validity and finally, section 6 present conclusion and future work.

2. Background

User Story technique is widely used in Agile development. They are characterized as short and high level description of required functionality written in customer language. The traditional US Template is:

"As a <role>, I want to <goal/desire>, so that <benefit>" [Abran et al. 2004].

US is used in very early of requirement gathering. It intend to contain just enough information in order to be able to produce estimate effort for implementation. A procedure it indicated to be written by the customer before the implementation for appropriated acceptance.

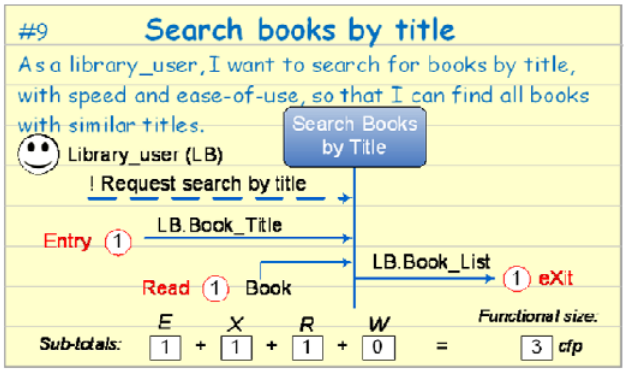
The agile guideline has the purpose of providing additional advice beyond the COSMIC Measurement Manual on Agile projects [Berardi et al. 2011]. The COSMIC method is perfectly suited for measuring the typically requirement sources found in Agile development without any adaptation [Berardi et al. 2011].

Sizing software in Agile development requires exactly the same knowledge, principles and rules of COSMIC Method when used in any other project management method [Berardi et al. 2011].

Message Sequence Diagram may be used to estimate user stories as shown in Figure 1. The vertical line represents a functional process and horizontal arrows represent data movements. Entries and Reads are shown as arrows incoming to functional process and Exit and Writes as outgoing arrows, appearing in the required sequence as top-down order [Berardi et al. 2011].

Besides the traditional template several number of templates have been appeared

Figure 1. User Story and Message Sequence Diagram. Source: [Berardi et al. 2011]



along the years intending to increase the expressiveness of the user story. A MLR was chosen as literature technique, based on the necessity of finding US template proposal which are not in traditional literature.

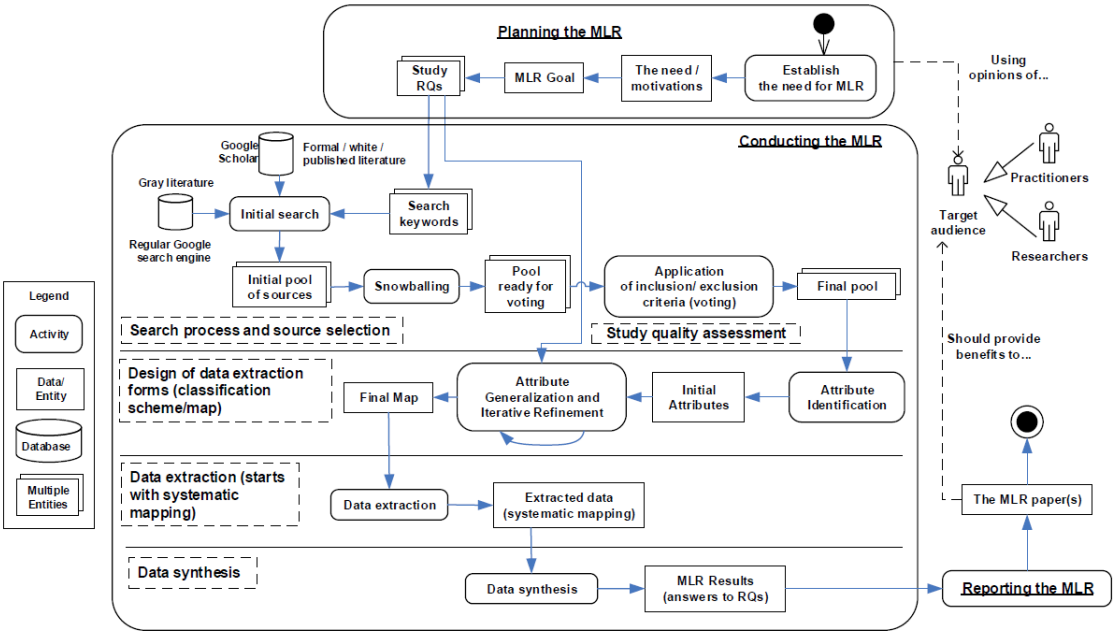
3. Multivocal Literature Review

The MLR is presented bellow, the section 3.1 presents the detailed study planning. Section 3.2 details de study conduction and finally, section 3.3 presents the result of the study.

3.1. MLR Planning

The general MLR process is presented in Figure. 2

Figure 2. Multivocal Literature Review Process. Source: [Garousi et al. 2017]



In Figure 2 it may be observed that MLR process is basically adding gray literature in the search step jointly to traditional literature. Based on this, a common protocol was created and it is presented in Table 1.

Table 1. MLR Protocol.

Research Objective
- Find the current user story models and/or templates in traditional and gray literature.

Research Questions
- Does the paper propose a template or model for user story writing?

Research Engines
- Scopus
- IEEE Xplore
- SpringerLink
- ACM DL
- ScienceDirect
- Compendex

Research Query
- TITLE((user story) OR (user stories)) AND KEY((user story) OR (user stories))

Including Criteria
- Paper must show an user story template/model.
- Paper must propose an user story template/model.

Excluding Criteria
- Paper not written in English.
- Paper does not show an user story template/model.

Quality Assessment
- Does the paper propose or cite an user story template?

Answers and Weight
- Paper which propose an US template -> 1.0
- Paper which cite an US template -> .5
- Paper does not propose or cite an US template -> -.5
- Full paper is Inaccessible -> -1.0

Data Extraction
- Proposed user story template/model.
- User story template/model reference.

3.2. MLR Conduction

The first step is the search in each search engine, second step is detection and elimination of duplicated papers, the third step is application of including and excluding criteria. The fourth step is the quality assessment and the last step is consideration of papers with weight higher than .5, in other words, papers which propose an US template. Table 2 presents the number of result for each search base in the table on the left and detailed number of paper after each step is in in the Table on the right. The final list of paper is presented in Table 3. The Table 4 shows with ID "FL" templates found.

Table 2. Papers returned for each search base and in each step.

Search Base	Papers
ACM Digital Library	14
El Compendex	89
IEEE Digital Library	15
Science@Direct	4
Scopus	71
Springer Link	0
Total	**193**

All results - 193
After duplicated detection - 100
Accepted - 74
Higher than 0 - 20
Higher than .5 - 3

Table 3. Papers from First result.

ID	Paper Name
FL01	Agile user stories enriched with usability [Moreno and Yagüe 2012]
FL02	User stories template for object-oriented applications [Zeaaraoui et al. 2013]
FL03	UserX story: Incorporating UX aspects into user stories elaboration [Choma et al. 2016]

After finding these result, we conducted the second search which is considering the gray literature. The protocol is basic the same, but with changing in search engine, that was replace by google. The including and excluding criteria, quality assessment and data extraction were kept the same.

Unlike common Systematic Literature Review, different stopping criteria for gray literature searches are needed in MLR [Abran et al. 2004]. We defined stopping criteria the first ten pages in google search, where each page shows 10 results, since there were evidence of exhaustion or saturation.

The search string used in google was *"\"user story\" template model standard example"*. This string returned 99.600 results. The resulting pages which a different US template was found were investigated until the root reference of this template (Snowballing), as happened in Wikipedia. The result of this search is presented in Table 4 with ID "GL".

3.3. MLR result

The final results are presented in Table 4. User story templates found in SLR are with ID FL, templates found in google search are with ID GL and previous known template is with ID KL.

The template FL01 [Moreno and Yagüe 2012] does not present so much contribution, just introduce and specific use of the user stories to express usability requirements, so in comparison with traditional template there is no relevant difference.

As mentioned before, the template FL02 [Zeaaraoui et al. 2013] does not also present significant difference, just explicit that the object must be present in the user story, which is a natural practice if you are using the traditional template idea.

Template FL03 [Choma et al. 2016] present contribution in terms of usability engineering. The last part which is responsible for express the feedback or expectation from the user point of view in traditional template, in FL03 this part is specific for Nielsen's heuristic values, which should be met in the user story.

Template GL01 [Cohn 2008], found in gray literature, was introduced setting the last part of traditional template as optional. Considering this, the user story is shorter and does not provide information about feedback or user expectation.

Template GL02 [Matts 2011] takes the received benefit, which is present as last part in traditional template, and put it as first part, before the user, then it follows the traditional approach.

The template GL03 [Pupek 2008] is also known as "Five Ws", it is added with more relevant information from from the user perspective. The added information are exactly, "when" and "where", the other "Ws" have already been present in traditional template.

Template KL01 was found in COSMIC Agile Guide [Berardi et al. 2011], the added information is precisely about non-functional requirement found in user story functional requirement. The guide goal was not propose this template but, the root reference for that was not found. Considering the counting example presented in the guide, this added information is not relevant for sizing estimation.

Table 4. List of Templates.

ID	User Story Template
FL01	"As a <role>, I want <usability requirement>" [Moreno and Yagüe 2012]
FL02	"As a <role>, I want to <action> <object>, so that <business value>" [Zeaaraoui et al. 2013]
FL03	"As a <persona>, I want/need <goal> so that <Nielsen's heuristic> will be met" [Choma et al. 2016]
GL01	"As a <role>, I want <goal/desire>" [Cohn 2008]
GL02	"In order to <receive benefit> as a <role>, I want <goal/desire>" [Matts 2011]
GL03	"As <who> <when> <where>, I <what> because <why>." [Pupek 2008]
KL01	"As a <role>, I want to <goal/desire>, <non-functional requirement>, so that <benefit>" [Berardi et al. 2011]

4. Result Analysis in terms of COSMIC

In terms of COSMIC sizing, the biggest part of found templates which add, in fact, modifications to traditional template, does not have significant contribution.

Templates which add usability information, provide valuable information for usability engineers, but it is more related to non-functional requirements, while COSMIC is a functional measurement method.

The "Five Ws" template, add few relevant information in terms of COSMIC sizing. Information of when the requirement should happen or be available and the information of where the user should run the requirement present in the user story, are in fact valuable for COSMIC estimators.

Despite this, There is still a gap of information for COSMIC sizing. The movements identification in a functional process are direct connected with the linked data groups or entities, present in the system.

5. Threats to Validity

Considering that this work presented in the paper is based on multivocal literature review, there are threats do validity mapped.

We performed a snowballing, in each gray literature that presented a new template and in the papers found in SLR, to find the root proposal of each US template.

Threats to validity the best result in gray literature. As mentioned in [Garousi et al. 2017], stop rules should be stated among the huge result number in a google search. We stated as considering the first 10 pages, with 10 links each page. In case of we perceive no saturation at this point, more pages would be considered, but in fact, results were found in only in the first page, so the following 9 pages did not presented any new result.

Another threat to validity, is that study was performed by only one researcher. In order to minimize this threat, we decided to perform a multivocal literature review instead of a systematic literature review. By this, any other result not considered in SLR could be found in gray literature step, moreover, a snowballing was performed in each resulting paper, book, or web page.

In order to avoid google shows result based on user experience, the search was performed in an incognito window of chrome browser, therefore, the search result tend to be replicable.

6. Conclusion and Future Work

We performed a multivocal literature review, in order to find proposed templates for user story writing. Furthermore, an specific analysis was made to find the relevance of these templates for COSMIC sizing purposes.

The result is that there are several approaches to extend the traditional template for US writing. Huge part of result found in this study are concerned with usability aspects in requirements. The information added or rewritten is concerned to usability information instead of other worries.

There was only one result that provide relevant information in terms of COSMIC sizing, The "Five Ws" template. Despite of this, the added information tend to solve two of the information gaps in traditional US template. The more important gap in traditional US template, that are entities linked to requirement expressed in the US are not solved.

As future work, we intend to study the viability to propose an extension to traditional user story template. This extension, will be performed to solve the gaps mentioned above and then, an empirical validation to this extension with the COSMIC community.

References

Abran, A., Baklizky, D., Desharnais, J.-M., Fagg, P., Gencel, C., Symons, C., R., J. K., Lesterhuis, A., Londeix, B., Nagano, S.-I., Santillo, L., Soubra, H., Trudel, S., Villavicencio, M., Vogelezang, F., and Woodward, C. (2015). *COSMIC Measurement Manual*. Common Software Measurement International Consortium.

Abran, A., Moore, J. W., Bourque, P., Dupuis, R., and Tripp, L. L. (2004). *Guide to the software engineering body of knowledge: 2004 version SWEBOK*. IEEE Computer Society.

Berardi, E., Buglione, L., Cuadrado-Collego, J., Desharnais, J.-M., Gencel, C., Lesterhuis, A., Santillo, L., Symons, C., and Trudel, S. (2011). *Guideline for the Use of COSMIC FSM to Manage Agile Projects*. Common Software Measurement International Consortium.

Choma, J., Zaina, L. A., and Beraldo, D. (2016). Userx story: incorporating ux aspects into user stories elaboration. In *International Conference on Human-Computer Interaction*, pages 131–140. Springer.

Cohn, M. (2008). Advantages of the "as a user, i want" user story template. `https://www.mountaingoatsoftware.com/blog/advantages-of-the-as-a-user-i-want-user-story-template`. Accessed: 2017-09-09.

Garousi, V., Felderer, M., and Mäntylä, M. V. (2017). Guidelines for including the grey literature and conducting multivocal literature reviews in software engineering. *arXiv preprint arXiv:1707.02553*.

Martin, J. (1991). *Rapid application development*. Macmillan Publishing Co., Inc.

Matts, C. (2011). Feature injection: three steps to success. `https://www.infoq.com/articles/feature-injection-success`. Accessed: 2017-09-09.

Maurer, F. and Martel, S. (2002). Extreme programming. rapid development for web-based applications. *IEEE Internet computing*, 6(1):86–90.

Moreno, A. M. and Yagüe, A. (2012). Agile user stories enriched with usability. In *International Conference on Agile Software Development*, pages 168–176. Springer.

Palmer, S. R. and Felsing, M. (2001). *A practical guide to feature-driven development*. Pearson Education.

Pupek, D. (2008). Writing user stories the 5 ws way. `http://blog.agilejedi.com/2008/03/writing-user-stories-5-ws-way-writing.html`. Accessed: 2017-09-09.

Schwaber, K. and Sutherland, J. (2011). The scrum guide. *Scrum Alliance*, 21.

Zeaaraoui, A., Bougroun, Z., Belkasmi, M. G., and Bouchentouf, T. (2013). User stories template for object-oriented applications. In *Innovative Computing Technology (INTECH), 2013 Third International Conference on*, pages 407–410. IEEE.

Proposta de Linguagem de Modelagem Gráfica com o Sirius: Uma Versão Open Source da DSL Canopus

João Carbonell, Bruno Medeiros, Pedro Silva,
Elder Rodrigues, Maicon Bernardino

[1]Universidade Federal do Pampa (UNIPAMPA)
Código Postal 97.546-550 – Alegrete – RS – Brasil

{joaocarbonellpc,brunobragamedeiros}@gmail.com, peu06@hotmail.com
elderrodrigues@unipampa.edu.br, bernardino@acm.org

Abstract. *Domain-Specific Languages (DSL) are programming languages developed to solve a specific problem of a domain, which can be represented graphically and/or textually. Thus, considering that many DSL development supporting tools are under commercial license, this study aims to propose the implementation of a new version of a DSL for modeling performance testing, called Canopus. For this, the study presents a proposal of a graphical DSL developed based on open source technologies, highlighting its requirements and design decisions, as well as a vision of the Eclipse Sirius modeling language development framework.*

Resumo. *Linguagem Específica de Domínio (Domain-Specific Language-DSL) é uma linguagem desenvolvida com o intuito de resolver um problema específico de um domínio, podendo ser representada de forma gráfica e/ou textual. Assim, considerando que muito do ferramental de apoio à criação de uma DSL é pago, este estudo tem como objetivo propor a implementação de uma nova versão da DSL Canopus para modelagem de testes de desempenho. Para isso, o estudo apresenta uma proposta de DSL gráfica desenvolvida com base em tecnologias open source, discutindo seus requisitos e decisões de projeto, bem como apresentando uma visão do framework de desenvolvimento de linguagem de modelagem Eclipse Sirius.*

1. Introdução

Linguagem específica de domínio (*Domain-Specific Language* - DSL) [Fowler 2010] é uma técnica para o desenvolvimento de sistemas computacionais com características particulares de um determinado domínio. Sua utilização é ampla, estando no desenvolvimento de aplicações relacionadas a campo geológico [De Sousa and Da Silva 2016] até a uma linguagem desenvolvida como auxílio a comunicação com banco de dados, *e.g.* SQL. O desenvolvimento de uma DSL é uma tarefa que envolve conhecimento teórico e prático, sendo necessário o entendimento do funcionamento de linguagens de programação de propósito geral para criar uma linguagem específica de domínio, como conhecimentos de gramática, sintaxe e semântica. Além disso, para o desenvolvimento de uma DSL devem ser utilizadas ferramentas que auxiliem no processo de criação da linguagem, fornecendo as funções para que seja possível a utilização da DSL ao seu usuário final. Estas ferramentas são conhecidas como *Language Workbench* (LW). Entretanto, muitas destas

LW's estão disponíveis comercialmente, gerando linguagens restritas e financeiramente custosas.

Um exemplo de DSL desenvolvida usando um LW comercial é a DSL Canopus [Bernardino et al. 2016], a qual foi criada com a MetaEdit+ da MetaCase[1]. A Canopus tem como objetivo a modelagem de testes de desempenho e geração automática de cenários e *scripts* de testes. Ela foi desenvolvida junto à indústria, por meio de uma parceria entre os pesquisadores e uma multinacional de TI. Apesar de considerarem a experiência com a MetaEdit+ satisfatória, tanto do ponto de vista dos pesquisadores ao criarem a linguagem, quanto do ponto de vista da empresa, o alto custo das licenças foi preponderante para a não adoção da linguagem.

Este estudo propõe uma nova versão *open source* da Canopus, com o propósito de disponibilizá-la em repositórios abertos a fim de que outros grupos de pesquisa e/ou comunidades de desenvolvedores possam contribuir, bem como incentivar que empresas a adotem na modelagem dos testes de desempenho.

Neste estudo, serão discutidos os requisitos e decisões de projetos necessários para a criação de desta nova versão *open source* da DSL gráfica, assim como uma visão geral dos conceitos e funcionamentos do *framework* para a modelagem Sirius, projeto do Eclipse Foundation [Eclipse 2017]. Este estudo foi organizado da seguinte forma: Seção 2 discute a fundamentação relacionada a DSL, *Language Wokbench* (LW) e trabalhos relacionados, bem como apresenta a Canopus. A Seção 3 discute a proposta da versão *open source* da Canopus, destacando seus requisitos e decisões de projeto para sua implementação. Por fim, a Seção 4 apresenta as conclusões relacionadas ao estudo e futuros passos em direção a nova versão gráfica da DSL.

2. Fundamentação Teórica

Linguagem Específica de Domínio (*Domain-Specific Language*-DSL) é uma linguagem de programação de computador, a qual têm foco no domínio de aplicação [Fowler 2010]. Focada em um domínio particular. Uma DSL pode ser dividida em duas categorias: **DSL Interna**: Linguagem que, geralmente, utiliza a estrutura da linguagem de propósito geral hospedeira, auxiliando em uma tarefa específica. Um exemplo: muitos dos mecanismos usados pelo Rails vêm do Ruby [Fowler 2010]. **DSL Externa**: Linguagem separada da linguagem host de prooósito geral(*General Purpose Language* - GPL), que realiza tarefas de apoio, mas de forma independente. Um exemplo: SQL (*Structured Query Language*) utilizada por outras linguagens, como o JAVA, para consulta e comunicação com bancos de dados. **Language Workbench (LW)**: São ferramentas que proporcionam um ambiente para a criação e suporte de DSL, por meio do desenvolvimento de metamodelos [Fowler 2010]. Suportando a criação, edição e manutenção de DSL, bem como uma IDE, que oferece um ambiente propicio à manipulação de GPL.

Um exemplo de LW que auxilia a notação gráfica de uma DSL é a MetaEdit+, que tem como objetivo tornar mais fácil a modelagem de domínios, possibilitando que os usuários se concentrem na criação de linguagens e modelos mais expressivos. Outra LW é a Whole Plataform, que é uma solução *open source* utilizada no auxílio de programação orientada a linguagem, mas que atualmente encontra-se sem suporte. Existe, também, o

[1]MetaEdit+ http://www.metacase.com/mwb/

framework Eclipse Sirius, cujas informações encontram-se mais detalhadas na Seção 3.3. Além de ferramentas que auxiliam no desenvolvimento da notação gráfica, existem ferramentas que contribuem no desenvolvimento de DSL textuais. Um exemplo é o Xtext, um *framework* da Eclipse Foundation para criação de novas linguagens usando representação textual.

2.1. Trabalhos Relacionados

Nos trabalhos relacionados, nota-se que existe uma carência de ferramentas gráficas *open source* para testes de desempenho. Existem alguns estudos propostos com o *framework* Sirius [Jäger et al. 2016] [Vujović et al. 2014], mas que não são para o domínio de teste de desempenho. O estudo de Jäger [Jäger et al. 2016] apresenta uma DSL gráfica, desenvolvida com a Sirius e o *Eclipse Modeling Framework*. A DSL foi proposta para um sistema de trem de brinquedos, visando ensinar a implementação de sistemas embarcados em tempo real, projetando um bloco central para controlar os trens e trocar de acordo com uma programação pré-definida. Foi relatado que, o conjunto de modelos específicos de domínio, editor gráfico, modelos de sistemas e simulações, levaram em torno de três dias para serem projetados e implementados com a abordagem proposta.

Já no estudo [Vujović et al. 2014], o objetivo é apresentar um editor gráfico de modelos específicos de domínio. Como exemplo de uso do editor gráfico, foi modelado um sensor RESTful para web. Cada elemento gráfico modelado representa um certo elemento dos sensores RESTful da web ou um serviço (elementos como números de nós sensores, métodos, abordagens) e é descrito pelas propriedades que descrevem esta condição. O estudo conclui que, desenvolvedores sem experiência com o domínio ou a solução específica, puderam facilmente definir tarefas e implementar serviços RESTFul.

2.2. Canopus

A Canopus [Bernardino et al. 2016] consiste em uma DSL gráfica e textual para modelagem de testes de desempenho, criada com a LW MetaEdit+. A Canopus faz usos de processos relacionados a abordagem de testes baseados em modelos (*Model-Based Testing* - MBT), a fim de melhorar e aperfeiçoar a modelagem e criação dos testes de desempenho. Assim, sendo possível gerar *scripts* por meio de modelos de desempenho que imitam a interação do usuário virtual (*Virtual User* - VU) com o sistema sob teste (*System Under Test* - SUT), então permitindo gerar representações textuais destes modelos ou *scripts*, os quais podem ser executados por ferramentas de desempenho, tais como o HP LoadRunner. Além disso, é possível gerar arquivos XML, para uso em outras ferramentas de teste de desempenho, como a PLeTsPerf [Rodrigues et al. 2015], capaz de interpretar o XML para gerar automaticamente cenários e *scripts* de testes de desempenho.

Para definir a Canopus, foi criado um metamodelo para a criação de modelos de testes de desempenho. Este metamodelo é composto por metatipos, os quais são usados para implementar uma DSL específica. Os metamodelos da Canopus, representados por 7 pacotes. São eles: *Modelo de Desempenho Canopus*, *Monitoramento de Desempenho Canopus*, *Cenário de Desempenho Canopus*, *Scripting de Desempenho Canopus*, *Métricas de Desempenho Canopus*, *Carga de Trabalho Canopus*, *Arquivos Externos de Desempenho Canopus*. Sendo que o Monitoramento de Desempenho Canopus, Cenário de Desempenho Canopus e *Scripting* de Desempenho Canopus, são os metamodelos principais, que juntos compõe o Modelo de Desempenho Canopus.

3. Proposta de DSL

Como apresentado anteriormente, a Canopus foi desenvolvida dentro da LW MetaEdit+, uma ferramenta de licença comercial. Este fator torna custoso financeiramente a aquisição e utilização da Canopus. Por isto, foi tomada a decisão de desenvolver uma nova versão da Canopus, a qual possuísse as mesmas características da sua versão antecessora, embora novas características possam ser adicionadas, mas que fundamentalmente estivesse disponível sob uma licença *open source*. Este trabalho apresenta uma proposta para a criação da representação gráfica da Canopus. Assim, primeiramente, será abordado o levantamento dos requisitos para a linguagem. Depois, as decisões de projeto da DSL serão discutidas, bem como as tecnologias disponíveis e desafios encontrados para a implementação da nova versão *open source* da Canopus.

3.1. Requisitos da Linguagem

Esta seção enumera os requisitos aplicados a Canopus, além de novos requisitos contemplados para a proposta da nova versão *open source* da DSL. A maioria dos requisitos foram previamente estabelecidos no estudo de Análise de Requisitos e Decisões de Projeto para a Canopus [Bernardino et al. 2014]. Nesta seção consta a releitura destes requisitos com algumas melhorias e apresenta alguns novos requisitos da linguagem.

RQ1) *A DSL deve permitir representar as características do teste de desempenho.* Uma das principais funções de um teste de desempenho é a identificação de limitações do sistema. Com isso, é necessário que as aplicações sejam medidas em pequenas partes chamadas transações. Dessa forma, cada atividade do sistema pode ser medida.

RQ2) *A linguagem deve ser implementada e disponibilizada sob uma licença open source.* Como principal mudança em relação a primeira versão da Canopus, qualquer ferramenta utilizada na implementação da linguagem deve ser *open source*, permitindo que ela seja disponibilizada em repositórios abertos a comunidade, academia e indústria.

RQ3) *A DSL deve prover uma representação gráfica das características do teste de desempenho.* Esse requisito não diz respeito a linguagem, mas sim a ferramenta utilizada para sua criação. Dessa forma, a ferramenta deve ser capaz de fornecer mecanismos de representar graficamente todas as características do domínio.

RQ4) *A DSL deve prover uma representação textual.* Dessa forma, engenheiros de testes, que estão acostumados com representação textual, irão ter mais facilidade em adotar a DSL, bem como documentar suas aplicações. A linguagem deve ter características e palavras-chaves que lembram o domínio de teste de desempenho.

RQ5) *A DSL deve incluir recursos que ilustrem os contadores de desempenho.* Dessa forma, é possível analisar o nível de qualidade da aplicação e a infraestrutura do servidor que a hospeda por meio de indicadores e métricas de desempenho.

RQ6) *A DSL deve permitir modelar comportamentos de diferentes perfis de usuários.* Esse requisito é específico do domínio de teste de desempenho. Esses comportamentos são modelados de acordo com os sistemas que estão em testes. Vale ressaltar que o foco da Canopus é para o domínio de aplicações Web.

RQ7) *A DSL deve realizar a rastreabilidade entre os elementos das notações gráficos e textuais.* A DSL proposta deve ser capaz de automatizar a conversão/tradução dos elementos gráficos para as suas contrapartes textuais e vice-versa.

RQ8) *A DSL deve permitir a exportação de modelos para formatos de tecnologias específicas.* Com esse requisito é possível garantir que a DSL proposta exporte seus modelos para ferramentas de testes de desempenho, *e.g.* HP LoadRunner, Apache JMeter.

RQ9) *A DSL deve exportar os modelos para o formato de arquivo XML.* Com isso, é possível que qualquer pessoa que queira usar as informações dos modelos Canopus, tenha apenas que interpretar o XML gerado para a sua tecnologia.

RQ10) *A DSL deve representar diferentes elementos do teste de desempenho em scripts de testes.* O diagrama modelado usando a DSL proposta deve representar múltiplos elementos de *scripts* de teste, tais como fluxos condicionais ou de repetição, entre outros.

RQ11) *A DSL deve permitir a modelagem de múltiplos cenários de testes.* Dessa forma, a DSL será capaz de modelar cenários de testes que representem cargas de trabalho normais, bem como cargas de estresse. Além disso, deve permitir a possibilidade de gerenciar a hierarquia de cenários, que é a decomposição de um cenário de teste em outros múltiplos cenários, a fim de permitir maior reuso entre eles.

3.2. Decisões de Projeto

Esta seção descreve as decisões de projeto para a criação da nova versão da Canopus, relacionadas aos requisitos mencionados na Seção 3.1. Assim, para cada decisão será apresentado seu(s) respectivo(s) requisito(s).

DD1) *O uso de soluções open source no auxílio da implementação de DSL gráficas* (RQ2, RQ3). Estes requisitos foram atendidos por meio de um mapeamento da literatura realizado, em que foram encontradas algumas ferramentas que atendiam a este requisito. Sendo assim, foi decidido pelo uso do *framework* Eclipse Sirius, de código aberto e que apoia a criação de DSL com notação gráfica.

DD2) *As características do domínio de teste de desempenho devem ser aplicadas de forma incremental* (RQ1, RQ5). Para estes requisitos foi adotado a estratégia de usar uma ontologia [Freitas and Vieira 2014], a fim de determinar e identificar as características que representam o domínio do teste de desempenho. Esta ontologia determina conceitos básicos, relações e restrições do domínio do problema.

DD3) *Fornecer uma linguagem gráfica que represente o comportamento de perfis de usuários para diferentes cenários de testes* (RQ6, RQ11). Foram analisados diferentes modelos e representações gráficas que auxiliam testes de desempenho para atender esses requisitos. A linguagem deve ter elementos capaz de representar o comportamento de diferentes perfis de usuários e que também representem as configurações dos cenários de teste, como as informações sobre o domínio (duração dos testes, tempo de processamento). Também a possibilidade de incluir a escolha aleatória e execução de probabilidades para as interações dos usuários virtuais. Além disso, deve ter a funcionalidade de representar de forma abstrata os dados que devem ser instanciados nas atividades do processo de teste de desempenho, como a geração de *scripts* de teste.

DD4) *Criar uma representação textual em linguagem semi-natural*. Para facilitar o entendimento dos testes por parte da equipe e dos intervenientes, deve-se permitir a representação textual em linguagem semi-natural (RQ4). Linguagens que utilizam linguagens naturais, como a Gherkin [Wynne and Hellesøy 2012], as quais permitem descrever testes funcionais, não costumam oferecer mecanismos para avaliar o teste de de-

sempenho. Por isto, pretende-se incluir as características do teste de desempenho visando estender esta linguagem. A Gherkin é comumente usada para ser interpretada por uma ferramenta de linha de comando chamada Cucumber, a qual automatiza a execução do teste de aceitação.

DD5) *Permitir a rastreabilidade entre as notações gráficas e textuais* (RQ3, RQ4, RQ10). A LW deve permitir a conversão/tradução entre as regras dos modelos. Ou seja, a modelagem bidirecional entre os elementos gráficos e seus respectivos ativos na notação textual devem ser mapeadas. Não sendo este mapeamento apenas de um para um, mas que permita que uma notação gráfica seja mapeada para várias instâncias textuais. Como não existe uma única ferramenta *open source* que atenda a isso, será realizada uma integração entre os *frameworks* Sirius [Eclipse 2017] e Xtext [Efftinge and Völter 2006].

DD6) *Integração entre a DSL e outras tecnologias (RQ8, RQ9).* Deve permitir a exportação dos modelos, cenários e *scripts* modelados para outros formatos, tanto formatos genéricos, *e.g.* XML, quanto formatos específicos, tal como HP LoadRunner.

3.3. Framework Eclipse Sirius

O desenvolvimento de uma DSL gráfica requer ferramentas que auxiliem o processo de criação de seus elementos. Após a pesquisa em estudos da área de desenvolvimento de DSL, foi encontrado um *framework* para auxiliar no desenvolvimento da versão *open source* da Canopus, o *framework* Sirius [Eclipse 2017]. O *framework* Sirius é um projeto da *Eclipse Foundation*, cujo objetivo é permitir a criação de ferramentas de modelagem. De maneira geral, o Sirius é utilizado para elaborar e analisar um sistema a fim de enriquecer a comunicação com outros membros da equipe, parceiros ou clientes [Eclipse 2017]. No Sirius, os editores são definidos por um modelo que tem o objetivo de estabelecer o comportamento e todas as ferramentas de edição e navegação utilizadas para adaptar uma DSL.

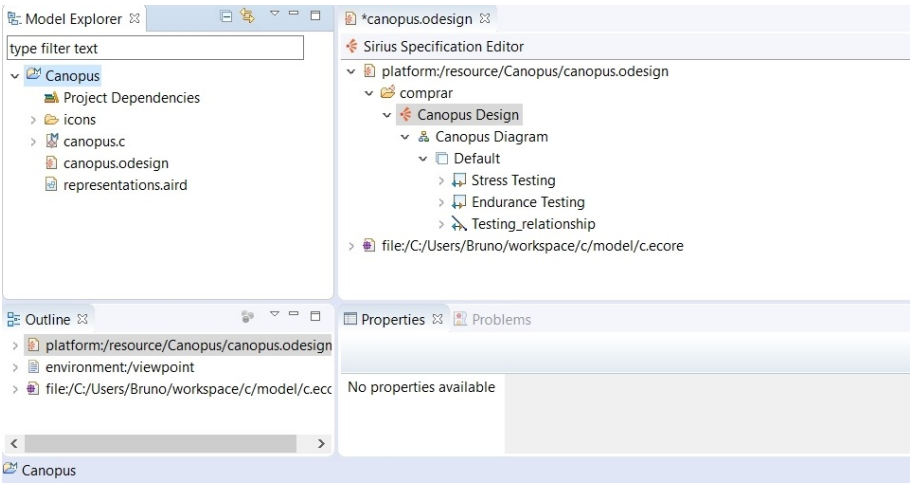


Figura 1. Interface da Ferramenta Sirius

Pode ser observado na Figura 1 o início da nova versão da Canopus. Ressalta-se que o estudo ainda está em andamento, porém é importante dissertar sobre o funcionamento da ferramenta e suas características em relação a DSL abordada no estudo. Ao lado esquerdo da Figura 1, na aba denominada `Model Explorer`, encontram-se os

arquivos originados na criação do projeto Canopus. A modelagem da DSL é realizada no arquivo de extensão `.odesign`. A tarefa de modelagem é feita na aba denominada `Sirius Specification Editor`, situada na parte direita da imagem. Todos os elementos gráficos da DSL, suas interações e representações são trabalhadas nessa aba. Como pode ser visto na Figura 1, a representação gráfica da Canopus possui três elementos gráficos. Dentro do pacote `canopus`, existe o `Canopus Diagram`, em que os elementos gráficos são inseridos. Esse diagrama possui três elementos: "Stress Testing" e "Endurance Testing"; além da conexão entre os dois, chamada de "Testing Relationship".

Os elementos gráficos contidos na modelagem de uma DSL dentro do *framework* Sirius têm suas características descritas na aba `Properties`, em que, para cada elemento, o desenvolvedor deve inserir desde características gráficas à lógicas de relacionamento entre elementos. Na aba denominada `Problems`, os problemas descobertos durante a validação da modelagem são expostos. O *framework* Sirius possui mecanismos para especificar a representação dos modelos em formas de diagramas, tabelas e árvores. Os diagramas são utilizados para modelagem gráfica de informações; as árvores, utilizadas em representações hierárquicas; e as tabelas têm o intuito de representar elementos posicionados em uma matriz, formada por linhas e colunas. Além disso, dentro de um elemento de modelagem podem ser criados "Regras de Validação", em que as informações de dados inseridos na modelagem dos elementos podem ser verificadas. Sendo assim, o *framework* Sirius apresenta uma série de características que possibilita o desenvolvimento da nova versão *open source* da Canopus.

4. Conclusão

Diante da dificuldade em encontrar ferramentas que auxiliem o processo de criação de DSL gráficas e que também seja de licença gratuita, o *framework* Sirius é uma solução que oferece a maioria das funcionalidades necessárias a implementação de linguagens gráficas, possuindo uma interface familiar a quem utiliza o ambiente Eclipse e abrindo a possibilidade de integração com outros *frameworks*. Nesse estudo foi apresentado uma proposta de implementação de uma versão *open source* da Canopus. Assim, foi apresentado uma visão geral sobre a Canopus e o *framework* de desenvolvimento Sirius. Além de serem levantados e discutidos os requisitos necessários para criação dessa versão da DSL, seus requisitos originais e os da nova versão foram relatados e debatidos, bem como foram discutidas as decisões de projetos necessárias para atender aos requisitos. Como trabalho futuro, será realizada a implementação da versão *open source* da Canopus. Outro ponto importante a ser destacado é, que além da implementação gráfica, a textual também será desenvolvida. Portanto, será necessário implementar a bidirecionalidade na conversão entre as notações, transformando a Canopus em uma DSL gráfica e textual, por intermédio da integração dos *frameworks* Sirius com o Xtext.

Referências

Bernardino, M., Zorzo, A., Rodrigues, E., Oliveira, F., and Saad, R. (2014). A domain-specific language for modeling performance testing: requirements analysis and design decisions. In *9th Int. Conf. on Software Engineering Advances*, pages 609–614.

Bernardino, M., Zorzo, A. F., and Rodrigues, E. M. (2016). Canopus: A domain-specific language for modeling performance testing. In *Int. Conf. on Software Testing, Verification and Validation*, pages 157–167, Chicago, USA.

De Sousa, L. M. and Da Silva, A. R. (2016). A domain specific language for spatial simulation scenarios. *GeoInformatica*, 20(1):117–149.

Eclipse (2017). Framework Eclipse Sirius. `https://eclipse.org/sirius/`.

Efftinge, S. and Völter, M. (2006). oAW xText: A framework for textual DSLs. In *Workshop on Modeling Symposium at Eclipse Summit*, volume 32, page 118.

Fowler, M. (2010). *Domain-specific languages*. Pearson Education.

Freitas, A. and Vieira, R. (2014). An ontology for guiding performance testing. In *IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies*, pages 400–407. IEEE Computer Society.

Jäger, S., Maschotta, R., Jungebloud, T., Wichmann, A., and Zimmermann, A. (2016). Creation of domain-specific languages for executable system models with the eclipse modeling project. In *Systems Conference (SysCon), 2016 Annual IEEE*, pages 1–8. IEEE.

Rodrigues, E., Bernardino, M., Costa, L., Zorzo, A., and Oliveira, F. (2015). PLeTsPerf-A Model-Based Performance Testing Tool. In *8th Int. Conf. on Software Testing, Verification and Validation*, pages 1–8.

Vujović, V., Maksimović, M., and Perisić, B. (2014). Sirius: A rapid development of dsm graphical editor. In *18th Int. Conf. on Intelligent Engineering Systems*, pages 233–238.

Wynne, M. and Hellesøy, A. (2012). *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. The Pragmatic Bookshelf.

Criando uma Linguagem Específica de Domínio usando o Xtext: Uma Opção *Open Source* para Implementar a Canopus

Aníbal I. Neto, Luiz P. Franz, Jonnathan R. Lopes,
Elder Rodrigues, Maicon Bernardino

[1]Universidade Federal do Pampa (UNIPAMPA)
Código Postal 97.546-550 – Alegrete – RS – Brasil

{netoiiung, luizpaulofranz, jonnathan.riquelmo}@gmail.com
elderrodrigues@unipampa.edu.br, bernardino@acm.org

Abstract. *Domain-Specific Language (DSL) is the name given to languages used in specific aspects of a system. Implementing a DSL is not a trivial task, since they are programming languages and have a syntax that is defined by a grammar. In this way, it is necessary to use tools that support the definition of concepts for the new language. Tools with this ability are popularly known as Language Workbenches. This article presents a guide to assist on the development of a DSL using the Xtext Eclipse framework and aims to present an alternative to the definition of the textual notation and development of the Canopus DSL.*

Resumo. *Linguagem Específica de Domínio (Domain-Specific Language- DSL) é o nome dado a linguagens utilizadas em aspectos específicos de um sistema. O desenvolvimento de uma DSL não é tarefa trivial, pois são linguagens de programação e possuem uma sintaxe que é definida por uma gramática. Desta forma, faz-se necessário a utilização de ferramentas que suportem a definição dos conceitos para a nova linguagem. Ferramentas com essa capacidade são popularmente conhecidas como Language Workbenches. Este artigo apresenta um guia para auxiliar o desenvolvimento de uma DSL utilizando o framework Eclipse Xtext e tem como objetivo apresentar uma alternativa para a definição da notação textual e desenvolvimento da DSL Canopus.*

1. Introdução

As Linguagens Específicas de Domínio (*Domain-Specific Languages*-DSL), são linguagens de programação que exercem um papel de auxílio fornecendo maior nível de abstração para problemas específicos. Essas linguagens visam modelar e descrever um domínio ou uma solução distinta e tem diversas aplicações. A adoção de uma DSL pode trazer vários ganhos, como geração automática de código e reuso [Fowler 2010].

Um grande tema da Engenharia de Software em que a utilização de uma DSL pode auxiliar é a área de testes de software. É de senso comum que um software, quando testado de forma adequada, atinge um bom nível de qualidade. Existem diversas técnicas e estratégias de teste de software, que podem variar com o nível da aplicação testada e com os objetivos do teste, *e.g.* teste de desempenho. Uma solução amplamente adotada diz respeito aos testes automatizados. Um conjunto de testes automatizados possibilita,

hipoteticamente, a oportunidade de um software ser testado com maior frequência e rapidez em diversos aspectos [Ammann and Offutt 2008]. Entretanto, pode se tornar difícil manter um conjunto de testes automatizados. À medida que um software muda, ou ainda mesmo durante o seu desenvolvimento, é necessário que os testes sejam adaptados.

Nesta perspectiva é possível afirmar que a qualidade dos testes pode ficar mais frágil quando estes estão no nível de interface gráfica do usuário. Isso ocorre pois geralmente esses testes são criados por meio de ferramentas que utilizam a técnica de Captura e Repetição (*Capture and Replay* - CR). Nesta técnica o engenheiro deve executar os passos dos testes manualmente no software uma vez usando alguma ferramenta que dê apoio à captura das interações e, em seguida, executar um modo de repetição para simular as ações previamente realizadas. Estas ferramentas geram *scripts*, porém estes testes produzidos são difíceis de serem modificados ou atualizados, tornando assim necessário, muitas vezes, a reexecução do processo da técnica CR.

Uma alternativa a técnica CR são os Testes Baseados em Modelos, do inglês *Model-Based Testing* (MBT). Algumas vantagens que o MBT pode ocasionar são a facilidade de compreensão, o maior nível de reuso e geração de artefatos de teste. É possível afirmar também que a preocupação com o projeto de modelos de teste, desde estágios iniciais de desenvolvimento, é maior utilizando essa abordagem [Kramer and Legeard 2016]. Na abordagem MBT, o engenheiro projeta os modelos de teste, aponta informações de desempenho e, logo após, usa uma ferramenta de apoio para gerar automaticamente um conjunto de *scripts* e cenários de teste. Existem algumas notações, linguagens e modelos que podem ser aplicados para o teste de desempenho, *e.g.* *Unified Modeling Language* (UML), *User Community Modeling Language* (UCML) e a *Customer Behavior Modeling Graph* (CBMG).

O objetivo do presente artigo é apresentar uma alternativa para a definição e desenvolvimento do metamodelo da Canopus [Bernardino et al. 2016b], que é uma DSL que especifica e automatiza o processo de teste de desempenho para aplicações Web. No que tange o uso de uma DSL para auxiliar o processo de teste de desempenho de aplicações Web, várias soluções com esse propósito já foram propostas na literatura. Especialmente no contexto da avaliação de desempenho de aplicações na nuvem pode-se citar o caso dos ambientes Expertus [Jayasinghe et al. 2012], Cloudbench [Silva et al. 2013], Cloud Crawler [Cunha et al. 2013], e Cloud Work Bench [Scheuner et al. 2014]. Além disso, a automação das atividades de teste de desempenho é uma das práticas DevOps[1] que têm sido cada vez mais utilizadas pela indústria, a qual pode ser realizada com o apoio de ferramentas abertas, independentes de qualquer tecnologia proprietária, como Gatling [GatlingCorp 2017] e Locust [Heyman et al. 2017].

A Canopus foi proposta em 2016, utilizando o MetaEdit+ Workbench [MetaCase 2017] como ferramenta de metamodelagem. A ferramenta usada provê suporte à definição de DSL, sendo importante salientar que é um instrumento proprietário. A alternativa apresentada nesta proposta é o *framework* Xtext, uma opção *open source* que possibilita a definição de DSL de forma textual e oferece outras funções, entre elas a geração de código-fonte.

[1]DevOps: termo provêm da composição de "desenvolvimento" e "operações". É um processo de desenvolvimento e entrega de software que enfatiza comunicação e colaboração entre profissionais de gerenciamento de produtos, desenvolvimento de software e operações.

Este artigo está organizado da seguinte forma. A Seção 2 apresenta uma contextualização sobre Linguagem Específica de Domínio, bem como ferramentas que fornecem suporte a criação de DSL. Prosseguindo, na Seção 3 é demonstrado uma ideia geral do processo de implementação de uma DSL no Xtext. A Seção 4 apresenta os trabalhos relacionados. Por fim, a Seção 5 descreve a conclusão e trabalhos futuros.

2. Linguagem Específica de Domínio

Uma linguagem de programação possui uma sintaxe, que é definida por sua gramática. E gramática segundo [Russell and Norvig 2004] é composta por definições matemáticas rígidas, que definem a ordem dos lexemas da linguagem, ou seja, a ordem esperada das "palavras", enquanto que a semântica de uma linguagem é o significado de uma expressão válida nessa linguagem [Fowler 2010]. As linguagens de programação são classificadas em dois grupos: Linguagens de Propósito Geral (*General Purpose Languages* - GPL), que são linguagens de programação de uso geral, tais como: JAVA, C, C# e Python, e; Linguagens Específicas de Domínio (*Domain-Specific Languages* - DSL) [Fowler 2010] são linguagens de programação voltadas para um aspecto específico de um sistema, em que não podem ser utilizadas para o desenvolvimento de sistemas inteiros, pois o que define uma DSL é a expressividade limitada e o foco no domínio. As DSL são amplamente utilizadas no desenvolvimento de sistemas, são exemplos de DSL: SQL, HTML.

Existem duas classificações de DSL [Fowler 2010]: (i) DSL Interna: uma forma específica de escrever uma linguagem de propósito geral (*General Purpose Language* - GPL). É uma linguagem que utiliza a estrutura de uma GPL como base, também conhecida como Linguagem Host; (ii) DSL Externa: uma linguagem separada de uma GPL, possui sintaxe e semântica própria e precisa de um compilador/interpretador próprio para ser compilada/interpretada [Fowler 2010]. Uma DSL fornece um meio para comunicar e expressar mais claramente a definição de uma parte/todo do sistema, e tornar mais fácil a realização de determinadas tarefas, tais como: ler um trecho de código, encontrar erros e, modificar o sistema.

2.1. Ferramentas de Suporte a Criação de DSL

Tanto na indústria como na academia, o surgimento de ferramentas de suporte para DSL cresce consideravelmente. Ferramentas que dão suporte a uma ou várias etapas do processo de desenvolvimento de DSL são classificadas como frameworks, IDEs e as chamadas *Language Workbenchs* (LW), que segundo Fowler [Fowler 2010], são IDEs especializadas em desenvolvimento de DSL. Bons exemplos desse tipo de ferramentas são: **MPS**: LW desenvolvida pela JetBrains que suporta a criação de DSL textual [JetBrains 2017]; **MetaEdit+**: LW com suporte para notação gráfica, desenvolvida pela MetaCase [MetaCase 2017]. **EMFText**: Plugin do Eclipse que dá suporte a definição de sintaxe de linguagens [Wende et al. 2017]; **Whole**: LW baseada no Eclipse que suporta a criação de DSL textuais e gráficas [Solmi 2017]; **Xtext**: *framework* Eclipse que apoia a criação de linguagens textuais [Eysholdt and Behrens 2010], o qual é o foco deste estudo.

2.2. Xtext

É um framework voltado para o desenvolvimento de linguagens de propósito geral e DSLs [Eysholdt and Behrens 2010]. Esse framework permite um rápido desenvolvimento de ferramentas que dão suporte a criação de novas linguagens, como suporte a

implementação da gramática, gerador de compiladores e analisadores léxicos e sintáticos. O Xtext faz parte do grupo de ferramentas voltadas para a criação de linguagens do Eclipse (*Eclipse DSL Tools*), possui licença *open source* e cobre grande parte dos requisitos comuns no desenvolvimento de linguagens textuais.

3. Desenvolvendo um Exemplo de DSL com o Xtext

Ao criar um projeto de DSL no Xtext, algumas configurações são necessárias. Inicialmente, é definido um nome e qual a extensão dos arquivos que vai conter a DSL criada. Na sequência, um *wizard* oferece opções para a criação do produto (IDE) que dará suporte a nova DSL, o Xtext oferece a possibilidade de criação *plugins* para as IDE *Eclipse* e *IntelliJ*. Por padrão, no momento da criação, o Xtext gera três projetos com seus nomes baseados no atributo *project name*. O primeiro projeto, com o nome que foi determinado na configuração do *wizard*, é o principal. Nele é definida a gramática, os validadores e gerador automático de código. Todos os demais projetos são dependentes deste. Para DSL simples, esse é o único projeto requerido para edição de código [Bettini 2016].

O projeto que finaliza com ".ide" é um projeto que contém os recursos para a implementação da IDE para a nova DSL. Esse projeto é gerenciado pelo próprio Xtext. Não é recomendado nenhum tipo de alteração, pois esse projeto contém a implementação dos *parsers* gerados a partir da definição da gramática. O projeto com extensão ".ui" é o responsável pela integração da nova DSL com a IDE *Eclipse*. Esse é o projeto que se torna um produto Eclipse, portanto, informações ou funcionalidades adicionais devem ser implementadas neste projeto, dependendo dos objetivos. Porém nesse cenário, o mais recomendado é o desenvolvimento de um *plugin* externo a esses projetos, seguindo o princípio de separação de responsabilidades. Nesta última abordagem, é possível importar as novas funcionalidades nos outros *plugins* ou produtos Eclipse.

Quando a opção *Testing Support* é marcada no *wizard*, são gerados dois projetos extras para fins de testes. O projeto com extensão ".tests", é usado para testar a nova DSL, em que um trecho da nova DSL é a entrada, comparando com a saída gerada. Já o projeto com a extensão ".ui.tests", é usado para testar o editor de código, *i.e.* a parte visual do produto Eclipse. Apresentados os projetos iniciais gerados, o artefato que merece mais atenção é o arquivo com a extensão ".xtext", localizado no projeto principal. É o arquivo em que é especificado a gramática da DSL a ser desenvolvida com a *Grammar Language*.

A *Grammar Language* apresenta uma sintaxe baseada em gramáticas, portanto, segue o conceito de regras. Cada regra da gramática da nova DSL deve ter: (i) um nome ou identificador válido, (ii) seguido do caractere ":", e (iii), na sequência as definições dessa regra. Essas definições podem conter outras regras, palavras reservadas e operadores. Estas regras são separadas em **regras terminais** e **regras não-terminais**. Regras terminais definem o léxico da linguagem por meio de expressões regulares e contém as palavras (ou *tokens*). As regras não-terminais por usa vez podem ser composições de outras regras terminais e não-terminais, além de poder conter palavras reservadas. As regras não-terminais são as responsáveis por definir a ordem dos lexemas e, por fim, definir a sintaxe da linguagem. O Xtext oferece um conjunto pré-programado com regras terminais, como identificadores válidos para variáveis, definições de comentários e *strings*. Esse conjunto é chamado de *Terminals*, e todo projeto de linguagem do Xtext já vem com esse recurso importado. O Xtext permite também a importação de outras linguagens definidas

por terceiros. O recurso que permite a mescla de DSL é chamado de *mix-in* de linguagens. A Figura 1 apresenta mais detalhes de como é uma *Grammar Language*.

```
1  grammar org.xtext.example.DomainModel with
2  org.eclipse.xtext.common.Terminals
3  generate domainModel "http://www.xtext.org/example/DomainModel"
4
5  Domainmodel: (elements+=Type)*;
6
7  Type: DataType | Entity;
8
9  DataType: 'datatype' name=ID;
10
11 Entity:
12     'entity' name=ID ('extends' superType=[Entity])? '{'
13         (features+=Feature)*
14     '}';
15
16 Feature: (many?='many')? name=ID ':' type=[Type];
```

Figura 1. *Grammar Language*

```
1  datatype String
2  entity Blog {
3      title: String
4      many posts: Post }
5
6  entity HasAuthor { author: String}
7
8  entity Post extends HasAuthor {
9      title: String
10     content: String }
```

Figura 2. Linguagem com Xtext

Nas duas primeiras linhas da Figura 1 ocorre o *mix-in* de linguagens, importando as regras terminais padrão. Em seguida a palavra reservada *generate* fornece instruções ao núcleo EMF, além de definir o nome da DSL proposta [Bettini 2016]. A primeira regra propriamente da DSL costuma ser bem genérica, é a raiz da gramática, ela define por onde o *parser* vai iniciar e é conhecida como regra inicial (linha 5 na Figura 1), [Bettini 2016]. A regra inicial neste exemplo é a regra *DomainModel*. Ela tem uma propriedade ou atributo (ou ainda variável) chamada *elements*. Esta propriedade contém uma coleção de ocorrências da regra *Type*. Os caracteres "(...)*" indicam que *Type* pode ocorrer zero ou muitas vezes. Essa coleção será armazenada na propriedade *elements*. Quando trata-se de coleções, o operador de atribuição "+=" é necessário para indicar que *elements* é uma coleção. Caso seja utilizado o operador de atribuição "=" apenas a última ocorrência de *Type* seria armazenada.

Na linha 7 da Figura 1 a regra *Type* é definida. Ela apenas informa que um *Type* pode ser um *DataType* **ou** *Entity*. A definição da regra *DataType* é simples, pois apenas apresenta a palavra reservada "datatype" (essa é a forma como palavras reservadas são definidas com *grammar-language*) e uma propriedade *name* que deve receber um identificador válido, definido na regra padrão ID, oferecida pelo Xtext.

A primeira orientação que a regra *Entity* define na linha onze é sua palavra reservada "entity". Após, a regra espera um identificador para a entidade em questão, que será atribuída à propriedade *name*. Uma entidade pode ou não estender outra entidade. A palavra reservada "extends" está contida em um parênteses, e esse parênteses é seguido do operador "?" que indica opcionalidade. É importante enfatizar o trecho "superType=[Entity]", onde os colchetes são usados para realizar a chamada **referência cruzada**. Este é um dos recursos mais poderosos do Xtext. Isso significa que a propriedade *superType* só receberá uma entidade já declarada. Isso se justifica no paradigma de orientação a objetos, em que é preciso declarar a classe mãe antes das classes filhas. O uso dos colchetes informa ao *parser* que nesse ponto é esperado o nome de uma *Entity* já declarada. Para a referência cruzada funcionar, a regra deve possuir a propriedade *name*.

A regra *Entity* ainda define que uma entidade pode conter uma lista de *Features* (linha trêze) e que essa lista pode ser vazia, por meio do operador "*" (zero ou muitos). Por fim, a regra *Feature*, linha dezesseis, inicia com a opcionalidade na palavra reservada "many". O operador de atribuição "?=", indica que essa é uma propriedade *booleana*,

portanto, quando houver a **palavra reservada** "many" no código de uma *Feature*, a **propriedade** *many* vai receber *true*, indicando que se trata de um *array*. Após isso, a regra espera um nome seguido da palavra reservada ":". Então, é indicado o tipo dessa *Feature*, utilizando novamente referência cruzada.

Essa DSL contém as seguintes palavras reservadas: "datatype", "entity", "extends", "{", "}", ":" e "many". Um trecho de código que respeita essa gramática é apresentado na Figura 2. Ela traz o *highlight* das palavras reservadas. O Xtext ainda gera uma série de artefatos, inclusive os *parsers* a partir da gramática descrita. Com a gramática definida e os artefatos criados é possível testar a DSL em um editor *Eclipse* por meio da opção "Eclipse Application".

As análises léxicas, sintáticas e ferramental de apoio da IDE são gerados automaticamente a partir da gramática. Com isso, obtêm-se um editor funcional para a DSL. Porém, é importante notar que a geração de códigos precisa ser implementada. Após a geração dos artefatos, um pacote "**generator**" é criado no projeto principal. Esse pacote já contém um esboço do gerador de código, e é nele onde são codificadas as regras para a geração de código e persistência em arquivos, o que seria a "compilação" de uma DSL. Além do pacote *generator*, outro pacote importante criado é o "**validation**". A corretude de um código não pode ser determinado unicamente pelas análises léxicas e sintáticas, sendo necessário muitas vezes uma etapa extra de **análise semântica**. No Xtext as análises semânticas são implementadas na forma de *validators* [Bettini 2016].

O Xtext ainda cria dois projetos separados para os testes. Um é responsável pelo *parser* e *generator* enquanto o outro testa a interface gráfica. Os testes para os *parsers* são relativamente simples. No exemplo gerado, basicamente, deve ser informado uma entrada (código da nova DSL) e uma saída. Esta pode ser um objeto do EMF ou uma *string* contendo os códigos gerados esperados. Por fim, ainda é possível empacotar o produto como um *plugin* Eclipse, ou Eclipse RCP, criando um produto Eclipse personalizado contendo apenas a nova DSL criada [Bettini 2016].

4. Trabalhos Relacionados

[Arkin and Tekinerdogan 2014] utilizam o Xtext para implementar duas DSL para aplicações paralelas em plataformas de computação paralela. Ambas as DSL criadas são externas, a primeira foi criada para descrever a configuração física e a segunda para descrever os componentes e a construção deles.

[Demirkol et al. 2013] apresentam uma DSL chamada *Semantic Web Enabled Agent Language* (SEA-L). Essa DSL foi criada utilizando o Xtext junto a alguns outros *plugins* de suporte a notação gráfica, tendo em vista o fato do Xtext dar suporte apenas a notação textual, também pertencentes ao grupo Eclipse. A DSL foi criada para diminuir a complexidade do desenvolvimento de agentes de software para sistemas multiagentes. Assim, os autores realizaram um estudo de caso para avaliar na prática os ganhos obtidos com o uso da DSL no desenvolvimento de um sistema multiagente. Nesse caso, trataram de um sistema de trocas eletrônicas baseado em agentes.

[Nakamura et al. 2012] apresenta uma DSL externa para resolver um problema recorrente na área de MSR (*Mining Software Repositories*). A DSL criada foi chamada de *QORAL*, a qual tem por objetivo facilitar o aumento do desempenho de múltiplas

iterações. O autor utilizou questionários com pesquisadores da área, sem explicar a gramática da *QORAL* para avaliar a curva de aprendizado da linguagem e a capacidade de modificar o código de acordo com o entendimento. Os resultados indicaram que a *QORAL* facilitou o entendimento e a fácil modificação do código.

Por fim, [Marand et al. 2015] utilizaram o Xtext para implementar a "parte textual" da DSML4CP, uma DSL voltada para a programação concorrente. Essa DSL foi criada para ser utilizada em um nível mais alto de abstração que o código fonte. Desta forma, como a DSL proposta é, ao mesmo tempo, gráfica e textual, isso facilita a modelagem do domínio, reduzindo complexidades no desenvolvimento desse tipo de sistema.

5. Conclusão

Este artigo resume parte de um levantamento iniciado pelo *Laboratory of Empirical Studies in Software Engineering* (LESSE), um grupo de pesquisa da UNIPAMPA. O grupo busca, entre outros estudos, encontrar meios para desenvolver uma solução *open source* que auxilie no processo de criação de testes de desempenho baseado por modelos. A Canopus foi selecionada como solução padrão. A escolha tem como apoio estudos que apresentam, entre outros resultados, vantagens da DSL em relação à UML quando aplicada para modelagem de testes de desempenho para aplicações Web [Bernardino et al. 2016a]. É importante salientar que mediante investigação não foram encontradas na literatura propostas com as mesmas características. A condução do desenvolvimento da Canopus usada como exemplo de uso para apresentar na prática o Xtext, nos possibilita afirmar que ele é um *framework* que contempla/suporta os requisitos de criação da notação textual da DSL.

Em outra perspectiva da pesquisa do LESSE está em andamento o estudo sobre o *framework* Sirius. O Sirius é um projeto *open source* que fornece meios de criação de metamodelos gráficos. O Sirius, assim como o Xtext, tem compatibilidade com o *Eclipse Modeling Framework* (EMF), um conjunto de recursos do Eclipse para representar modelos e gerar código equivalente. O objetivo final da pesquisa é a integração de ambos os metamodelos da Canopus, convergindo em um *plugin* com os metamodelos textuais e gráficos no Eclipse. Desta forma, o produto final será uma solução *open source* para modelagem bidirecional das notações textual e gráfica para o teste de desempenho, com capacidade de geração de *scripts* e cenários de desempenho para outras ferramentas. Assim, espera-se que esta solução proposta permita viabilizar a adoção por parte da indústria, bem como prover o envolvimento de outros grupos de pesquisa e/ou comunidade para futuras contribuições e evolução do projeto.

Referências

Ammann, P. and Offutt, J. (2008). *Introduction to Software Testing*. Cambridge Press.

Arkin, E. and Tekinerdogan, B. (2014). Domain specific language for deployment of parallel applications on parallel computing platforms. Vienna, Austria.

Bernardino, M., Rodrigues, E. M., and Zorzo, A. F. (2016a). Performance Testing Modeling: An Empirical Evaluation of DSL and UML-based Approaches. In *31st Annual ACM Symposium on Applied Computing*, pages 1660–1665, New York, USA. ACM.

Bernardino, M., Zorzo, A. F., and Rodrigues, E. M. (2016b). Canopus: A domain-specific language for modeling performance testing. In *Int. Conf. on Software Testing, Verification and Validation*, pages 157–167.

Bettini, L. (2016). *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd.

Cunha, M., Mendonca, N., and Sampaio, A. (2013). A declarative environment for automatic performance evaluation in iaas clouds. In *6th Int. Conf. on Cloud Computing*, pages 285–292. IEEE.

Demirkol, S., Challenger, M., Getir, S., Kosar, T., Kardas, G., and Mernik, M. (2013). A DSL for the Development of Software Agents working within a Semantic Web Environment. *Computer Science and Information Systems*, 10(4):1525–1556.

Eysholdt, M. and Behrens, H. (2010). Xtext: Implement your language faster than the quick and dirty way. In *Int. Conf. Companion on Object Oriented Programming Systems Languages and Applications Companion*, pages 307–309, New York, USA. ACM.

Fowler, M. (2010). *Domain-specific languages*. Pearson Education.

GatlingCorp (2017). Gatling. `http://gatling.io/`.

Heyman, J., Byström, C., and Hamrén, J. (2017). Locust. `http://locust.io/`.

Jayasinghe, D., Swint, G., Malkowski, S., Li, J., Wang, Q., Park, J., and Pu, C. (2012). Expertus: A generator approach to automate performance testing in iaas clouds. In *5th Int. Conf. on Cloud Computing*, pages 115–122. IEEE.

JetBrains (2017). MPS - Create your own domain-specific language. `https://www.jetbrains.com/mps/`.

Kramer, A. and Legeard, B. (2016). *Model-Based Testing Essentials - Guide to the ISTQB Certified Model-Based Tester: Foundation Level*. Wiley.

Marand, E. A., Marand, E. A., and Challenger, M. (2015). DSML4CP: a domain-specific modeling language for concurrent programming. *Computer Languages, Systems & Structures*, 44:319–341.

MetaCase (2017). MetaEdit+ Modeler – Supports your modeling language. `http://www.metacase.com/mep/`.

Nakamura, H., Nagano, R., Hisazumi, K., Kamei, Y., Ubayashi, N., and Fukuda, A. (2012). QORAL: An External Domain-Specific Language for Mining Software Repositories. In *4th Int. Workshop on Empirical Software Engineering in Practice*, pages 23–29.

Russell, S. and Norvig, P. (2004). *Inteligência artificial*. CAMPUS - RJ.

Scheuner, J., Leitner, P., Cito, J., and Gall, H. (2014). Cloud Work Bench–Infrastructure-as-Code Based Cloud Benchmarking. In *6th Int. Conf. on Cloud Computing Technology and Science*, pages 246–253. IEEE.

Silva, M., Hines, M. R., Gallo, D., Liu, Q., Ryu, K. D., and Da Silva, D. (2013). Cloudbench: Experiment automation for cloud environments. In *Int. Conf. on Cloud Engineering*, pages 302–311. IEEE.

Solmi, R. (2017). Whole. `https://whole.sourceforge.io/`.

Wende, C., Seifert, M., Heidenreich, F., Karol, S., and Johannes, J. (2017). EMFText. `http://www.emftext.org`.

Desenvolvimento de Software Baseado em Componentes Usando IDL

Yury A. Lima[1], Giliardi Schmidt[1], Elder Rodrigues[1], Maicon Bernardino[1]

[1]Universidade Federal do Pampa (UNIPAMPA)
Código Postal 97.546-550 – Alegrete – RS – Brasil

yuryalencar19@gmail.com, gili.schmidt@hotmail.com
elderrodrigues@unipampa.edu.br, bernardino@acm.org

Abstract. *This article adresses the problems encountered in the characterization of software components, and proposes and improvment using a description language, IDL (Interface Description Language). It is a technology that has as its main objective the description of interfaces at the programing level, making it easier for developers to understant. Its application togheter with Component Based Development significantly impacts development time, plus assisting in the implementation of the components, as well as in the creation of a Middleware for the communication of these components, even providing the means for a hybrid software.*

Resumo. *Este artigo aborda os problemas encontrados na caracterização de componentes de software, e propõe uma melhoria nesta descrição utilizando uma linguagem de descrição. IDL (Interface Description Language) é uma tecnologia que tem como o maior objetivo a descrição de interfaces a nível de programação, facilitando o entendimento do desenvolvedor. Sua aplicação conjunta com o Desenvolvimento de Software Baseado em Componentes impacta significativamente no tempo de desenvolvimento, além de auxiliar tanto na implementação dos componentes, quanto na criação de middlewares para comunicação dos mesmos, possibilitando até a confecção de um software híbrido.*

1. Introdução

O desenvolvimento de software tem como intuito informatizar processos, provendo soluções práticas que possam ser realizadas em um pequeno espaço de tempo. Para isso, várias formas de otimizar o desenvolvimento foram criadas com o objetivo de diminuir os custos, tempo de produção e complexidade, além de colaborar na manutenibilidade, segurança e reúso dos artefatos implementados. O Desenvolvimento de Software Baseado em Componentes (DSBC) [DeRemer and H. Kron 1976] é uma alternativa fundamentada no princípio da divisão e conquista, onde é feita uma separação das responsabilidades na forma de componentes independentes de software. Essa abordagem proporciona um melhor entendimento de cada responsabilidade/funcionalidade e diminuindo o esforço na programação. Além disso, a adoção de DSBC aumentando a qualidade do software já que os componentes depois de desenvolvidos e testados podem ser reutilizados e evoluídos em outros projetos.

Durante o projeto desses componentes deve ser realizado um detalhamento por meio de Modelos de Componentes e uma especificação das *interfaces* a serem providas,

sendo estas etapas cruciais para o entendimento e qualidade dos componentes a serem desenvolvidos. Entretanto, esta especificação pode ser extensa e acarretar dificuldades ao desenvolvedor como por exemplo, atraso no desenvolvimento decorrente a ambiguidade nas descrições dos requisitos dos serviços do componente [Paldês et al. 2016]. Assim, impactando no entendimento e implementação, comprometendo o artefato final.

Para se mitigar estes problemas, a utilização de uma Linguagem de Descrição de Interfaces (*Interface Description Language* - IDL) [OMG 2017] pode ser uma solução viável. A adoção de IDL fornece várias vantagens ao programador, como por exemplo, descrever a interface de modo que a mesma fique mais próxima da sua implementação final e representar a arquitetura e estrutura dos pacotes intensificando também a interoperabilidade entre componentes.

Com o objetivo de explorar as vantagens apresentadas, este trabalho propõe uma abordagem para a especificação das interfaces de componentes utilizando IDL. Destaca-se que em nosso trabalho a caracterização pode descrever interfaces de qualquer componente do sistema, não estando restrita apenas a utilização em arquiteturas de sistemas distribuídos. Esta abordagem apresenta benefícios tanto para o desenvolvedor de componentes, quanto para a criação de *middlewares* responsáveis pela comunicabilidade entre os mesmos. Estas vantagens derivam da simplicidade do uso de IDL e de sua especificação livre de ambiguidade.

Com o objetivo de solucionar tais problemas decorrentes a especificação, o presente artigo foi estruturado como segue. A Seção 2 apresenta o estado da arte em que descreve sobre os componentes de software e suas vantagens, a seguir é exposto sobre o tipo de desenvolvimento baseado em componentes, quando surgiu a proposta e o que esta forma de desenvolver proporciona, bem como conceitua IDL com o objetivo de explicar sobre a linguagem e suas vantagens. Além disso, descreve o impacto de sua utilização em arquiteturas. A Seção 3 é o segmento responsável pela comparação entre como é feita a especificação tradicional e a proposta para utilização de IDL dentro do processo. A Seção 4 apresenta um exemplo de especificação de componente usando DSL. Finalizando, a Seção 5 descreve as considerações finais do estudo.

2. Estado da Arte

2.1. Componentes de Software

O conceito de componente não é unanimemente determinado e existem várias definições do termo. Um conceito geral de componentes reconhecido é que um componente de software deve ser um pacote coerente e independente que pode ser entregue de forma unitária, onde o mesmo pode ser utilizado em conjunto com outros para construir algo maior sem que haja a necessidade de mudanças em sua implementação [D'Souza and Wills 1998].

Componentes podem ser: qualificados, adaptados, aglutinados e atualizados e implementam um interesse específico do sistema, ocultando seus detalhes de confecção e provendo seus serviços por meio de *interfaces*. De acordo com sua especificidade os componentes podem ser mais reutilizáveis do que outros. Sabendo disto estes artefatos podem ser divididos em três grupos (ver Figura 1). Sendo o primeiro os `Genéricos` em que se localizam os componentes que possuem maior reusabilidade, um exemplo destes são módulos que manipulam a interface. Outro grupo é definido pelos que provêm

`Serviços` como impressão de nota fiscal, possuindo uma reusabilidade intermediária dentre os conjuntos. O último conjunto é definido pelos componentes de `Domínio`, estes retratam módulos específicos que possuem a menor reusabilidade como os que são utilizados em clínicas médicas.

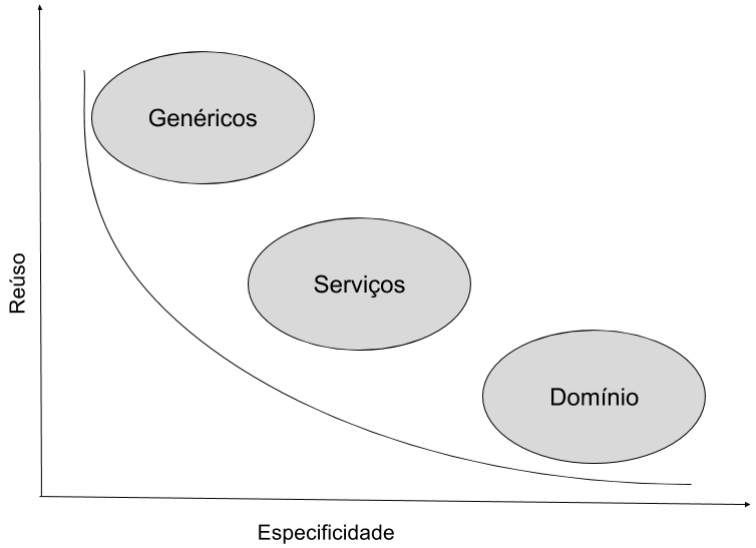


Figura 1. Níveis de reúso dos componentes

Estes componentes tem como principal objetivo aumentar o reúso, e com isso garantir uma maior confiabilidade, qualidade, além de uma redução no tempo de desenvolvimento, custos e complexidade dos sistemas. A gerência de mudanças em um software confeccionado por componentes também é facilitada, tendo em vista que um componente pode ser facilmente adicionado ou substituído, desde que sua interface seja mantida fornecendo novas funcionalidades.

2.2. Desenvolvimento de Software Baseado em Componentes (DSBC)

A utilização de componentes para o desenvolvimento de software foi exposta pela primeira vez em 1968 durante uma conferência da OTAN sobre Engenharia de Software em que foi apresentada a ideia de componentização e reúso por meio do artigo *Mass Produced Software Components* [McIlroy 1968]. Entretanto, o paradigma de desenvolvimento só foi proposto em 1976 por intermédio de uma comparação entre duas formas de implementação de software, em que mencionava a programação utilizando somente componentes e os conectando entre si [DeRemer and H. Kron 1976].

A proposta se fortaleceu em 1980 com a orientação a objetos que proporcionou maior reutilização de código. Este desenvolvimento possui cinco etapas essenciais de acordo com [Brown and Short 1997], são elas: (1) Procurar componentes que possam ser utilizados no desenvolvimento da aplicação; (2) Selecionar os componentes que suprem as necessidades dos requisitos da aplicação a ser desenvolvida; (3) Realizar as adaptações necessárias nos componentes selecionados; (4) Efetuar a composição dos componentes entre si; (5) Atualizar os componentes necessários que demandem modificações.

Com a aplicação destas etapas é possível melhorar a qualidade do software desenvolvido, pois os componentes já foram testados, além do software ser liberado em um

tempo reduzido para o mercado. Outra possibilidade decorrente da utilização de componentes, além do reúso, é a manutenção facilitada do software, pois os componentes podem facilmente serem substituídos, removidos ou adicionados ao produto. De acordo com [Cheesman and Daniels 2000] o princípio adotado para este desenvolvimento é o de divisão e conquista, simplificando a implementação e entendimento do sistema.

A implementação de componentes é uma das perspectivas do desenvolvimento de software baseado em componentes. Estes artefatos devem ser criados de maneira que possam ser reutilizados em outras aplicações, podendo estas pertencerem ao mesmo domínio ou não. Para isto, existem técnicas de criação que podem ser utilizadas visando a melhoria da qualidade dos artefatos criados.

2.3. Interface Description Language (IDL)

A IDL é uma linguagem declarativa que tem como foco a descrição das interfaces de objetos [OMG 2017]. Com a mesma é possível especificar parâmetros de entrada, atributos e operações a serem realizadas pelo componente. É uma linguagem fortemente tipada, baseada em C++, onde se pode estipular constantes tipos e até as exceções que podem ocorrer no módulo durante a execução além de possibilitar especificação dos retornos. Provê também suporte para o pré-processamento, substituição de macros e herança múltipla. A adição de documentação também é possível dentro do mesmo arquivo, como divisão dos módulos e comentários sobre a especificação. É amplamente utilizada para a definição de funções em sistemas distribuídos ou baseados em componentes. Tecnologias como a arquitetura CORBA (*Common Object Request Broker Architecture*) [OMG 2012], juntamente com o DCOM [Redmond 1997], o utilizam para chamada de processos remotos, possibilitando uma maior interoperabilidade entre os componentes além da viabilidade de criação de softwares híbridos.

Atualmente, existem especificações diferentes para IDL, tendo como padrão a elaborada pela OMG que é direcionada para utilização em conjunto com o CORBA. Uma das alternativas de uso disponíveis é o MIDL (*Microsoft Interface Definition Language*), que também é utilizado juntamente com o RPC (*Remote Procedure Call*) para chamada de procedimentos remotos e interoperabilidade entre componentes [Myerson 2002]. É possível encontrar compiladores disponibilizados pelos fornecedores, os quais tem como objetivo a conversão do arquivo IDL para uma estrutura em uma das linguagens de programação suportadas pela especificação escolhida.

2.4. Impacto da IDL na Arquitetura

Em sistemas distribuídos os componentes podem ser escritos em diferentes linguagens de programação e utilizar estruturas de dados incompatíveis. Neste caso, o uso de *middlewares* se torna necessário para confecção dos mesmos. Com a utilização de arquivos IDL para especificação dos componentes, viabiliza uma clara definição dos módulos, possibilitando sua implementação de forma simultânea com os *middlewares* assegurando sua integração. A importância da utilização de IDL é ainda maior quando o *middleware* é orientado a objetos, pois a interação entre componentes é realizada pela invocação de métodos, que também podem ser descritos utilizando IDL, e podem ser traduzidos para qualquer outra linguagem mapeada pela especificação escolhida.

A arquitetura CORBA é baseada em componentes distribuídos que utiliza o modelo cliente-servidor. Neste tipo de arquitetura cada componente pode atuar como cliente

e/ou servidor. Para permitir a comunicação entre estes componentes é utilizado *middlewares* orientado a objetos. Dentro do CORBA estes *middlewares* são denominados ORB (*Object Request Broker*), e têm como fator crucial efetuar chamadas de métodos remotos dentro do sistema [Link et al. 2013]. Com esta finalidade o ORB utiliza das especificações das interfaces dos componentes em IDL no formato padrão definido pela OMG. Assim, não necessitam conhecer a tecnologia sob a qual o módulo foi implementado para realizar sua invocação.

Em uma arquitetura cliente/servidor para invocar alguma operação em outro componente do sistema é necessário conhecer detalhes do servidor, como os parâmetros de entrada, retorno e as operações a serem realizadas. Para isto, dentro do CORBA é utilizado o ORB que tem como função ser responsável por todos os mecanismos necessários para a localização da implementação do objeto invocado, garantindo que assim que o mesmo possa receber a requisição e transferir os resultados de volta ao cliente [Link et al. 2013].

3. Uma Proposta de Especificação de Componentes Utilizando IDL

No DSBC é realizado a especificação dos componentes no repositório, isto é relevante para que seja possível ter um melhor controle dos mesmos. É importante conter uma quantidade de informações suficientes, pois assim é possível recuperar esses artefatos por meio de mecanismos de busca [Redolfi et al. 2004]. Com o objetivo de suprir essa necessidade a especificação das interfaces podem ser realizadas com arquivos IDL, facilitando a busca como acontece na arquitetura CORBA com um mecanismo semelhante as ORB's, além de também garantir a interoperabilidade e portabilidade dos componentes quando desenvolvidos. Caso aconteça a utilização de alguma arquitetura que faça seu uso dentro das comunicações entre os componentes, sua utilização acaba sendo potencializada no desenvolvimento.

O modo tradicional de se caracterizar e especificar as interfaces de componentes é realizado por meio de diagramas e tabelas contendo informações sobre o artefato. Essa abordagem é interessante por conter um detalhamento em linguagem natural sobre os serviços que os componentes oferecem. Esta abordagem, no entanto, pode acarretar problemas como ambiguidade e omissão de informações a nível de implementação, ocasionando diversos problemas para a equipe de desenvolvimento, tais como: aumento do tempo gasto no desenvolvimento e diminuição na qualidade do componente criado. A utilização de IDL é uma solução para os problemas presentes no modo tradicional, além de proporcionar um melhor entendimento do funcionamento dos componentes também contém o necessário para a comunicação com os mesmos, impactando tanto no desenvolvimento de componentes quanto na implementação do sistema que os utiliza.

Com a especificação das interfaces definidas em IDL é possível saber quais são as entradas e saídas dos métodos de forma simplificada e rápida, impactando no tempo de desenvolvimento das conexões entre os componentes. Tendo em vista que desta forma a especificação é realizada de forma independente da linguagem de programação, é possível representar um componente implementado em qualquer tecnologia. A adoção de arquiteturas que utilizam a linguagem juntamente com mecanismos de chamada remota possibilita a criação de softwares híbridos e melhorias relacionadas a portabilidade e comunicabilidade entre os componentes. Com a utilização de IDL é viável o gerenciamento dos módulos e adicionar comentários proporciona uma especificação ainda mais facilitada

para o desenvolvedor do sistema.

Como IDL é uma linguagem baseada em C++, o entendimento da notação pela equipe de desenvolvimento não é afetado drasticamente com a adoção da tecnologia, sabendo que esta linguagem é amplamente conhecida e utilizada também como base para o desenvolvimento de outras como Java e C#, que por sua vez também são tecnologias aplicadas na implementação de sistemas baseados em componentes [Sebesta 2009]. No caso do desenvolvimento de componentes a especificação se torna ainda mais eficaz, já que possui compiladores que geram código não funcional a partir da tecnologia, definindo assim uma estrutura para o componente. Dependendo do compilador utilizado, o código gerado é baseado na arquitetura, usado comumente para a arquitetura CORBA.

4. Exemplo de Componente Especificado em IDL

Com a finalidade de ilustrar um componente especificado utilizando IDL, a ferramenta da linha de produtos PLeTs [Rodrigues 2013] foi escolhida como exemplo, já que seu desenvolvimento ocorreu baseado no uso de componentes. Este software tem como funcionalidade a criação de casos de teste para sistemas web baseado em modelos formais, *e.g.* Máquinas de Estados Finitos. Seu diagrama de componentes pode ser visualizado na Figura 2.

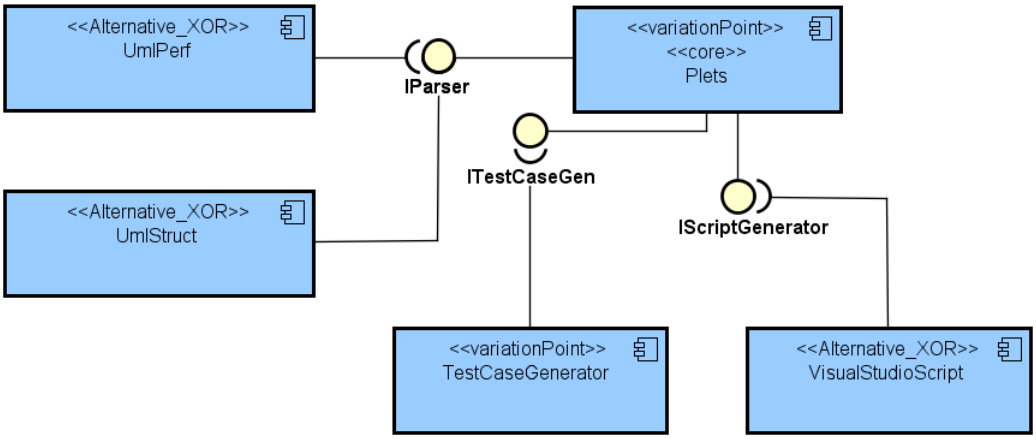


Figura 2. PLeTs: Parte do Diagrama de Componentes UML- Baseado em [Rodrigues 2013]

O módulo escolhido para a representação aplicando IDL foi o *IParser* utilizando a notação UML (*Unified Modeling Language*) [Booch et al. 2000]. Esse componente tem como função a análise de um arquivo no formato XMI, com o objetivo de retirar os dados referentes aos modelos pertencentes ao sistema onde serão executados os testes gerados. No diagrama de componentes da Figura 2 é possível visualizar alguns dados sobre o artefato, entretanto, falta informações a nível de implementação. Uma alternativa pode ser o uso de diagramas de classe, mas como a utilização de IDL acarreta também várias outras melhorias, sua aplicação pode ser mais benéfica ao desenvolvimento. O exemplo pode ser visualizado por meio do trecho de código da Figura 3, o qual retrata o componente *ParserUML* com base na interface *IParser*.

Com a especificação da interface *IParser* é perceptível uma maior quantidade de detalhes relacionadas ao funcionamento, diferente do diagrama encontrado na Figura 2.

```
#include<UML.idl>
module Project {
    module ExampleParserXMI{
        interface ParserUML{
            void managerParser(in string pathArchiveXMI);
            Uml::UMLElement createActor(in string partialArchiveXMI);
            Uml::UMLElement createUseCase(in string partialArchiveXMI);
            Uml::UMLElement createInitAction(in string partialArchiveXMI);
            Uml::UMLElement createFinalAction(in string partialArchiveXMI);
            Uml::UMLElement createAction(in string partialArchiveXMI);
            Uml::UMLElement createFork(in string partialArchiveXMI);
            Uml::UMLElement createJoin(in string partialArchiveXMI);
            Uml::UMLAssociation createAssociation(in string partialArchiveXMI);
            Uml::TypeAssociation createInclude(in string partialArchiveXMI);
            Uml::TypeAssociation createExtend(in string partialArchiveXMI);
            Uml::TypeAssociation createGeneralization(in string partialArchiveXMI);
            Uml::UMLElement createDecisionNode(in string partialArchiveXMI, in Uml::
                TypeDiagram typeDiagram);
            boolean addDiagram(in Uml::UMLDiagram newDiagram, in Uml::TypeDiagram
                typeDiagram, in string uniqueName);
            Uml::UMLDiagram getDiagram(in Uml::TypeDiagram typeDiagram, in string
                uniqueName);
        };
    };
};
```

Figura 3. Especificação IDL do componente *ParserUML* com base em *IParser*

Tendo em vista que é possível estruturar os pacotes, *interfaces*, métodos, entradas, saídas e até mesmo a inclusão de outros módulos ao projeto. É importante salientar que sua quantidade de detalhes pode superar a de um diagrama de classes comum, pois além das possibilidades descritas também é possível o uso de comentários em conjunto à especificação.

5. Considerações Finais

A utilização de IDL no Desenvolvimento de Software Baseado em Componentes (DSBC) pode se mostrar superior ao modelo tradicional no aspecto especificação e entendimento do componente para o desenvolvedor, impactando positivamente tanto na criação quanto na utilização. Além de possibilitar a implementação do mesmo em paralelo com o *middleware* necessário para a integração. Outro fator positivo seria a facilidade na busca do repositório, sua importância é justificada em grandes empresas decorrente da quantidade elevada de componentes, criando a necessidade de um mecanismo de busca que pode utilizar IDL com esta finalidade, proporcionando uma maior praticidade, e tornando viável a procura do componente de acordo com seu comportamento. Assim, podendo auxiliar na redução do tempo de desenvolvimento de um sistema orientado a componentes.

É importante destacar que a aplicação de IDL na especificação de componentes não é independente. Mesmo que IDL proporcione uma série de melhorias a linguagem não pode substituir toda a especificação tradicional. Tendo em vista que esta também é aplicada no entendimento do componente para as demais partes interessadas, *i.e. stakeholders*. A linguagem natural mesmo que possa conter ambiguidades é o modo de proporcionar uma melhor assimilação no momento da negociação, em especial com clientes não técnicos. Diagramas e tabelas explicativas também podem ser mais expressivas nesse cenário do que somente o uso de IDL.

Assim, o uso da tecnologia deve ser realizado de modo conjunto com a especificação tradicional, pois devido ao seu impacto no desenvolvimento é possível uma melhor busca, compreensão e melhoria do artefato gerado. Além disso, a diminuição da ambiguidade das tabelas explicativas, tendo em vista que os arquivos podem ser utilizados para verificação e validação tanto do documento de especificação quanto do código implementado. Por fim, como trabalho futuro será realizada a aplicação desta proposta dentro de um cenário real de desenvolvimento. Assim, com o propósito de avaliar o impacto que a equipe poderá ter com a adoção desta proposta na especificação para softwares baseados em componentes.

Referências

Booch, G., Rumbaugh, J., and Jacobson, I. (2000). UML, Guia do Usuário.

Brown, A. W. and Short, K. (1997). On components and objects: the foundations of component-based development. In *Assessment of Software Tools and Technologies, 1997., Proceedings Fifth International Symposium on*, pages 112–121. IEEE.

Cheesman, J. and Daniels, J. (2000). *UML components: a simple process for specifying component-based software*. Addison-Wesley Longman Publishing Co., Inc.

DeRemer, F. and H. Kron, H. (1976). Programming-in-the-large versus programming-in-the-small. SE-2:80 – 86.

D'Souza, D. F. and Wills, A. C. (1998). *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Link, E., Alexandre, E. B. P., Wolf, J. L., and Strzykalski, M. S. (2013). Uma Introdução ao CORBA.

McIlroy, M. D. (1968). Mass-produced software components. *Proc. NATO Conf. on Software Engineering, Garmisch, Germany*.

Myerson, J. M. (2002). *The complete book of middleware*. CRC Press.

OMG (2012). Information technology: Common Object Request Broker Architecture (CORBA), Interfaces. Formal/2012-05-03.

OMG (2017). Interface Definition Language. Version 4.1.

Paldês, R., Calazans, A., Mariano, A., Castro, E., and de Souza da Silva, B. (2016). A utilização da linguagem natural na especificação de requisitos: um estudo por meio das equações estruturais.

Redmond, F. (1997). *DCOM: Microsoft Distributed Component Object Model*. Professional Series. IDG Books Worldwide.

Redolfi, G., de Araujo Spagnoli, L., Bastos, R. M., Cristal, M., and Espindola, A. P. (2004). Especificando informações para componentes reutilizáveis. *Porto Alegre, Abril*.

Rodrigues, E. d. M. (2013). PLeTs: a product line of model-based testing tools.

Sebesta, R. W. (2009). *Conceitos de linguagens de programação*. Bookman Editora.

Uma Ferramenta para Sincronização de Conteúdos Produzidos pelo Software QGIS

Madson Verdi Junior, Claudio Schepke

[1]Universidade Federal do Pampa (UNIPAMPA) - Campus Alegrete
Av. Tiarajú, 810, 97546-550, Alegrete – RS – Brasil

madson.verdi@alunos.unipampa.edu.br, claudioschepke@unipampa.edu.br

Abstract. *Geographic Information System (GIS) is a computational resource that allows the management of information through data referenced from a coordinate system. Such systems store the information using two modes: files or Database (DB). The EIRE group stores energy resource data in QGIS (a GIS software) projects using files from a local server. To make the synchronization among local file data used by QGIS and a DB possible, it is necessary to implement a plugin for QGIS. This work describes the development of this plugin using software engineering techniques such as requirements analysis, prototyping, among others.*

Resumo. *Um* Sistema de Informações Geográficas *(SIG) é um recurso computacional que permite o gerenciamento de informações por meio de dados referenciados a partir de um sistema de coordenadas. Tais sistemas armazenam as informações utilizando dois modos: arquivos ou* Banco de Dados *(BD). O grupo EIRE armazena dados de recursos energéticos em projetos do SIG QGIS em arquivos de um servidor local. Para possibilitar a sincronização dos dados de arquivos locais utilizados pelo QGIS com um BD é necessário a implementação de um plugin para o SIG em questão. Este trabalho descreve o desenvolvimento deste plugin utilizando técnicas de* Engenharia de Software *como análise de requisitos, prototipagem, entre outras.*

1. Introdução

O *Grupo Exploração Integrada de Recursos Energéticos* (EIRE)[1] trabalha com a gestão de recursos energéticos utilizando um *Sistema de Informações Geográficas* denominado QGIS. A informação gerada e trabalhada pelo grupo está armazenada em arquivos. Estes arquivos estão armazenados em *Hard Driver*s externos ou diretamente nos computadores do grupo, e, em alguns casos, compartilhados em uma rede interna. A necessidade de um trabalho conjunto entre mais de um integrante do grupo de pesquisa exige a troca de informações geradas por cada membro. No modelo utilizado atualmente pelo grupo, a atualização contínua dos dados é dificultosa. O problema de compartilhamento

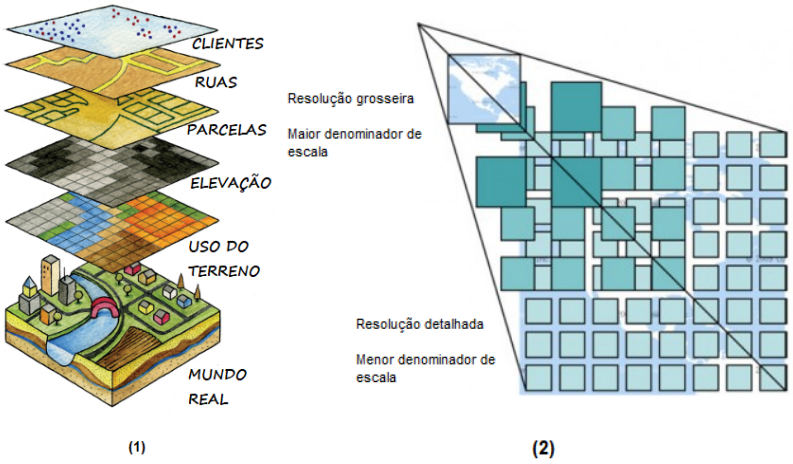
[1]*Grupo Exploração Integrada de Recursos Energéticos* ou EIRE é um grupo de pesquisa criado em 2011 e apresenta como principais objetivos: consolidar ações de ensino, pesquisa e extensão no campo do planejamento e gestão de recursos energéticos; desenvolver métodos e modelos para a simulação de gerenciamento de energia pelo lado da oferta e demanda; atuar na pesquisa e desenvolvimento de equipamentos de elevada eficiência energética; contribuir com ações de extensão e ensino para o desenvolvimento regional; pesquisa e desenvolvimento em fontes renováveis de energia.

de informação ocorre também em outros ambientes que utilizam o software para incluir conteúdo. Este problema, porém, só é observado quando mais de um usuário interage com o mesmo arquivo.

Este trabalho aborda a forma de centralizar a informação do software QGIS. O objetivo da criação do *plugin* é sincronizar os dados de forma fácil e clara, ressaltando também como a informação será armazenada em um banco de dados. Desta forma, os dados poderão ser facilmente difundidos em partes, ou totalmente, para outros grupos, facilitando o trabalho em conjunto, através de protocolos: WMS e WTMS.

A parte da esquerda da Figura 1 exemplifica a aplicação do padrão WMS onde as camadas, *CLIENTES*, *RUAS*, *PARCELAS*, *ELEVAÇÃO* e *USO DO TERRENO* representam os dados carregados de forma dinâmica, sobre um mapa real, representado no exemplo como a última camada, *MUNDO REAL*. Já a da parte direita da Figura 1 abstrai o padrão *Web Map Tile Service Implementation Standard* (WTMS), onde blocos podem ser carregados em diferentes níveis de detalhamento. Cada bloco de imagem pode conter as informações de todas camadas, já carregados em forma de imagens, essas divididas em pequenos pedaços.

Figura 1. Exemplo dos padrões (1) WMS (*Web Map Service Interface Standard*) e (2) WTMS (*Web Map Tile Service Implementation Standard*)



Este trabalho tem como objetivo prover um melhor aproveitamento dos dados catalogados pelo grupo de pesquisa EIRE. A motivação para o trabalho é a necessidade de centralizar as informações geradas de forma fácil e ágil. Isso é possível através da adoção de um banco de dados centralizado, onde a informação poderá ser acessada através de diversos dispositivos, incluindo móveis até computadores convencionais e, principalmente, garantir que todos os usuários do QGIS, envolvidos em um determinado projeto, possuam a informação atualizada. Para isso, as informações geradas através do software QGIS serão sincronizadas com um banco de dados que poderá ser acessado por todos envolvidos em determinado projeto.

2. Metodologia

2.1. Análise e Revisão de requisitos

Inicialmente foi necessário negociar e levantar os requisitos, conforme descrito por [Boehm et al. 1998]. A negociação é constituída pelas seguintes atividades:

1. identificação dos principais interessados no sistema ou subsistema;
2. determinação das "Condições de ganho" dos interessados;
3. negociação das condições gerais de ganho dos interessados para reconciliá-las em um conjunto de condições "ganha-ganha" para todos os envolvidos.

Com base nos requisitos apresentados acima e a descrição apresentada por [Pressman 2011], foi possível definir dois itens:

- **Análise de requisitos do problema**: Avaliar quais as tarefas que devem ser realizadas visando definir como proceder com o projeto;
- **Revisão dos requisitos analisados**: Após a análise dos requisitos é necessário avaliar se estes estão todos de acordo, conforme [Pressman 2011, p. 146-147]. Uma revisão nos requisitos pode identificar possíveis problemas a serem enfrentados e também gerar novos requisitos que não foram identificados anteriormente.

2.2. Implementação da solução

O *plugin*, também denominado *complemento*, implementado, é responsável pela sincronização dos dados entre cliente e servidor. O servidor irá processar as requisições geradas por este *plugin*, a fim de manter os dados atualizados entre o cliente e o servidor.

Para a implementação, os seguintes passos foram executados:

1. **Protótipos de tela para o *plugin***: A prototipação de uma solução, segundo [Pressman 2011, p.62], é necessária quando o desenvolvedor não se sente seguro na interação homem/máquina e esta interação necessita ser detalhada.
2. **Geração das consultas**: o comando *shp2pgsql* [Ramsey et al. 2005] possibilita gerar os arquivos com a extensão *.sql* a partir dos dados dos arquivos com a extensão *.shp*. O oposto é realizado através do comando *pgsql2shp* [Ramsey et al. 2005].
3. **Servidor da aplicação**: o servidor recebe as requisições e encaminha respostas para um *cliente*. Seu acesso será realizado através de PostGIS, que é uma extensão para PostgreSQL. Nele é possível armazenar as informações no banco de dados. O acesso ao banco de dados é realizado quando o computador estiver conectado a uma rede onde o servidor esteja visível.
4. ***Plugin***: funciona como uma interface para o usuário utilizar a solução. Este está diretamente ligado ao *Software QGIS*, funcionando como um complemento, adicionando a funcionalidade proposta neste trabalho.

3. Resultados

Os resultados gerados foram divididos em alguns grupos, descrito na apresentação da metodologia, e são discriminados nas sub-seções abaixo.

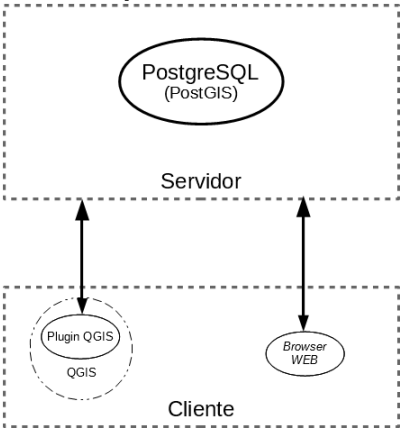
3.1. Análise e Revisão de requisitos

Durante o processo de análise de requisitos foi possível identificar a necessidade de implementação das seguintes funcionalidades:

- cliente (*plugin* para o QGIS):
  - capturar os eventos ocorridos na edição de dados de um projeto no QGIS;
  - enviar as alterações para o servidor;
  - receber as atualizações do servidor;
  - o cliente não possui banco de dados.
- cliente (*borwsers*):
  - visualizar mapas através de *browsers* na WEB e intranet;
  - a visualização deve possuir controle de acesso.
- servidor:
  - armazenar os mapas atualizados;
  - receber atualização de clientes;
  - enviar atualização para clientes.
  - WEB - QGIS Server:
    - permite a utilização dos padrões WMS e WTMS.

Uma simplificação desta análise pode ser visualizada na Figura 2.

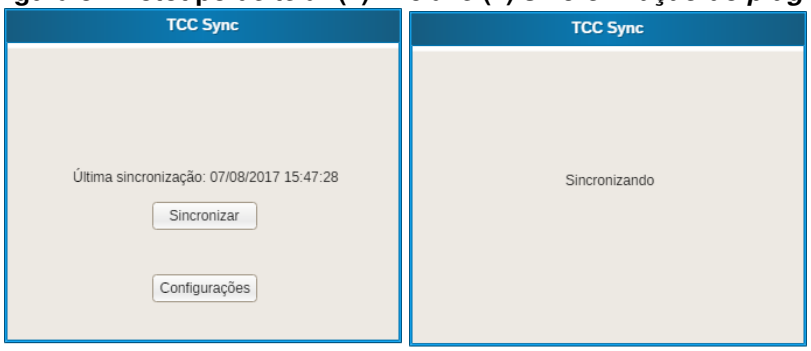
Figura 2. Exemplo da estrutura do servidor



Durante a revisão dos requisitos levantados, conforme descrito anteriormente nesta seção, foi possível identificar alguns desafios:

- a sincronização deverá ocorrer quando o *plugin* for carregado;
  - para realizar o envio, o arquivo deverá ser convertido para o formato *.sql*;
  - para realizar o recebimento, o arquivo deverá ser convertido para o formato *.shp*.
- o servidor deverá armazenar todos os scripts recebidos e repassar para todos clientes, se for necessário;
- o cliente deverá manter uma data de atualização para verificar qual ou quais scripts devem ser retornados do servidor;
- o nome do banco deverá ser armazenado junto ao projeto;
- o servidor deverá receber o script e atualizar/criar as suas bases de dados;
- os usuários necessitam de níveis de acesso.

Figura 3. Protótipo de tela: (1) inicial e (2) *sincronização* do *plugin*

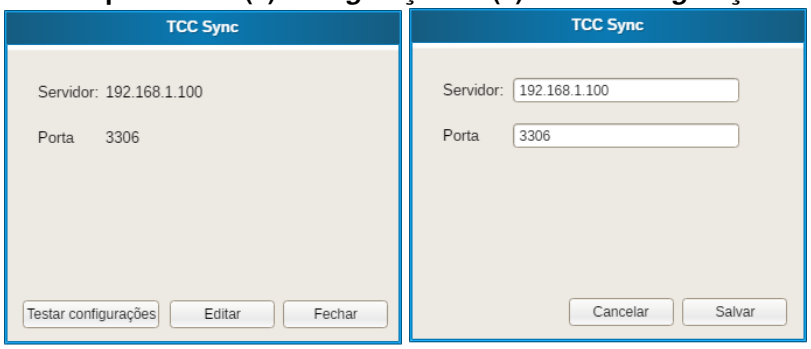


3.2. Prototipação da solução

Os protótipos de tela gerados para exemplificar a interação homem/máquina foram gerados utilizando o software *Pencil Project*[2].

A parte (1) da Figura 3 representa o protótipo da tela inicial do *plugin*, ao clicar em *Sincronizar* espera-se que seja carregada a tela descrita na parte (2) da Figura 3 e ao final da sincronização deve retornar para a parte (1). Caso tenha ocorrido erro no processo uma mensagem de erro deve ser exibida no local da data da última atualização.

Figura 4. Protótipo da tela: (1) *configurações* e (2) *editar configurações* do *plugin*



A opção *Configurações*, disponível na tela inicial, deve carregar a tela descrita na parte (1) da Figura 4. Nesta tela deverão ser listadas todas as configurações que o *plugin* necessita para realizar a conexão com o servidor, permitindo que o usuário altere essas configurações de forma visual, conforme a parte (2) da Figura 4. As informações de configuração devem ser carregadas e armazenadas no arquivo de configuração do *plugin*.

3.3. Implementação da solução

A implementação da solução ocorreu em duas fases, primeiro a fase de validação da ideia e em um segundo momento a implementação da solução por completo.

[2]É um sofware de código aberto sobre a licença *GNU Public License version 2* [License 1989] e pode ser baixado gratuitamente pelo link https://pencil.evolus.vn/Downloads.html

3.3.1. Primeira fase

Durante a implementação da solução do cliente foi possível identificar que um arquivo em formato *.shp* com aproximadamente 10 MB quando convertido para formato *.sql* ultrapassava os 35 MB. Desta forma buscou-se gerar arquivos menores o que por sua vez gerou os seguintes desafios:

- **como gerar arquivos menores**: utilizando um padrão utilizado em sistemas de versionamento, encaminhando somente as alterações realizadas nos arquivo. Para isso duas alternativas foram encontradas:
  - em sistemas *Unix* o comando nativo *diff* realizava a tarefa sem problemas, porém em sistemas *Windows* não era uma alternativa e logo foi descartado;
  - o *Python*[3] possui uma biblioteca, *difflib* [van Rossum and Drake 2017], com o mesmo propósito de gerar a diferença entre um arquivo de entrada e outro de saída.
- **como enviar**: é necessário encaminhar o resultado da diferença para o servidor, o Script 2 realiza a conversão para o formato *.sql*;
- **como receber**: o *Cliente* recebe somente as atualizações que são mescladas com seu arquivo original e convertidas novamente para o formato *.shp*, a conversão é realizada através do Script 3;
- **processamento do arquivo recebido**: após o *Cliente* receber os comandos *.sql* de atualização eles são adicionados em um arquivo já gerado. Posteriormente este arquivo é convertido para *.shp*, nesta etapa o arquivo de configuração é gerado e um exemplo pode ser visualizado no Script 1.

A Figura 2 mostra a estrutura montada entre cliente e servidor, exemplificando a troca de informações entre as partes, onde o cliente encaminha os dados e o servidor recebe. As informações são atualizadas no servidor, assim o próximo cliente que tentar realizar a sincronização deverá, primeiramente, atualizar os dados locais conforme as informações armazenadas no servidor. O script em *Python* gera um novo arquivo local com as modificações realizadas no cliente para ser encaminhado para o servidor.

As consultas são geradas pelo cliente da aplicação e convertidas para o formato local correto, um arquivo de configuração é criado no cliente para auxiliar no gerenciamento dos arquivos que estão sendo trabalhados, um trecho deste arquivo de configuração pode ser verificado no Script 1.

```
1 [Rodovias_RioGrandeDoSul_2015_1]
2 local = 2017-06-19 14:50:49.938528
3 file_original = ../arquivos_entrada/MapaRodoviarioRS/Rodovias_RioGrandeDoSul_2015_1.shp
4 server = 2017-06-19 14:50:50.758714
5 sql_original = ../arquivos_saida/Rodovias_RioGrandeDoSul_2015_1_original.sql
6 file_temp = ../arquivos_saida/Rodovias_RioGrandeDoSul_2015_1.shp
7 sql_updated = ../arquivos_saida/Rodovias_RioGrandeDoSul_2015_1.sql
```

Script 1. Trecho de arquivo de configuração gerado após atualização

O servidor da aplicação consiste em um servidor com um banco de dados PostgreSQL com a extensão PostGIS.

O cliente foi implementado em *Python* para facilitar a implementação de um *plugin* para o QGIS. A implementação do cliente roda basicamente em cima de duas funções.

[3]Python é uma linguagem de programação de alto nível [Venners 2003], interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte.

A primeira, descrita no Script 2, realiza a conversão do arquivo binário em *.shp* para o formato *.sql*. A segunda função, descrita no Script 3, recupera a informação armazenada no servidor e gera o arquivo *.shp*.

```
def create_sql(bin_folder, input_file, output_files_folder, filename, sufix = ''):
  # nomes de arquivos de origem e atual - SQL
  input_sql_file_name = output_files_folder + DS + filename + sufix + '.sql'
  # print input_sql_file_name
  # gerando arquivos para comparacao
  args_input = bin_folder + DS + 'shp2pgsql.exe -W "latin1" -e ' + input_file + ' > ' + input_sql_file_name

  # print args_input

  try:
    retcode = subprocess.call(args_input, shell=True)
    if retcode < 0:
      print >>sys.stderr, "Child was terminated by signal", -retcode
    else:
      print >>sys.stderr, "Child returned", retcode
  except OSError as e:
    print >>sys.stderr, "Execution failed:", e

  return input_sql_file_name
```

Script 2. Conversão de .shp para .sql

```
def create_shp(bin_folder, output_files_folder, filename, sufix = ''):
  # nomes de arquivos de origem e atual - SQL
  shp_file_name = output_files_folder + DS + filename + sufix
  # print input_sql_file_name
  # gerando arquivos para comparacao
  args_input = bin_folder + DS + 'pgsql2shp.exe -f "' + shp_file_name + '" -h ' + config_get('SERVER', 'pg_host') + \
    ' -P ' + config_get('SERVER', 'pg_pass') + \
    ' -u ' + config_get('SERVER', 'pg_user') + \
    ' ' + config_get('SERVER', 'pg_database') +\
    ' ' + filename.lower()

  print args_input

  try:
    retcode = subprocess.call(args_input, shell=True)
    if retcode < 0:
      print >>sys.stderr, "Child was terminated by signal", -retcode
    else:
      print >>sys.stderr, "Child returned", retcode
  except OSError as e:
    print >>sys.stderr, "Execution failed:", e

  config_set(filename, 'file_temp', shp_file_name + '.shp')

  return shp_file_name
```

Script 3. Conversão de .shp para .sql

A fim de realizar a validação da implementação da solução, uma bateria de testes foi realizada utilizando duas VMs[4] , assim, permitindo uma simulação entre 2 clientes. Sendo que o **Cliente 1** é o computador Host das VMs. O seguinte cenário foi executado:

1. Cliente 1 gera um novo mapa e sincroniza com servidor;
2. Cliente 2 realiza o recebimento desse novo mapa;
3. Cliente 2 atualiza localmente o mapa;
4. Cliente 2 realiza sincronização com o servidor;
5. Cliente 1 atualiza localmente o mapa;
6. Cliente 1 realiza sincronização com o servidor;
   (a) Recebe *.sql* do servidor;
   (b) Gera *.sql* localmente com base no arquivo do mapa;
   (c) Gera o arquivo de diferença do arquivo *.sql*;
   (d) Executa o script *.sql* no servidor, somente com as modificações geradas;

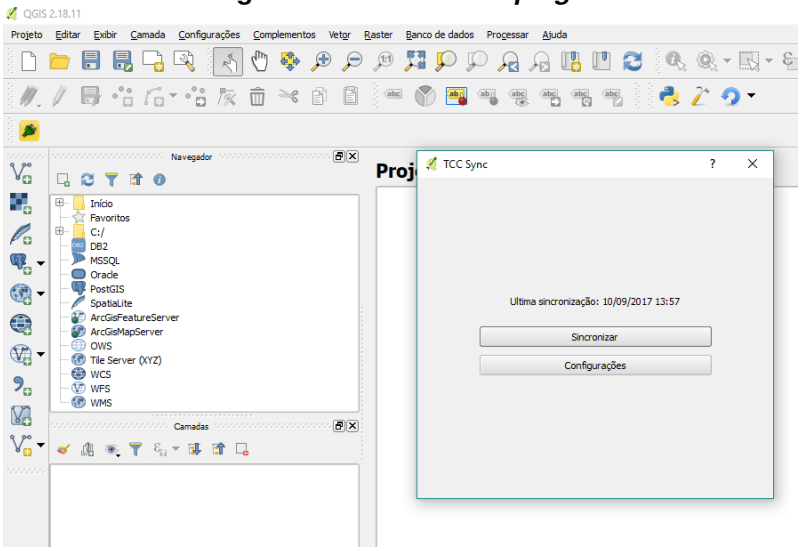
A solução implementada se comportou conforme o esperado, o que possibilitou a implementação da segunda fase do projeto, descrita na Seção 3.3.2.

[4]Uma máquina virtual (*Virtual Machine ou Máquina Virtual* – VM) pode ser definida como ”uma duplicata eficiente e isolada de uma máquina real”.

3.3.2. Segunda fase

A segunda fase da implementação da solução consistiu em gerar uma interface para o usuário interagir com a aplicação. A interface foi implementada seguindo o modelo prototipado e suas interações.

Figura 5. Tela inicial do *plugin*



A Figura 5 é referente a tela inicial do *plugin* implementado, onde os botões *Sincronizar* e *Configurações* redirecionam para suas devidas funcionalidades.

4. Considerações finais e Trabalhos futuros

A implementação do *plugin* foi finalizada e encontra-se na etapa de testes com o usuário final. Agora é necessário um agendamento com o EIRE para dar continuidade ao trabalho O software também será validado utilizando alguma empresa do *Parque Científico e Tecnológico do Pampa* (PampaTec).

Referências

Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., and Madachy, R. (1998). Using the winwin spiral model: a case study. *Computer*, 31(7):33–44.

License, G. G. P. (1989). Version 2, june 1991. *Copyright (C)*, 1991:02111–1307.

Pressman, R. S. (2011). Engenharia de software: uma abordagem profissional. 7ª edição. *Ed: McGraw Hill*.

Ramsey, P. et al. (2005). Postgis manual. *Refractions Research Inc*.

van Rossum, G. and Drake, F. L. (2017). disponível livremente na internet em http://www. python. org. br/wiki.

Venners, B. (2003). The making of python. *Artima. com. http://www. artima. com/intv/python. html [accessed 2003-12-09]*.

Identificação de características de sistemas legados a partir da análise de conteúdo da literatura

Daniele Martins, Alex Chervenski, Andréa S. Bordin

Curso de Bacharelado em Engenharia de Software – Universidade Federal do Pampa (UNIPAMPA) – Campus Alegrete
97546-550 – Alegrete– RS– Brasil

{danimarttiins,alex.chervenski}@gmail.com,andreabordin@unipampa.edu.br

Abstract. *Legacy systems tend to consume a lot of resources and so, requires to be well managed. Deciding whether a system is becoming legacy or not is a decision that involves identifying the characteristics of such a system. This paper aims to present the mapping of features found in the literature's definitions of legacy systems. For this, a systematic process of search, extraction and analysis of the definitions of legacy systems found between the years 1995, 2000, 2005, 2010 and 2015 was performed. The results analysis reveal that in the studied period, the characteristics with the highest frequency of occurrences were, to be vital to the organization, to be a obsolete technology and to have no documentation.*

Resumo. *Sistemas legados tendem a consumir muitos recursos e por isso devem ser bem gerenciados. Decidir se um sistema é ou está se tornando legado é uma decisão que envolve a identificação das características de tal sistema. Este artigo tem como objetivo apresentar o mapeamento de características encontradas nas definições de sistemas legados utilizadas na literatura. Para isso foi realizado processo sistemático de busca, extração e análise de conteúdo das definições de sistemas legados encontradas nos anos de 1995, 2000, 2005, 2010 e 2015. A análise dos resultados revela que as características relacionadas ao uso de tecnologia obsoleta, ser vital para a organização e não possuir documentação são as que possuem maior frequência de ocorrências no período pesquisado.*

1. Introdução

[Ning et al. 1994] descreve sistemas legados como sistemas que inibem o crescimento de uma organização e sua capacidade de mudança, [Bennett 1995] diz que são sistemas que realizam tarefas úteis para a organização, mas foram desenvolvidos com tecnologias atualmente consideradas obsoletas. Outra definição diz que sistemas legados "são sistemas com valor crítico para o negócio das empresas desenvolvidos há cinco ou mais anos" [Umar 1997]. Por outro lado, [Juric et al. 2000] definem que “é qualquer sistema que, independentemente da idade ou arquitetura, ainda é útil e está em

uso".

As quatro definições mostram diferentes características e evidenciam que não há um consenso sobre o que é um sistema legado. De acordo com [O'Byrne e Wu 2000] não saber ao certo quais são essas características torna difícil o processo de identificar se um sistema é ou está se tornando realmente "legado". Ainda, segundo esses autores, é difícil determinar com precisão se um sistema é ou não é legado, logo a pergunta mais adequada poderia ser "Em qual o grau de sistema legado um determinado sistema se encontra? ".

A identificação desse tipo de sistema é importante para uma posterior avaliação e tomada de decisão sobre o que fazer com o sistema. Este é um grande desafio para as organizações, visto que tais sistemas costumam consumir muitos recursos para mantê-los. [Ramsom et al. 1998] e [Sommerville 2011] apontam que possíveis ações, resultantes de um processo de avaliação de sistemas legados, incluem continuar com a manutenção, reengenhar o sistema, substituir ou aposentar o sistema.

Este artigo objetiva apresentar o mapeamento das características encontradas com mais frequência nas definições de sistemas legados utilizadas na literatura. Os resultados desse mapeamento apontam as características mais utilizadas pelos autores, as quais por sua vez, podem ser utilizadas como referência para a identificação de sistemas que estão se tornando ou já são sistemas legados.

2. Metodologia

O processo sistemático de busca de definições de sistemas legados ocorreu entre maio a setembro de 2016 e foi conduzido com base nas etapas de planejamento e execução de revisão sistemática propostas por [Kitchenham 2004]. Na fase de planejamento foi identificada a inexistência de pesquisas relacionadas ao mapeamento de definições de sistemas legados e foi elaborado um protocolo no qual foram definidas a questão de pesquisa, a estratégia de busca e os critérios de inclusão e exclusão dos estudos.

A questão de pesquisa definida foi: "Quais as definições de sistema legados utilizadas na literatura?". A estratégia de busca e os critérios são apresentadas na Tabela 1:

Tabela 1. Estratégia de busca e Critérios de Inclusão e Exclusão

Fonte de busca	Google Acadêmico
Idioma de estudo	Inglês
String de busca	("legacy system is") OR ("legacy software is") OR

	("legacy systems are") OR ("legacy software are")
Critérios de Inclusão (CI)	(a) Conter uma definição de sistemas legados no abstract ou no corpo do artigo; (b) Ter sido publicado no ano de 1995 ou 2000 ou 2005 ou 2010 ou 2015.
Critérios de Exclusão (CE)	Estudos não disponíveis em texto completo.

A fonte de busca foi escolhida em função da abrangência de indexação de estudos e porque outras bases de dados retornaram baixo número de estudos nas buscas preliminares à essa revisão. O intervalo de cinco anos foi determinado em função da análise da variação do conceito de sistemas legados ao longo do tempo

Foram encontrados no período de 1995 até 2015 um total de 7.600 artigos, os quais foram lidos e selecionados através dos critérios de inclusão e exclusão. O número de artigos resultantes após a aplicação do CI (b) é exibido na Tabela 2.

Tabela 2. Número de artigos antes e depois dos CI e CE

Ano	Antes dos CI e CE	Depois dos CE e CI
1995	88	6
2000	268	17
2005	360	24
2010	445	22
2015	388	31
Total	**1549**	**100**

Um total de 125 definições de sistemas legados foram extraídas manualmente e armazenadas a para posterior análise individual de cada definição. Foram considerados os seguintes tópicos: (a) as definições de sistemas legados; (b) a autoria do artigo; (c) o endereço do repositório; (d) o ano de publicação. A soma das definições supera a quantidade de artigos selecionados porque alguns estudos apresentaram mais de uma definição distinta para sistemas legados.

Utilizou-se a técnica de análise de conteúdo para o tratamento das definições

extraídas dos estudos. Para [Moraes 1999] a análise de conteúdo tem como objetivo "descrever e interpretar o conteúdo de toda classe de documentos e textos obtidos através de uma pesquisa.". O processo de análise de conteúdo possui cinco etapas: preparação das informações; unitarização do conteúdo; categorização das unidades; descrição das categorias e interpretação dos resultados.

Na etapa de unitarização do conteúdo cada definição foi analisada detalhadamente em dois momentos, individualmente e em grupo, com o objetivo de extrair a unidade de análise, neste artigo denominada de característica. Como resultado desse processo, as principais características de cada uma das definições e respectivas frequências foram extraídas e armazenadas. Ao final, das 125 definições de sistemas legados analisadas, geraram-se 62 unidades de análise ou características. Optou-se por utilizar somente as definições que apresentavam frequência maior ou igual a 2, de forma que neste trabalho apresenta-se as 31 características principais.

Logo após iniciou-se a fase de categorização, onde as unidades com características semelhantes foram agrupadas em categorias aqui denominadas de Aspectos. Por exemplo, a característica "Desenvolvido há muito tempo" foi agrupada com a característica "Desenvolvido há mais de 20 anos", formando o Aspecto de Tempo. O resultado deste agrupamento gerou um total de sete categorias identificadas como: "Aspectos de Estrutura de Software", "Aspectos de Manutenção", "Aspectos de Recursos Humanos", "Aspectos de Tecnologia Obsoleta", "Aspectos de Tempo", "Aspectos Financeiros" e "Aspectos Organizacionais". A seguir, na Tabela 3 os seus significados são brevemente explicados.

Tabela 3. Descrição dos Aspectos

Aspecto	Descrição
Estrutura de Software	Engloba características como alta complexidade, ausência de documentação e considerável número de linhas de código, o que torna a arquitetura inflexível a modificações e desatualizada para competir com equivalentes modernos.
Manutenção	Considera características relacionadas à necessidade de extensivas manutenções ao longo de seu ciclo de vida, à lentidão e custo das manutenções.
Recursos Humanos	Refere-se a características relacionadas a inexistência de mão-de-obra especializada ou indisponibilidade dos desenvolvedores originais dos sistemas, o que inviabiliza modificações no software.

Tecnologia obsoleta	Características de sistemas desenvolvidos para hardware antigo e linguagens de programação que não são mais utilizadas. O termo tecnologia inclui hardware, software e técnicas utilizadas para a construção de tais sistemas.
Tempo	Características relacionadas ao período do tempo em que o sistema foi desenvolvido, num passado remoto ou próximo.
Financeiro	Características relacionadas ao valor econômico investido pela organização ao longo dos anos.
Organizacionais	Refere-se a características que indicam a importância do sistema para a organização.

3. Resultados e Discussão

A Figura 1 exibe as características mais presentes nas definições de sistemas legados classificadas por Aspectos e com suas respectivas frequências absolutas, cujo total é 324.

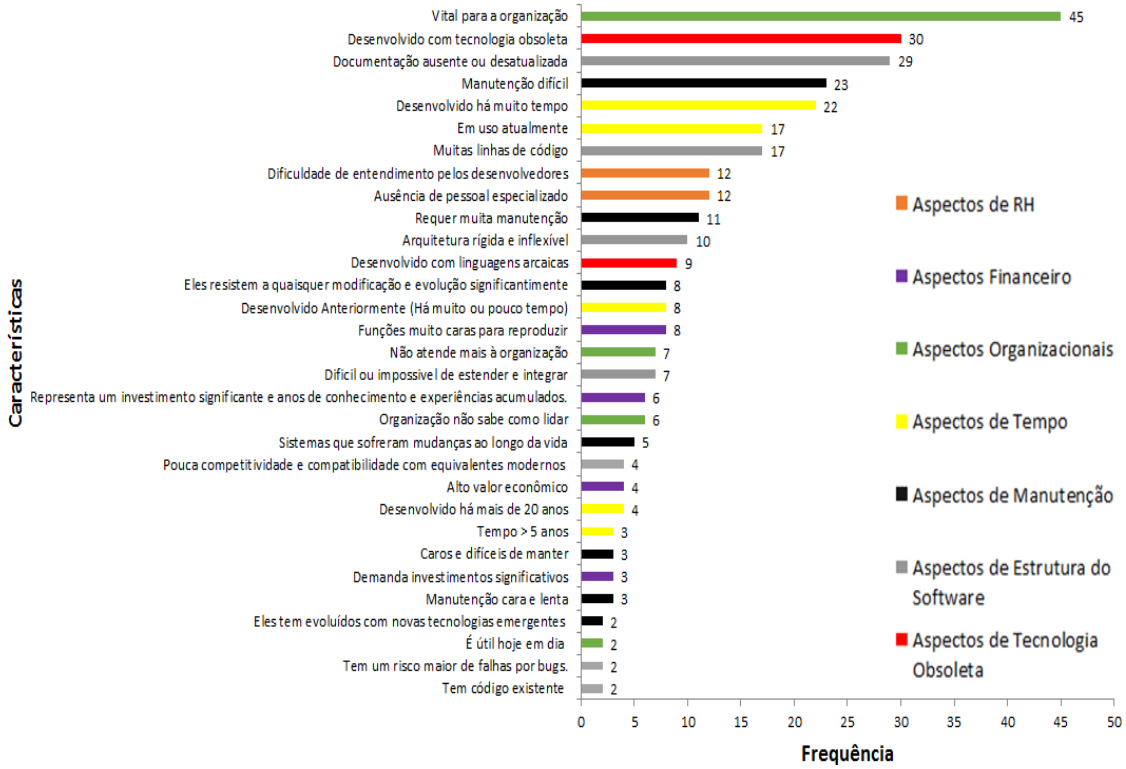
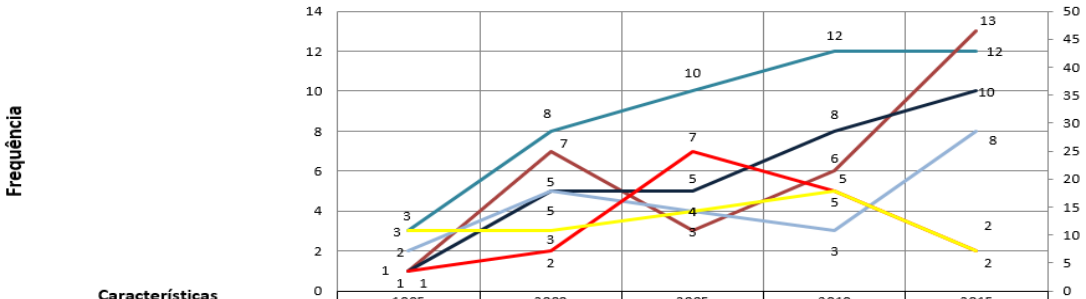


Figura 1. Frequência Total de Características de Sistemas Legados

Observa-se que a característica com maior frequência relativa é “Vital para organização” com 13%, uma característica não tecnológica que evidencia o papel fundamental desse tipo de sistema para as organizações. Na sequência, aparecem as características “Desenvolvido com tecnologia obsoleta” com 9%, “Documentação ausente ou desatualizada” com 8,9%, “Manutenção difícil” com 7% e “Desenvolvido há muito tempo” com 6,7%, às quais já são amplamente conhecidas e imediatamente relacionadas à ideia de sistema legado.

As características presentes nas definições de sistemas legados sofreram variações ao longo do período pesquisado. A evolução da frequência das características encontradas simultaneamente em todos os anos estudados é visualizada na Figura 2. A partir de uma visão global observa-se essas características aparecem em todos os anos do período pesquisado já pode ser considerado um indicador significativo da importância das mesmas. Destaca-se ainda que a frequência de todas as características aumenta a cada ano do período pois também aumenta o número de artigos publicados com essa temática (Tabela 2).



Características	1995	2000	2005	2010	2015
Vital para a organização	3	8	10	12	12
Desenvolvido com tecnologias obsoletas	1	7	3	6	13
Documentação ausente ou desatualizada	1	5	5	8	10
Desenvolvido há muito tempo	2	5	4	3	8
Em uso atualmente	1	2	7	5	2
Muitas linhas de código	3	3	4	5	2
TOTAL	11	30	33	39	47

Figura 2. Evolução de características no intervalo dos anos estudados.

Em uma perspectiva local, a característica ‘Vital para a organização’ foi a mais frequente e manteve uma evolução crescente nos quatro primeiros anos pesquisados, o que demonstra uma conscientização crescente da importância desse tipo nas organizações. Já a característica ‘Desenvolvido com tecnologia obsoleta’ apareceu de forma inconstante nas definições ao longo dos anos, sendo citada apenas uma vez em 1995 e treze vezes em 2015, sendo, portanto, a característica mais citada no último período pesquisado. Esse aumento significativo no último ano deve estar relacionado à própria evolução das tecnologias de informação, que faz com que novos recursos de hardware e software se tornem disponíveis, tornando as tecnologias anteriores

rapidamente obsoletas. Outra característica com evolução considerável foi 'Documentação ausente ou desatualizada' que em 1995 apareceu em somente uma definição, passando para dez vezes em 2015. Aqui o motivo pode estar relacionado ao aumento na demanda de manutenção de sistemas legados, pois é senso comum que muitos sistemas, principalmente os desenvolvidos no passado, carecem de documentação.

Como mencionado na metodologia, a fase de categorização agrupou as características em categorias denominadas Aspectos. Sendo assim, visualiza-se na Figura 3 os Aspectos com suas respectivas frequências em ordem decrescente.

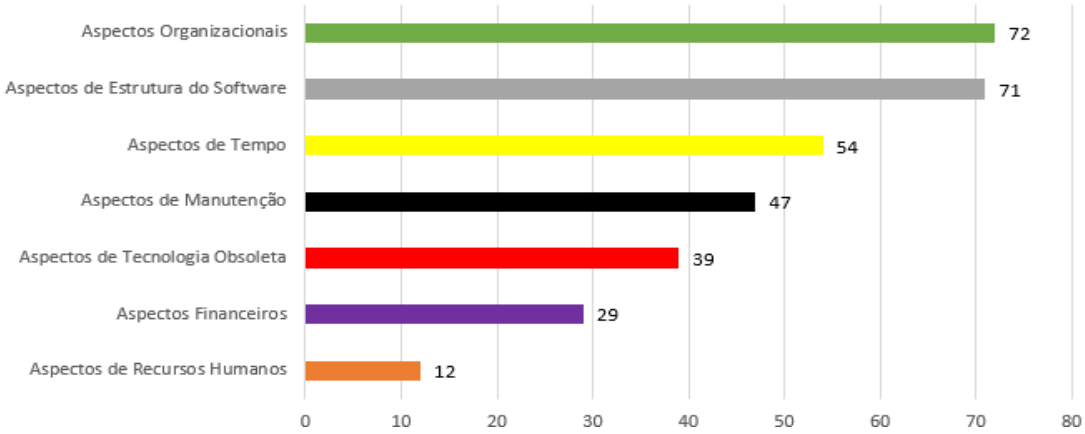


Figura 3. Agrupamento de características em Aspectos

Ao analisar a Figura 3, nota-se a elevada frequência dos Aspectos Organizacionais e dos Aspectos de Estrutura do Software. Essa situação parece refletir que tanto aspectos técnicos quanto aspectos sociais e organizacionais estão envolvidos no escopo de sistemas legados.

4. Ameaças à validade

Neste estudo é considerada uma ameaça à validade a interpretação das características encontradas nas definições de sistemas legados, que pode conduzir a equívocos na contabilização das frequências das características e à atribuição equivocada da característica a um aspecto.

5. Conclusão

O mapeamento de um conjunto consolidado de características de sistemas legados é o propósito principal desta pesquisa. O processo de busca sistemática resultou em 100 artigos, a partir dos quais e, através da análise de conteúdo, foram identificadas 31 características diferentes, que por fim foram agrupadas em 7 aspectos.

A identificação revelou um conjunto considerável de características, com

destaque, em termos de frequência absoluta, para a característica 'Vital para a organização'. Ao analisar as demais características com frequência maior que 20 é possível propor que um sistema legado seja 'Um sistema vital para organização, desenvolvido com tecnologia obsoleta, com documentação ausente ou desatualizada, de difícil manutenção e desenvolvido há muito tempo'.

Conclui-se que a utilização da abordagem de busca sistemática em conjunto com a técnica de análise de conteúdo mostrou-se relevante para a obtenção do conjunto final de características e categorias que representam o conceito de sistema legado do ponto de vista acadêmico ao longo de 20 anos e podem contribuir para facilitar a identificação de tais sistemas nas organizações e, com isso, permitir que a melhor decisão em relação a esses sistemas seja tomada.

Como trabalhos futuros, pretende-se realizar a busca sistemática em todos os anos, com o propósito de tornar os resultados mais robustos e, a partir do conjunto final de características e aspectos, propor a construção de uma taxonomia de características de sistemas legados com o objetivo de organizar o conhecimento acerca deste tópico.

Referências

Bennett, K. (1995). Legacy systems: Coping with success. IEEE Software, 12(1):19–23.

Juric, M. B., Rozman, I., and Hericko, M. (2000). Performance comparison of corba and RMI. Information and Software Technology, 42(13):915–933.

Kitchenham, B. (2004). Procedures for performing systematic reviews. Technical report, Keele University, Keele, UK, Software Engineering Group. Moraes, R. (1999). Análise de conteúdo. Revista Educação, 22(37):7–32.

Ning, J., Engberts, A. P., and Kozaczynski, W. V. (1994). Automated support for legacy code understanding. Communications of the ACM, 37(5):50–57.

O'Byrne, P. and Wu, B. (2000). Lace frameworks and technique - identifying the legacy status of an information system from the perspectives of its causes and effects. In Proceedings International Symposium on Principles of Software Evolution.

Ramsom, J., Sommerville, I., and Warren, I. (1998). A method for assessing legacy systems for evolution. Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering, 42(13):915–933.

Sommerville, I. (2011). Engenharia de Software. Pearson Brasil, 9th edition.

Umar, A. (1997). Application (re)engineering: Building Web-based Applications and Dealing with Legacies. Prentice Hall, 1th edition.

DevOps adoption in Junior Enterprise: an experience report of software development

Peterson Rodrigues[1], Juliano R. Macedo[1], Pedro H. França[1]
Luiz P. Franz[1], João Pablo S. da Silva[1], Jean F. P. Cheiran[1]

[1]Federal University of Pampa (UNIPAMPA)
Caixa Postal 118 – 97500-970 – Alegrete – RS – Brazil

{petersonlrr,juliano.rmacedo,peu06,luizpaulofranz}@gmail.com

{joaosilva,jeancheiran}@unipampa.edu.br

Abstract. *Adopting new approaches to increase software development success rate in junior enterprise context is a constant challenge for this kind of company. In this paper, we report the experience on adoption of DevOps foundations to integrate continuous delivery in software development process. This adoption had three main moments: company diagnosis of maturity, application of tools to reach DevOps maturity level, and analysis of benefits of integration through one project in the company. Results indicate better transparency in project status through interactive communication provided by unifying development and operation areas, and faster software delivery.*

1. Introduction

In the last decade, the manner in which software is delivered has drastically evolved. While corporations like Amazon, Google, Netflix and Facebook provide new features for their clients transparently, many times a day and without shutdown their services, other companies need months to update systems. The key to success in features delivery is the way in which company runs the development cycle [Duvall 2012].

Evolution of software development cycle provided by agile methodologies, and Lean Software Development above them all, has culminated in 'micro delivery' concept. These deliveries are product backlog items split into small activities that are developed and placed in a Quality Assurance (QA) pipeline until they are approved and delivered to final user [de Lange et al. 2016].

DevOps culture puts together Development (Dev), Operations (Ops) and Quality Assurance (QA), and it focuses in collaboration and integration of all company divisions, from developers to stakeholders [West and Groll 2017]. Moreover, Junior Enterprises (JE) present a leading role to disseminate and update knowledge acquired in Brazilian universities. Taking the importance of DevOps culture propagation into consideration, JE allow undergraduate students to learn practices and cultures of modern software industry, and among them is DevOps [Bogo et al. 2014].

The Junior Enterprise (JE) presented in this report is composed of a multidisciplinary team of Software Engineering and Computer Science undergraduate students. Like many other JE, our enterprise is affected by a high turnover of staff, impairing directly our capability to deliver software. Regarding these difficulties, adoption of DevOps

culture could improve maturity of development cycle, software quality and knowledge dissemination among the team.

This paper has goal to report the experience of adoption of DevOps concepts in a JE placed at Alegrete campus of Universidade Federal do Pampa (Unipampa) in Brazil. To reach this goal, we identify JE maturity level, we propose and apply a DevOps workflow with automation tools to support software development process, and we collect JE team perceptions about the impact of DevOps on our project.

Text is organized as follows: at section 2, we present main concepts of DevOps movement; at section 3, we describe our JE in which our project was run; at section 4, we present diagnosis of JE maturity and its analysis; at section 5, we report integration of DevOps movement on JE; at section 6, we discuss lessons of DevOps adoption on our project; and at section 7, we present our findings, conclusions and learnings.

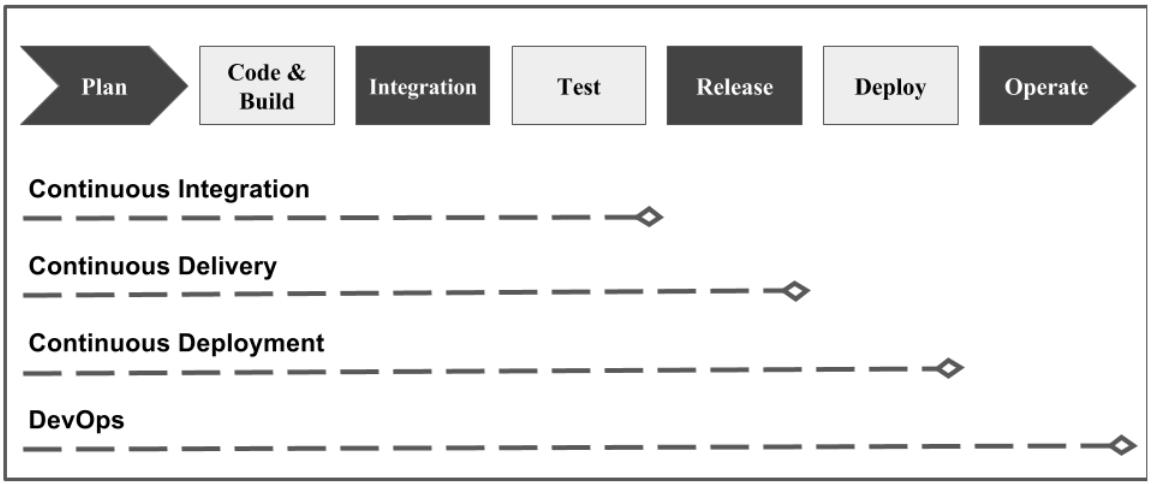
2. DevOps Movement

Software project usually present clear division of teams into roles and responsibilities. Developers create and evolve software products by adding value to business, while operations team is responsible for keeping production environment where product is deployed and consumed by clients [Sato 2014].

In this context, there is a conflict of interest between development and operations, since the first introduces new features and consequent change and the second value stability of offered services by avoiding unavailability and technical problems. This divergence affects release timing by impairing agile principles and by compromising product scheduling [Hüttermann 2012].

Agile movement describes a set of principles for software development in which requirements and solutions evolve through a collaborative effort of auto-organized, multifunctional teams [Beck et al. 2001]. DevOps movement appears as a natural evolution of agile movement and it aims to mitigate conflicts, approximate responsibilities and improve communication of Dev, Ops and QA [Hüttermann 2012, Sato 2014]. At Figure 1, we present an overview of DevOps workflow followed by detailed description.

Figure 1. Collaboration Diagram. Adapted of [Rathod and Surve 2015]



Continuous Integration (CI): It is automation of code **build** and test after every change. It is supported by tools integrated to code repository that instantly report failures on build and test tasks. In doing so, the main goal of CI is to provide a rapid response to developers by alerting when a bug is introduced on source code [Virmani 2015, Rathod and Surve 2015].

Continuous Delivery (CDel): it is partial automation of generation of new system releases. It covers CI phase, and it also includes submission of changes approved by automated testing to a QA team for review. It usually means that application deploy is manually done shortly after QA approval. This approach is commonly applied to safety critical, money critical or information security systems which depend on high availability [Humble and Farley 2010].

Continuous Deployment (CDep): unlike CDel, CDep brings complete automation to system release and deploy. After adding new features and after running automated tests of CI, a new release is automatically sent to deploy system and it is directly deployed at production environment. This strategy is used by most of Software as a Service (Saas) applications like social networks, email servers, etc [Claps et al. 2015].

DevOps: it is complete integration and automation of previous approaches, aiming the ability to develop and deliver software continuously through a transparent, repeatable process. This process promotes full collaboration of Dev, Ops and QA teams in order to allow continuous delivery and infrastructure automation which easily respond to software evolution. DevOps movement is largely used on industry, and it recently has reached academy [West and Groll 2017].

De Lange et al. [de Lange et al. 2016] performed an experiment at RWTH Aachen University that aims to teach DevOps movement to undergraduate students. They investigate what is the technical preparation level of students to perform functions on DevOps-compatible start-up companies after learning about DevOps. Authors also highlight that the experiment allowed to students to reduce the gap between university learning and industry practices.

Perera et al. [Perera et al. 2016] present a study performed on Sri Lanka companies that has found correlations between quality, business adaptation needs and agility in DevOps adoption. In order to do this, they performed interviews with Information Technology (IT) professionals who knew DevOps movement.

Both aforementioned studies do not cover JE peculiarities and this kind of material is deeply scarce on scientific databases, so experience reports on this environment are pioneer and necessary.

3. Our Junior Enterprise (JE)

Ideiah Soluções em Software JE was founded in 2012. The company consists of Software Engineering and Computer Science undergraduate students, and it currently has seven members on staff and one adviser professor.

Since 2014, our JE has adopted agile approaches for flexibility and standardization in product development. So, a Scrum team [Schwaber et al. 2016] composed of five developers, one Product Owner and one Scrum Master continuously perform Extreme Programming (XP) practices [Beck 2000] along software development.

Employing daily Scrum (or daily meeting) and other techniques, we noticed deficiency in used tools for software coding, testing and delivering and we also observed a centralized responsibility on tools that had caused delivery delays. Particularly, we had not promoted homogeneous autonomy on tools to avoid the bottleneck caused by a largely concentrated infrastructure.

Moreover, tools were not mutually integrated, resulting in inefficient communication between development team and infrastructure management team. Since developers seek to deliver software as soon as possible and infrastructure team needs to keep production environment without issues, it is obligatory to reduce this gap.

These problems resulted in many delivery delays and therefore improvement of JE processes is necessary. In order to address problems with tools and responsibilities, we perform a brief literature review on strategies and instruments to identify JE maturity level. Due to this search, DevOps movement emerged as an approach to mitigate issues and to collaborate in supporting agile philosophy of our company. Results of diagnosis of JE maturity is presented at next section.

4. Diagnosis of Maturity

Diagnosis of maturity seek to identify practices and tools adopted by JE and their integration. With this in mind, we can recognize what is missing to effectively implant DevOps movement. Once DevOps has evolved from agile movement, JE already meets the requirement to embrace agile culture.

Figure 1 was used to guide diagnosis process and to implant DevOps concepts integrally. We performed intensive, direct observations [de Andrade Marconi and Lakatos 2010] during an entire development cycle, mapping practices and tools. At the end of the cycle, we compiled information and compared it to DevOps principles and workflow.

Continuous Integration (CI): our JE uses version control and remote distributed repository as integration resources. Tools for test writing and automation are also used, but there was not integration to repository or other tools. In addition to that, there was not policies and support on test reuse and there was not support on test rerunning by coverage analysis tools.

Continuous Delivery (CDel) and Continuous Deployment (CDep): there was no software delivery support tool. As a result, we were heavily overloaded with manual software delivery to QA and with manual software deployment into production environment.

DevOps: JE did not applied a clear, repeatable process of software delivery. It was necessary to manually deploy software and to manually adapt infrastructure after each release to support software evolution. Also, communication between development, operations and QA teams was not efficient.

Since agile culture was met and collaboration of JE divisions seems fulfilled, the adoption of new development workflow focused on environment integration and automation should be sufficient to support DevOps and to mitigate aforementioned problems.

5. Implanting DevOps

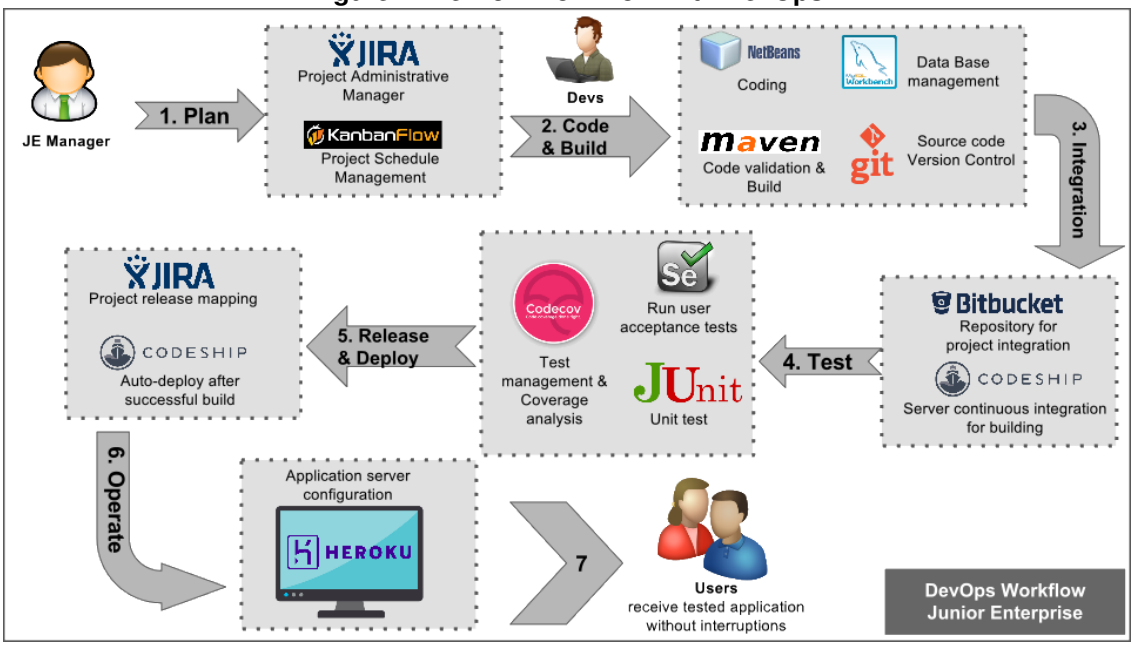
Selection of tools to support integration and automation for a JE commonly has a serious constraint. Because the usual JE educational, nonprofit profile and because the lack of institutional funding, we need to search low cost – or completely free – tools. Even though there is eventual incomes from external sources, JE business model does not assure continuous support to maintain non-free software. Besides that, it is also necessary to choose "easy to learn" tools in order to minimize adaptation problems caused by a high turnover of staff.

JE members from outside the project in which DevOps would be implanted searched tools that fit our context, verified their limitations and costs, and performed pilot tests on another JE software development project.

5.1. Tools identification and selection

Through literature review focused on needs highlighted at section 4, we select and test tools to support software development process. At Figure 2, we draw the new JE workflow with chosen tools[1] and the phase they are applied.

Figure 2. New JE WorkFlow with DevOps



Tools at Figure 2 interact each other in favor of complete integration and automation – from coding to deployment.

We perform the entire project planning (stage 1) with JIRA and KanbanFlow by providing software scope, scheduling milestones and defining production strategies. NetBeans, MySQL Workbench, Maven and GIT cover development and change control

[1]Tools available at `http://jira.atlassian.com` — `www.kanbanflow.com` — `www.netbeans.org` — `http://maven.apache.org` — `www.git-scm.com` — `www.mysqlworkbench.org` — `www.bitbucket.org` — `www.codeship.com` — `www.junit.org` — `www.seleniumhq.org` — `www.codecov.io` — `www.heroku.com`

(stage 2). New or changed artifacts are sent to Bitbucket repository, and it allows CodeShip to try to build the project (stage 3). CodeShip itself runs test scripts (using JUnit and Selenium in our project) while building, and CodeCov runs tests again to keep updated the coverage analysis and other statistics (stage 4). If building is succeeded, CodeShip does an auto-deploy to application server Heroku and JIRA update project status when the new release is detected at repository (stage 5 and stage 6). At the end of this process, users receive a new version of software without service interruption (stage 7).

5.2. Adoption of new workflow

The new workflow implantation was done in two moments: (1) pilot implantation with a small team and (2) full project implantation with the whole team.

In order to test the new workflow, we ran a pilot study during a small increment in a stable project. So, we could identify adaptations in development team practices and particularities of brand-new tools. Workflow implantation time was short, since the team uses branches to organize development. As result, new tools tutorials and workflow overview document were created to facilitate understanding of all.

Furthermore, DevOps workflow was completely integrated into a project for developing an Enterprise Resource Planning (ERP) software and it involved the whole team. Tutorials in fact made implantation easier even for a small, agile team that embrace the change. Detailed information about implantation in production environment is showed at section 6.

6. Lessons Learned

The main project of this report is running for four months using DevOps principles, and it was already running for twelve months when DevOps adoption started. Since DevOps implantation, we performed regular diagnostic evaluation sessions with the team to collect feedback about the new workflow. Besides discussing lessons of DevOps in our JE business environment, we analyzed test coverage statistics to investigate collateral benefits of new tools.

Opinions gathered in diagnostic evaluation suggest positive lessons and team satisfaction from DevOps adoption. Not to mention the overall perception of more effective software delivery, we point out the following main learned lessons:

Agility in identifying bugs: bugs in the project are quicker identified and addressed, since tools run tests more frequently and they allow transparency along the entire workflow to the team. Moreover, new tools allow to monitor real-time project status for developers and managers.

Issues reduction: full test automation prevents developers to skip test phases and assure that failures to be detected earlier that production.

Better communication between Development and Infrastructure teams: even in our agile context, there was a natural conflict between Dev and Infra. When responsibilities are decentralized, communication intensifies and conflict decreases.

Continuous deployment: automated release and deployment increase available time to perform acceptance testing. So, development team and infrastructure team contribute directly to QA.

Learning new tools: there is a motivational impact on JE team, because the very educational nature of a JE environment. Since there is no incomes for its members, learning is a key value to keep staff and to develop professional abilities.

Quicker delivery: as a result of eliminating the gap between end of coding and software delivery, each software increment is delivered quicker.

Improving business value: spreading responsibilities and intensifying communication foster team cohesion to reach a goal, increasing business value.

Code coverage and acceptance testing information reinforce benefits observed on team opinion. There is a significant increase in code coverage assessed through CodeCov before new workflow started (3% coverage) and after four months on new workflow (68% coverage). 3% initial coverage represented only tests on critical parts of the software, so 68% current coverage represents a meaningful increase, considering the complexity of reported project (an ERP software).

Software acceptance rate also increased after DevOps. Before new workflow started, one in four releases had been accepted by product owner according requirements and by team according quality criteria. After new workflow, three in four releases was accepted, and the rejection of the first release was possibly caused by new technological challenges and stricter quality criteria.

As a major lesson, we notice the importance of an organization open to change and the increase of competitive factor through adoption of new technologies and techniques even if it demands technical effort and cultural changes by team members.

7. Final Considerations

We presented in this paper an experience report describing DevOps movement adoption in a software development Junior Enterprise (JE). Still, we presented main DevOps features to locate our JE agile practices into DevOps workflow.

Results indicate DevOps is characterized not only by behavioral features native of agile approaches but also by productivity increase through automated tools. As subject, we present a new software development workflow with stages and tools that promotes deployment automation and communication improvements.

Our main findings reinforce benefits of agile approaches integrated to DevOps culture. So, we has achieved (1) quicker failure detection caused by bugs and by infrastructure issues, (2) more visibility to project status, (3) better communication between development team and infrastructure team, (4) continuous deployment, and (5) quicker delivery.

An important limitation to this paper is project confidentiality clauses that prevents detailed data presentation in order to support statements and findings. Nonetheless, we believe this report solidly contributes to DevOps movement literature by presenting new case study in a very particular kind of company (Junior Enterprise) and by detailing workflow information.

Future work includes a controlled experiment in our JE with explicit authorization to use project artifacts, data and statistics, so we could reevaluate the new workflow applicability.

References

[Beck 2000] Beck, K. (2000). *Extreme programming explained: embrace change*. addison-wesley professional.

[Beck et al. 2001] Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). Manifesto for agile software development.

[Bogo et al. 2014] Bogo, A. M., Henning, E., Schmitt, A. C., and Marco, R. G. D. (2014). The effectiveness of junior companies from the viewpoint of engineering students at a brazilian university. In *2014 IEEE, EDUCON*, pages 745–750. IEEE.

[Claps et al. 2015] Claps, G. G., Svensson, R. B., and Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way. *Information and Software technology*, 57:21–31.

[de Andrade Marconi and Lakatos 2010] de Andrade Marconi, M. and Lakatos, E. M. (2010). *Fundamentos de Metodologia Científica*. Atlas, São Paulo.

[de Lange et al. 2016] de Lange, P., Nicolaescu, P., Klamma, R., and Koren, I. (2016). Devopsuse for rapid training of agile practices within undergraduate and startup communities. In *ECTEL*, pages 570–574. Springer.

[Duvall 2012] Duvall, P. (2012). Breaking down barriers and reducing cycle times with devops and continuous delivery. Online. Acessado em Julho–2017.

[Humble and Farley 2010] Humble, J. and Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. Pearson Education.

[Hüttermann 2012] Hüttermann, M. (2012). Beginning devops for developers. *DevOps for Developers*, pages 3–13.

[Perera et al. 2016] Perera, P., Bandara, M., and Perera, I. (2016). Evaluating the impact of devops practice in sri lankan software development organizations. In *Advances in ICT for Emerging Regions (ICTer)*, pages 281–287. IEEE.

[Rathod and Surve 2015] Rathod, N. and Surve, A. (2015). Test orchestration a framework for continuous integration and continuous deployment. In *2015 International Conference on Pervasive Computing (ICPC)*, pages 1–5.

[Sato 2014] Sato, D. (2014). *DevOps na prática: entrega de software confiável e automatizada*. Editora Casa do Código.

[Schwaber et al. 2016] Schwaber, K., Sutherland, J., and Beedle, M. (2016). The definitive guide to scrum: the rules of the game. *Scrum Guide*.

[Virmani 2015] Virmani, M. (2015). Understanding devops & bridging the gap from continuous integration to continuous delivery. In *Innovative Computing Technology (INTECH), 2015 Fifth International Conference on*, pages 78–82. IEEE.

[West and Groll 2017] West, D. and Groll, J. (2017). The convergence of scrum and devops. Online, Scrum.org and DevOps Institute. Acessado em Setembro–2017.

Resolução de Problemas no Curso de Engenharia de Software: Uma Experiência Envolvendo Extensão e Ensino

Jonnathan Riquelmo Lopes[1], Gabriela Córdova Medeiros[1], Dienefer Fialho[1], Andréa Sabedra Bordin[1]

[1]Universidade Federal do Pampa (UNIPAMPA) - Campus Alegrete - Curso de Bacharelado em Engenharia de Software

97.546-550 - Alegrete - RS - Brasil

{jonnathan.riquelmo, gabiceemi, dieneferfialho}@gmail.com, andreabordin@unipampa.edu.br

Abstract. *In the Problem Based Learning (PBL) methodology, learning is guided by the solving of a problem. Identifying and using real problems in academic settings can be challenging. The university extension provides, through a dialogical relationship with the community, the identification of diverse demands. Many of them have the potential to be solved in curriculum components of Software Engineering. This article presents an experience report on solving a problem, identified through an extension program, in a curricular component conducted through PBL. An evaluation instrument was developed with the students to evaluate aspects of the choice of problems coming from the community and their results are discussed.*

Resumo*. Na metodologia de Aprendizagem Baseada em Problemas (ABP) a aprendizagem é guiada pela resolução de um problema. A identificação e uso de problemas reais em ambientes acadêmicos pode ser um desafio. A extensão universitária propicia, através de uma relação dialógica com a comunidade, a identificação de demandas diversas. Muitas delas têm potencial para serem solucionadas em componentes curriculares de Engenharia de Software. Este artigo apresenta um relato de experiência de resolução de um problema, identificado através de um programa de extensão, em um componente curricular conduzido através da ABP. Foi elaborado um instrumento de avaliação junto aos discentes para avaliar aspectos da escolha de problemas oriundos da comunidade e seus resultados são discutidos.*

1. Introdução

As metodologias tradicionais de educação no ensino superior em sua grande maioria estão centradas na aquisição de conhecimentos passados pelos professores nas salas de aula, onde os alunos assumem uma postura mais passiva. Nesses casos, o incentivo a habilidades necessárias em um ambiente real de trabalho, relacionadas ao trabalho em equipe, proatividade e tomada de decisão, acabam não sendo tão bem exploradas. Contudo, há uma crescente preocupação em tornar o ambiente acadêmico mais próximo a esse ambiente real e o uso de metodologias ativas, tal como a Aprendizagem Baseada em Problemas (ABP) vêm sendo utilizadas com este propósito. [Oliveira *et al.*, 2013]

Na ABP, a aprendizagem deve ser direcionada por um problema. Ou seja, um problema de fim aberto, que não comporta uma única solução correta, deve preceder à teoria, atuando como foco da aprendizagem, e promover a integração dos conceitos e habilidades necessárias para sua solução [Barrows, 2001]. Esta é a principal característica que distingue a ABP de outros processos de ensino-aprendizagem.

O curso de Engenharia de Software da Universidade Federal do Pampa (UNIPAMPA) possui componentes curriculares chamados de Resolução de Problemas (RP), que são voltados ao aprendizado baseado em problemas. Nesses componentes curriculares os alunos são divididos em equipes e buscam a solução do problema proposto utilizando os conceitos de Engenharia de Software vistos até aquele ponto do curso. Cada equipe de trabalho tem um professor tutor que acompanha suas atividades, entretanto, são os próprios estudantes que tomam as decisões durante o desenvolvimento da solução.

Os problemas propostos pelos professores, em sua grande maioria, simulam situações do mundo real onde um sistema de software é necessário, ou seja, não são demandas reais. Entretanto essa simulação, apesar de possibilitar a aprendizagem, acaba não propiciando aos alunos uma experiência real de resolução de problemas através do desenvolvimento de software, que envolva por exemplo, o contato com cliente e *stakeholders* diversos. Há um grande desafio em trazer problemas oriundos de demandas da sociedade para os componentes curriculares de resolução de problemas.

Em 2011 houve um projeto conjunto entre os cursos de Engenharia de Software e Enfermagem da UNIPAMPA. Este projeto criou oportunidades de aprendizado para acadêmicos de ambas as áreas. Enquanto os estudantes do curso de Engenharia de Software em Alegrete realizaram o desenvolvimento de uma solução para um problema real, os alunos da área de saúde do curso de Enfermagem na cidade de Uruguaiana obtiveram um produto de software apropriado para as atividades do Laboratório de Práticas de seu curso [Billa, 2012]. No entanto, a demanda se originou de uma necessidade interna da universidade e, mesmo que a solução final tenha propiciado uma integração inédita entre os campi, não atingiu a comunidade externa em razão do contexto do problema.

Neste viés, a extensão universitária, cuja diretriz fundamental é a relação dialógica com a comunidade, pode ser uma importante aliada na identificação de problemas reais que possam ser resolvidos dentro do ambiente acadêmico. O programa de extensão Programa C da UNIPAMPA é uma iniciativa nessa direção, pois objetiva criar uma cultura local de resolução de problemas com o apoio de tecnologias computacionais dando vazão a demandas locais pelo desenvolvimento de software [Mello, 2016].

O objetivo deste artigo é relatar e avaliar a experiência da resolução de um problema real identificado pelo Programa C, utilizando a ABP dentro do componente curricular de Resolução de Problemas V. A técnica utilizada para levantar os dados e realizar a avaliação da experiência foi a de pesquisa do tipo levantamento (*survey*).

O artigo está organizado da seguinte forma, a Seção 2 apresenta os principais fundamentos sobre a ABP. A seguir, a Seção 3 discute o emprego da extensão universitária. A Seção 4 apresenta detalhes da disciplina de Resolução de Problemas V, sua divisão em grupos, o problema real que foi alvo das atividades e a atuação dos tutores frente aos grupos. Os retornos da turma de 2017 são apresentados na Seção 5, acompanhados de uma análise

sobre as respostas dos acadêmicos. Por fim, há a conclusão deste artigo na Seção 6 acompanhada das referências utilizadas.

2. ABP na Engenharia de Software

A ABP é uma metodologia focada no aluno, o que permite uma aprendizagem ativa e uma melhor compreensão e retenção do conhecimento. Ela instiga o conhecimento através de um processo de investigação e construção de soluções dirigidas pela proposição de ideias.

De acordo com [Oliveira *et al.,* 2013], os engenheiros de software encontram muitos problemas de adaptação no mercado devido a variação de escopo e prazos de projetos de software. Uma das grandes vantagens da ABP é o aprimoramento do conhecimento de um conteúdo, ao mesmo tempo em que promove o desenvolvimento da comunicação, pensamento crítico, colaboração e habilidades de aprendizagem autodirigidas que ajudarão o aluno durante o processo de aprendizagem e sua vida profissional.

Na UNIPAMPA, o curso de Engenharia de Software foi implantado em 2010 com uma organização curricular que adota a ABP de forma integrada ao currículo através de seis componentes curriculares chamados Resolução de Problemas (RP). Esses componentes visam integrar diferentes temas da ES em uma situação-problema, aproximando assim os acadêmicos da realidade profissional de um engenheiro de software.

De acordo com o Projeto Político Pedagógico (PPC) do curso (2015), o objetivo central dos componentes de RP é dar uma visão específica de diversas áreas da ES. Estes componentes possuem foco em conceitos de programação, modelagem, projeto, verificação, validação e testes, visam também introduzir os discentes a conceitos de metodologias ágeis, e ao gerenciamento de projetos.

Segundo [Silva *et al.,* 2017], em RP V os alunos têm a missão de instanciar e executar processos de software de acordo com os objetivos do projeto. Com a finalidade de produzir software de maneira iterativa e incremental, são adotadas abordagens ágeis. Desde a primeira oferta do componente de RP V tem sido adotado o método Scrum acompanhado de práticas XP.

3. Extensão Universitária

O ensino, a pesquisa e a extensão formam o tripé de apoio do processo de ensino-aprendizagem das universidades brasileiras. Segundo o artigo 1º da Resolução Nº 104 da UNIPAMPA "A extensão é um processo educativo, cultural e científico que articula, amplia, desenvolve e realimenta o ensino e a pesquisa e viabiliza a relação transformadora entre comunidade universitária e comunidade externa, possibilitando a produção e a troca de conhecimentos entre as duas instâncias sociais" (UNIPAMPA, p.1, 2015).

Ainda de acordo com a Resolução Nº 104, no artigo 3º as ações de extensão "[...] devem refletir sua interface com o ensino e a pesquisa e contribuir para a formação acadêmica ao promover a articulação entre aspectos teóricos e práticos em todas as áreas do conhecimento". No Curso de Engenharia de Software da UNIPAMPA existem iniciativas de interface entre extensão e ensino que ocorreram de forma bidirecional. Há situações onde uma solução criada dentro de um componente curricular tornou-se posteriormente um projeto de extensão e onde um programa de extensão trouxe problemas passíveis de solução em um componente curricular.

Um exemplo de projeto de extensão criado a partir de um componente curricular é o Sistema de Classificados [Bordin, 2016]. Este projeto de extensão originou-se a partir do desenvolvimento de um sistema de classificados no componente curricular de Resolução de Problemas VI (2013/02). A partir do entendimento de que tal sistema seria útil tanto para comunidade acadêmica, como para comunidade externa, foi proposto que o mesmo fosse evoluído em um projeto de extensão. O sistema, denominado de ClassiPampa, permite anunciar e consultar produtos e serviços.

A experiência abordada neste artigo deu-se de outra forma, com o programa de extensão Programa C identificando um problema que poderia ser resolvido computacionalmente. Este programa é organizado em ciclos anuais divididos em quatro etapas análogas aos processos de software. A primeira etapa denominada *Gera!* corresponde a etapa de engenharia de requisitos. Esta etapa foi a responsável pela identificação do problema proposto no relato deste artigo. Uma das ações executadas foi promover o contato com a comunidade externa, realizando um levantamento de problemas no município de Alegrete. Entre os resultados identificou-se diversas demandas que poderiam ser resolvidas pela comunidade acadêmica, entre elas a que foi levada e trabalhada no componente curricular RP V 2017/01.

4. Metodologia

A metodologia utilizada no artigo aborda primeiramente a escolha do problema, onde são apresentados os argumentos que validam a proposta que foi aplicada no componente curricular RP V. A seguir expõe-se a organização do componente curricular e, por fim, se descreve a composição do instrumento aplicado na avaliação da experiência.

4.1. Escolha do Problema

A proposta do problema identificado originou-se através de entrevistas com quatro gestores municipais e mais de quatrocentos membros da comunidade. Mapearam-se dados que apontaram para diversos problemas da cidade e sua administração. O problema selecionado foi apontado pelo diretor do setor de Tecnologia da Informação (TI) da Secretaria de Administração e consistia na criação de um aplicativo para dispositivos móveis que facilitasse o acompanhamento de processos abertos no setor de protocolo da Prefeitura Municipal de Alegrete.

Esta proposta se mostrou compatível com a organização do componente de RP V. Além de ser passível de solução computacional, também foi considerado um problema que poderia ser desenvolvido dentro do período de duração do componente curricular, e o escopo inicial estava de acordo com as características de um projeto que suportaria ser resolvido através de uma metodologia ágil.

4.2. Organização do Componente Curricular

A organização de RP V foi planejada pelos tutores responsáveis pela aplicação do componente. Após a apresentação geral do problema, houve a divisão do total em oito equipes de três membros cada, e uma com quatro integrantes. Neste processo de formação das equipes, foi oportunizado a escolha individual aos alunos. Os grupos receberam identificações que partiam de 1 até 9, onde um tutor ficou responsável pela avaliação de grupos ímpares e o outro dos pares, alternando-se a cada entrega prevista.

O problema foi resolvido através da utilização de um método ágil, conforme o previsto pela ementa de RP V. A metodologia ágil preconiza um conjunto de valores, tais como indivíduos e interações entre eles mais que processos e ferramentas, software em funcionamento mais que documentação abrangente, colaboração com o cliente mais que negociação de contratos e responder a mudanças mais que seguir um plano. A partir destes valores são derivados uma série de princípios, que são a base para o desenvolvimento ágil.

O *Scrum*, que é uma metodologia ágil para gerenciamento foi selecionado como procedimento base para o decorrer da disciplina, onde os projetos são divididos em ciclos chamados de *Sprints*. O *Sprint* representa um período de tempo dentro do qual um conjunto de atividades devem ser executadas [Schwaber, 2004].

Mediante os conceitos do *Scrum*, o componente foi dividido em quatro *Sprints*. Em cada sprint era previsto o desenvolvimento e entrega de um conjunto de requisitos. No início do projeto foram apresentados um conjunto de requisitos que foram sendo evoluídos. Os dois tutores da disciplina assumiram diferentes papéis previstos na metodologia *Scrum*. Enquanto um foi o *Product Owner,* responsável por definir e priorizar as funcionalidades a serem desenvolvidas, além de ser uma ponte entre o cliente e a equipe, o outro foi o *Scrum Master* que atua como um facilitador junto à equipe de desenvolvimento, garantindo que a metodologia seja seguida.

No transcorrer de todo o projeto houveram encontros com os clientes, na figura dos funcionários da Prefeitura Municipal de Alegrete. Foram realizadas três reuniões nas dependências da Universidade visando a apresentação do andamento do projeto e recebimento de *feedback*. Uma reunião foi no Centro Administrativo da Prefeitura, onde houve uma apresentação para diversos membros da administração pública.

O produto final entregue consistia em realizar consulta a um processo da prefeitura mediante entrada de dados de identificação, passando por uma validação e busca do processo em um *web service* e oferecia a possibilidade de compartilhamento dos andamentos do processo em diferentes plataformas.

4.3. Instrumento de Avaliação da Experiência

A técnica usada para realizar a pesquisa foi do tipo levantamento (*survey*). Pesquisas do tipo levantamento caracterizam-se pela interrogação direta das pessoas cujo comportamento se deseja conhecer. Basicamente procede-se a solicitação de informações a um grupo de pessoas, para, em seguida, mediante análise quantitativa, discutir-se conclusões sobre o contexto e questionamentos abordados [Bertucci, 2014].

Neste viés um instrumento de avaliação foi elaborado e aplicado aos discentes que cursaram o componente curricular de RP V. O instrumento é composto por quatro afirmações que têm como objetivo avaliar aspectos relacionados com a escolha do problema oriundo da comunidade e o papel da extensão universitária no contexto acadêmico.

As afirmações foram dispostas em escala Likert, com itens de um a cinco que averiguam o quanto um indivíduo concorda ou discorda de uma declaração específica [McLeod, 2008]. O presente instrumento foi enviado para os vinte e oito discentes que finalizaram o componente curricular, obtendo o retorno de vinte e cinco avaliações.

5. Resultados e Discussão

A partir do instrumento de avaliação foi gerado um gráfico que reúne os resultados obtidos, organizando-os de acordo com a porcentagem de cada afirmação dentro da escala Likert, como é mostrado na Figura 1.

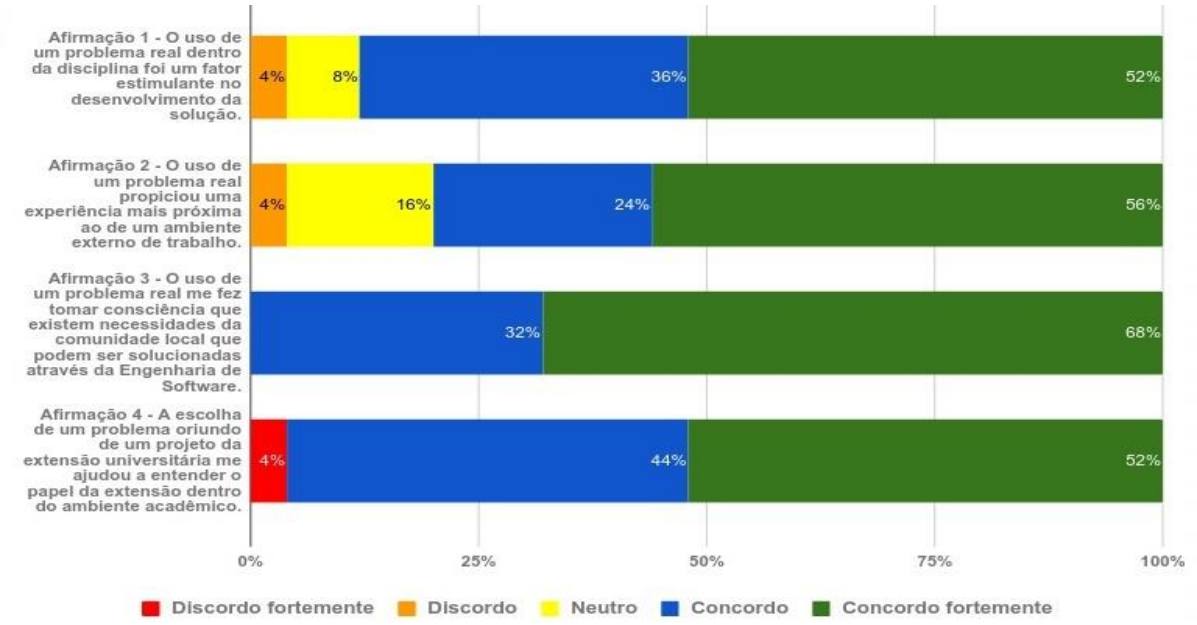


Figura 1 - Resultados do instrumento de avaliação da experiência de RP V.

Os problemas propostos precisam ser desafiantes para que os acadêmicos questionem-se e sintam-se estimulados a propor uma solução, para atingir as metas previstas com a utilização da ABP [Santos e Angelo 2009]. Isso pode ser observado na Figura 1, onde a Afirmação 1, que diz respeito ao fato de um problema real aplicado na disciplina ser um fator estimulante para desenvolvimento da solução, teve um alto percentual de concordância (88%).

A Afirmação 2 possui uma concordância de 80%, indicando que para os discentes o uso de um problema real proporciona uma experiência mais próxima à de um ambiente externo de trabalho. Uma hipótese para esta concordância alta é que esta experiência proporcionou que as habilidades associadas ao uso da ABP, que preparam para o mercado de trabalho, fossem melhor exercitadas pois os stakeholders envolvidos no processo eram reais e, consequentemente, o compromisso assumido com os mesmos e a perspectiva de ter a solução usada por usuários finais (comunidade) era maior.

É importante salientar o índice de alunos que não concordaram com as duas primeiras afirmações. Um discente discordou da Afirmação 1 e 2 totalizando 4% em ambas. Em outras respostas dois se declararam neutros na Afirmação 1 e outros quatro na Afirmação 2, totalizando 8% e 16% de opiniões neutras, respectivamente. Apesar de serem percentuais baixos, é importante e também desafiador levantar hipóteses para estas impressões. Entre essas hipóteses pode-se citar a falta de motivação ou comprometimento de alguns discentes que cursaram o componente curricular e a não identificação de diferenças entre essa experiência e as vivenciadas nos componentes cursados anteriormente.

Observa-se na Afirmação 3 o maior índice de concordância de todo o instrumento, alcançando 100%. Possivelmente o motivo para esse nível máximo de conformidade em

relação às necessidades da comunidade esteja associado ao fato que não ter ocorrido iniciativas semelhantes nos componentes de RP cursados anteriormente. Sendo assim, nesta experiência os discentes se depararam pela primeira vez com essa possibilidade, podendo constatar que soluções desse tipo podem causar impacto social na região onde estão inseridos.

Por fim, na Afirmação 4 houve um índice de concordância de 96% em relação ao entendimento dos discentes acerca do papel da extensão no ambiente acadêmico. Observa-se que alunos de graduação podem, muitas vezes, se aproximar da conclusão do curso sem ter contato direto com as iniciativas de extensão. Alguns prováveis motivos a serem elencados são a falta de interesse dos próprios discentes e a pouca divulgação de ações da extensão no meio acadêmico. Este último pode ser justificado, do ponto de vista dos docentes, pelo aspecto desafiador de realizar a associação da extensão com o ensino nas ofertas curriculares. Sendo assim, é compreensível que a percepção dos discentes, em relação à extensão universitária na esfera acadêmica seja fraca e que essa experiência tenha proporcionado um entendimento melhor acerca da mesma.

6. Conclusão

A experiência apresentada neste artigo mostra a resolução de um problema identificado em um projeto de extensão e desenvolvido em um componente curricular através da metodologia ABP. A avaliação da experiência mostrou que os discentes sentiram-se mais estimulados a resolver um problema real e que essa experiência permitiu a vivência de situações semelhantes às que ocorrem no mercado de trabalho. Além disso, possibilitou a conscientização do potencial da extensão universitária.

Essa experiência evidencia um elo entre ensino e extensão, que em geral é bastante desafiador em vários cursos e áreas de conhecimento. Entretanto, no Curso de Engenharia de Software da UNIPAMPA, onde aplica-se a metodologia de Aprendizagem Baseada em Problemas de forma integrada ao currículo, esse elo parece ser muito promissor e factível, visto que o condutor da aprendizagem é um problema. A relação dialógica com a comunidade permite tanto que o problema seja proposto pela comunidade como possibilita a Universidade fazer a proposição. Em ambos os casos, acredita-se que os produtos de software gerados tenham um grande potencial de impacto social.

O desenvolvimento de soluções para problemas da sociedade em componentes curriculares traz consigo o desafio da manutenção dos sistemas. Componentes curriculares tem período de duração pré-determinado, logo após a sua conclusão fica inviável manter o compromisso dos discentes na manutenção do sistema. Uma solução para isso é viabilizar a manutenção através de alguma ação de extensão, como no caso do Sistema de Classificados.

Pode-se concluir que o Curso de Engenharia de Software da UNIPAMPA buscou exercer uma função social através da extensão, articulando o ensino com as necessidades da comunidade onde se insere. Desta forma, avalia-se que esta experiência é fruto de uma iniciativa inédita no âmbito dos componentes curriculares de Resolução de Problemas e abre maiores possibilidades para que o curso atue como agente ativo e transformador na realidade social que o cerca.

Referências

Bordin, A. S. (2016) Da Aprendizagem Baseada em Problemas à Extensão Universitária: Caso do sistema de classificados, P.117-133 In: Extensão Universitária: Vivências nas Engenharias e na Computação. Bagé: Ediurcamp.

Barrows, H. (2001) Problem-based Learning (PBL). Stanford University Newsletter on Teaching. Vol.11, No. 1.

Billa, C. Z., Devincenzi, S. da Silva, Welfer, D. (2012) Experiência de ABP aplicado em Engenharia de Software. In: Congreso Internacional PBL - ABP. Cali.

Bertucci, J. L. de Oliveira. (2009) Metodologia Bá□sica Para Elaboração De Trabalhos De Conclusão De Cursos (TCC): Ênfase Na Elaboração De TCC De Pós-Graduação Lato Sensu. Atlas.

McLeod, S. A. (2008) Likert Scale. Disponível: <www.simplypsychology.org/likert-scale.html>. Acesso em setembro, 2017

Mello, A. V. (2016) Programa de Extensão C. Disponível: <https://www10.unipampa.edu.br//portal/resumo.php?projeto_id=7248>. Acesso em setembro, 2017

Oliveira, A. M. C. A., Dos Santos, S. C., Garcia, V. C. (2013) PBL in teaching computing: An overview of the last 15 years. IEEE Frontiers in Education Conference (FIE), Oklahoma City.

Santos, J. A. M., Angelo, M. F. (2009) Análise de Problemas Aplicados em um Estudo Integrado de Programação utilizando PBL. In: Anais do Workshop sobre Educação em Informática WEI.

Schwaber, Ken. (2004) Agile Project Management with Scrum.

Silva, J. P. S., Tolfo, C., Guedes, G. T. A., Kepler, F. N. (2017) Aprendizagem Baseada em Problemas aplicada no Ensino De Abordagens Ágeis, P.93-113 In: Aprendizagem Baseada em Problemas na Engenharia de Software: Relatos de Experiência. Bagé: Ediurcamp.

Universidade Federal do Pampa. (2015) Projeto Político-Pedagógico do Curso.

Universidade Federal do Pampa. (2015) Conselho universitário. Normas para as atividades de extensão: (Resolução Nº 104/2015). Bagé, RS. Disponível: <http://porteiras.r.unipampa.edu.br/portais/proext/files/2015/05/Res.-104_2015-Normas-de-Extens%C3%A3o-e-Cultura.pdf>. Acesso em setembro, 2017

Percepção da Contribuição da AGES para o Aprendizado em Engenharia de Software

Alessandra C. S. Dutra, Azriel Majdenbaum,
Cássio A. W. Trindade, Daniel A. Callegari, Marcelo H. Yamaguti

Faculdade de Informática – Pontifícia Universidade
Católica do Rio Grande do Sul (PUCRS)
Avenida Ipiranga, 6681– Porto Alegre – RS – Brazil

{alessandra.dutra, azriel.majdenbaum,
cassio.trindade, daniel.callegari, marcelo.yamaguti}@pucrs.br

Abstract. *Researchers and practitioners have been looking for solutions to the classic problems of software development since the 1970s using a team-focused approach. In this context, the Experimental Agency of Software Engineering (AGES) is an innovative space within the Pontifical Catholic University of Rio Grande do Sul (PUCRS), which allows students of the Software Engineering course to develop software projects with real clients. This article aims to present a first analysis about the students' perception of the AGES regarding the evolution of their knowledge of the technologies and skills that are developed in the course for the performance of the profession of Software Engineering.*

Resumo. *Pesquisadores e profissionais têm procurado, desde a década de 1970, soluções para os problemas clássicos do desenvolvimento de software utilizando uma abordagem focada no funcionamento das equipes. Neste contexto, a Agência Experimental de Engenharia de Software (AGES) é um espaço inovador, dentro da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), que permite aos alunos do curso de Engenharia de Software desenvolverem projetos de software com clientes reais. Este artigo tem como objetivo apresentar uma primeira análise sobre a percepção dos alunos da AGES a respeito da evolução do seu conhecimento das tecnologias e habilidades que são desenvolvidas no curso para o desempenho da profissão de Engenheiro de Software.*

1. Introdução

Pesquisadores e profissionais têm procurado, desde a década de 1970, soluções para os problemas clássicos do desenvolvimento de software utilizando uma abordagem focada nas pessoas e no funcionamento das equipes. Segundo Gibbs (1994), a qualidade da capacitação em Engenharia de Software pode contribuir para a 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. 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, 2014).

Neste contexto, a Agência Experimental de Engenharia de Software (AGES) é um espaço inovador, dentro da Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS), que permite aos alunos do curso de Engenharia de Software (ES), implantado em 2015, desenvolverem projetos de software com clientes reais. A possibilidade de os alunos atuarem em diversos papéis - tais como desenvolvedor, testador, analista, arquiteto de software e gerente de projetos - potencializa o desenvolvimento de suas habilidades técnicas e não-técnicas. A Agência também desenvolve atividades de formação, tais como: participação dos alunos em palestras; capacitação em tecnologias (conhecimentos) necessárias para o desenvolvimento dos projetos; preparação para apresentação de palestras ou apoio a capacitação; prática em língua estrangeira, mais especificamente em língua inglesa. A AGES foi projetada para se integrar à matriz curricular do curso de Engenharia de Software. Os alunos devem realizar quatro módulos na agência: Prática na AGES I, II, III e IV, que ocorrem respectivamente no 2º, 4º, 6º e 7º semestres do curso.

Por ser um curso novo, com uma abordagem inovadora, onde a prática vem concomitante com a teoria, faz-se necessário avaliar como vem sendo percebida tal contribuição segundo a percepção dos discentes.

Portanto este artigo tem como objetivo apresentar a importância da AGES na formação dos alunos do novo curso, por meio da análise da sua percepção a respeito da evolução do seu conhecimento das tecnologias e habilidades que são desenvolvidas para o desempenho da profissão de Engenheiro de Software. Os dados deste estudo foram coletados por meio de uma *survey* respondida por uma amostra de 52 dos 69 estudantes atualmente matriculados nas disciplinas de Prática na AGES do curso de ES. Esta é a primeira coleta formal de dados, a fim de evidenciar a evolução dos alunos em relação ao conhecimento das tecnologias e habilidade desenvolvidas nos projetos dentro da AGES. Já cursaram a disciplina de Prática na AGES, 217 alunos desde sua implantação, que ocorreu no segundo semestre de 2015.

2. A Agência Experimental de Engenharia de Software

A Agência Experimental de Engenharia de Software foi concebida para ser um ambiente prático de aprendizagem com finalidade de propiciar aos estudantes o exercício de situações reais de projetos em um ambiente com foco na aprendizagem, viabilizar a integração interdisciplinar dos conteúdos do curso, integrar ensino, pesquisa e extensão, e permitir contato com organizações e empresas sempre que este contato agregar elementos para a formação do perfil do egresso (Yamaguti et al, 2017).

Cada módulo da AGES, é considerado uma disciplina especial com 4 créditos e que envolve 120 horas de atividades sendo 60 horas com um professor orientador, e possui como objetivo desenvolver na prática uma (ou mais) de um conjunto de competências esperadas de um egresso em Engenharia de Software. Em função das competências a serem desenvolvidas, cada módulo possui um conjunto de requisitos, de forma que o aluno, com auxílio do professor orientador, possa relacionar os conteúdos

abordados em aula com o cotidiano de sua vivência nos projetos. O Quadro 1 apresenta as competências e requisitos dos módulos.

Em geral, o aluno desenvolverá uma das competências do módulo. Eventualmente, pode haver a oportunidade para atuação em mais de um papel, desenvolvendo, portanto, mais de uma competência no mesmo módulo. Espera-se que o aluno desenvolva suas competências de forma, inclusive, a desempenhar papéis de liderança e gerência nos projetos, e auxiliando no funcionamento da Agência em si.

Quadro 1: Etapas da realização de um módulo da Agência

Módulo	Competências relacionadas	Requisitos da matriz curricular
I	• Programação • Teste unitário • Depuração	• Programação Orientada a Objetos (co-requisito) • Introdução à Engenharia de Software (requisito especial)
II	• Projeto de Banco de Dados • Análise de requisitos • Desenvolvimento	• Modelagem e Projeto de Banco de Dados (requisito especial) • Programação Orientada a Objetos (pré-requisito)
III	• Testes e verificação • Projeto de software • Arquitetura de software	• Verificação e Validação II (requisito especial) • Projeto e Arquitetura de Software (pré-requisito)
IV	• Gerenciamento de projetos • Aprofundamento de outras competências desenvolvidas no curso • Portfólio de conclusão de curso	• Gerenciamento de Projeto de Software (pré-requisito)

Como podemos perceber, cada módulo está estruturado de forma que, sistematicamente, competências de diferente complexidade sejam desenvolvidas fomentando a formação do futuro profissional.

3. Metodologia

Este trabalho caracteriza-se como uma pesquisa de campo que utilizou uma *survey*, desenvolvida no lócus de atuação dos entrevistados. Como instrumento de coleta de dados foi utilizado um questionário com questões semiestruturadas. A unidade de análise desta pesquisa foi o conjunto de alunos matriculados nas disciplinas AGES de 2017, 2º semestre. Antes de rodar a pesquisa, procedeu-se à realização de um pré-

teste com três alunos que possuíam características similares à amostra desejada. O pré-teste teve como objetivo medir o tempo médio necessário para o preenchimento, bem como a clareza das questões formuladas. Após os pré-testes, o questionário foi novamente refinado, gerando, assim, a sua versão final.

A *survey* na sua versão final, ficou composta por 78 questões fechadas e foi dividida em três grandes partes. A primeira pretendia avaliar a percepção dos alunos sobre o quanto a experiência na AGES contribuiu para o conhecimento deles sobre tecnologias, habilidades técnicas e não-técnicas. Utilizamos para cada pergunta uma escala baseada na Likert (Likert, 1932), com 6 itens, a saber: Não Contribuiu, Contribuiu Muito Pouco, Pouco, Levemente, Muito e Completamente. Estas perguntas, portanto, foram respondidas apenas pelos alunos do grupo B.

A segunda grande parte pretendia avaliar quanto os alunos percebem sobre o atual conhecimento deles sobre os mesmos itens. Neste caso, a escala utilizada foi: Sem conhecimento, Baixo, Razoável, Bom, Ótimo, Excelente. Estas perguntas foram respondidas por todos os alunos.

Por fim, a terceira parte procurou medir a experiência (em meses) sobre o mesmo conjunto de itens. A medida foi fornecida livremente pelos respondentes, e foi interpretada como um número inteiro. Estas perguntas também foram respondidas por todos os participantes.

Após a realização da execução da *survey*, procedeu-se a análise dos dados, apresentada na próxima seção.

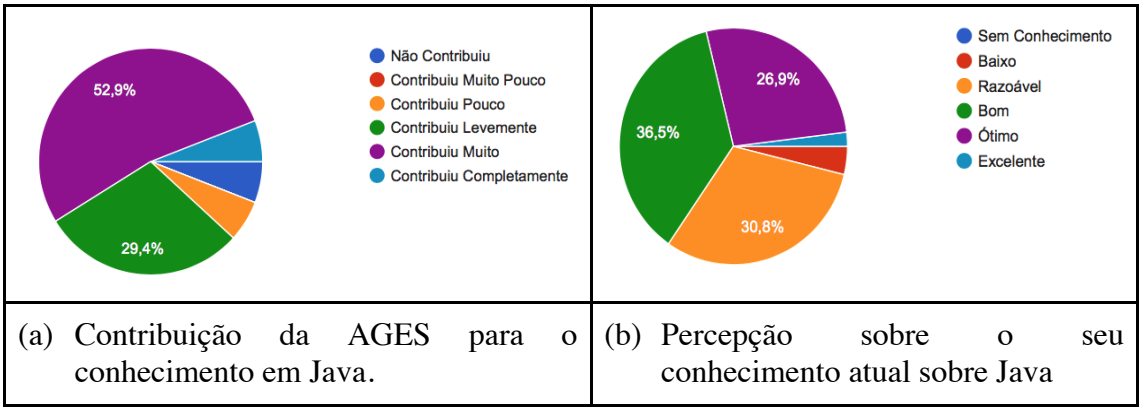
4. Análise dos Dados

Para fins de análise, dividimos a amostra de 52 participantes em dois grupos, a saber: Grupo A, representando os alunos que estão no primeiro nível da AGES; e Grupo B, representando os alunos que estão atualmente cursando as disciplinas AGES II ou AGES III.

Dado que o curso de ES ainda está em implantação, ainda não temos alunos cursando o último nível da AGES. Desta forma, o Grupo A possui 35 representantes, enquanto que o Grupo B possui 17. Do total de 52 respondentes, 35 (67,3%) são alunos do primeiro nível da AGES. Os demais respondentes são alunos que estão cursando os níveis II e III da Agência.

A título de informação, algumas das tecnologias envolvidas na pesquisa são: Java, JavaScript, Node.js, NoSQL, tecnologias Web e de testes de software. A escolha das tecnologias decorre do fato de serem trabalhadas ao longo do curso e também serem amplamente utilizadas no mercado de trabalho. Foram avaliadas também habilidades técnicas tais como Programação, Projeto de Banco de Dados, Arquitetura de Software e Gerenciamento de projetos. Em relação às habilidades não-técnicas, alguns exemplos de medidas são: Liderança, Proatividade e Resiliência.

Para exemplificar, podemos traçar um contraponto avaliando as respostas de uma tecnologia amplamente utilizada (como Java) versus uma tecnologia cujo contato tipicamente só ocorreu na Agência (como Node.js).



(a) Contribuição da AGES para o conhecimento em Java.

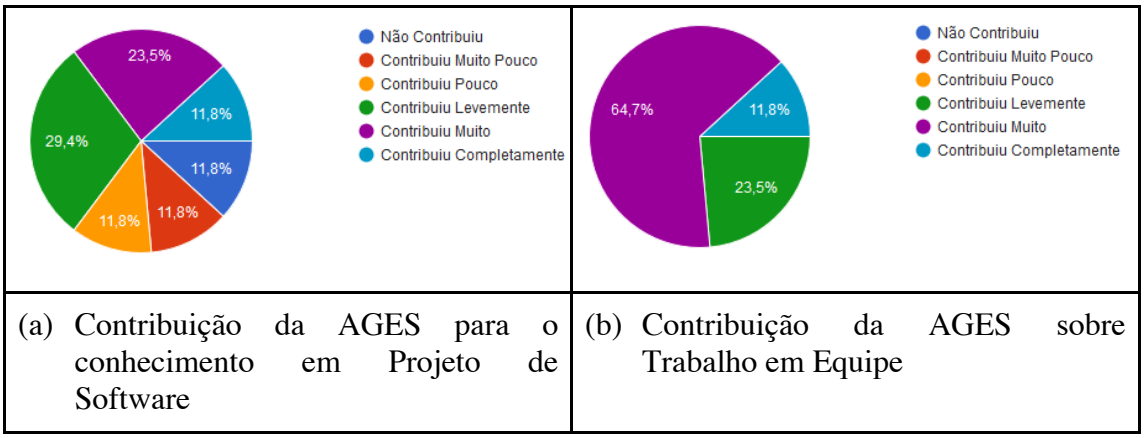
(b) Percepção sobre o seu conhecimento atual sobre Java

Figura 1 - Conhecimento em Java do aluno X Contribuição da AGES no aprendizado

No caso de Java, conforme apresentando na Figura 1, aproximadamente 52,9 % considera que a agência contribuiu expressivamente na formação do seu conhecimento em tal tecnologia. A AGES consolidou com a experiência prática os conhecimentos adquiridos em sala de aula. No entanto 29,4% consideraram que a vivência na AGES contribuiu levemente para seu conhecimento. E, por fim, somente 11,8 % afirmam que não houve contribuição. Uma possível explicação para esse resultado é que os indivíduos já tenham trazido na sua bagagem conhecimento suficiente para o grau de complexidade demandado pelos projetos em que trabalharam. Outra possibilidade que realmente para alguns indivíduos a experiência na agência tenha agregado pouco.

No caso de Node.js, quase 80% dos respondentes afirmaram que nunca tiveram contato com a tecnologia (Dutra, 2017). Os alunos que foram envolvidos em projetos que utilizam a tecnologia se beneficiaram de tal experiência.

A figura 2 ilustra dois casos interessantes da análise a respeito da contribuição da AGES com (a) conhecimento em Projeto de Software e (b) Trabalho em Equipe: no primeiro caso, há uma distribuição mais homogênea das respostas, enquanto que no segundo caso, percebe-se claramente que a Agência proporcionou uma grande contribuição à habilidade de trabalhar em equipes.



(a) Contribuição da AGES para o conhecimento em Projeto de Software

(b) Contribuição da AGES sobre Trabalho em Equipe

Figura 2. Dois exemplos de resultados da coleta de dados.

Os resultados da análise trazem uma conclusão muito positiva e que, de certa forma, já era esperada: as respostas para a contribuição da AGES para habilidades

técnicas são mais homogêneas em termos de distribuição na escala, quando comparadas às respostas para as perguntas que mediam a contribuição sobre as habilidades não-técnicas. Neste segundo caso, pode-se evidenciar a importância da AGES para a complementação da formação técnica dos alunos com o exercício contínuo das habilidades não-técnicas, em especial Comunicação e Proatividade.

A análise completa dos dados, bem como uma descrição detalhada sobre a metodologia e das questões da pesquisa, podem ser encontradas no relatório técnico 85 (Dutra, 2017).

5. Considerações Finais

A Engenharia de Software evoluiu, e hoje a indústria tem muito interesse no desenvolvimento de software de alto desempenho, tendo como objetivo entregar cada vez mais rápidas, com mais qualidade e com mais eficiência (Nearshore, 2012). Neste sentindo, este artigo apresentou a importância da AGES na formação dos alunos, por meio da análise da sua percepção a respeito da evolução do seu conhecimento das tecnologias e habilidades que são desenvolvidas para o desempenho da profissão de Engenheiro de Software, validando os dados por meio dos resultados da *survey*.

Como trabalhos futuros, os dados coletados na *survey* para este artigo servirão como linha base de pesquisas futuras para o aprimoramento do ensino aprendizado a ser utilizado dentro da AGES.

Referências

ACM/IEEE. (2014) Computer Science Curriculum, Guidelines for Undergraduate Degree Programs in Software Engineering.

DUTRA, Alessandra C. S.; Majdenbaum, Azriel; Trindade, Cássio A. W., Callegari, Daniel A., Yamaguti, Marcelo (2017) Relatório Técnico TR85/2017. Disponível em: http://conteudo.pucrs.br/wp-content/uploads/sites/19/2017/10/Relatorio-Tecnico-RT85.pdf

GIBBS, W. (1994) “Software's chronic crisis”. Scientific American, pp. 86–95.

LIKERT, R. (1932) “A technique for the measurement of attitudes”. Archives of Psychology, n. 140, pp. 44-53.

NEARSHORE AMERICAS (2012). “Collaborate. Innovate. Accelerate. Creating successful software requires a new model of development and a new kind of development team”. Capturado em: http://www.nxtbook.com/nxtbooks/nextcoast/nearshore_americas/#/1.

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.

YAMAGUTI, M., Oliveira, F., Trindade, C. A. W., Dutra, A.C.S. (2017) "AGES – an interdisciplinary space based on projects for Software Engineering learning", SBES'17, September 20–22, 2017, Fortaleza, CE, Brazil.

Experiência de uso do *eXtreme Programming* (XP) em disciplina de Engenharia de Software

Lis Ângela De Bortoli[1]

[1]Instituto Federal do Rio Grande do Sul (IFRS) – Campus Sertão
Rodovia RS 135, Km 25 – Distrito Eng. Luiz Englert – 99.170-000 – Sertão – RS – Brazil

lis.debortoli@sertao.ifrs.edu.br

Abstract. *This article describes an experience of using the agile process eXtreme Programming (XP) with the third level classes, of Software Engineering course in the Systems Analysis and Development program, IFRS, campus Sertão. The dynamic was the development of an application, chosen by the class, using the practices and values of XP. The team was set, the classroom environment was reinvented and students played the roles, along with the teacher. The activity was applied to three classes and the results show that the students could better understand the method, in addition to experience in practice, an actual working environment.*

Resumo. *Este artigo descreve uma experiência de uso do processo ágil eXtreme Programming (XP) com turmas do terceiro semestre, da disciplina de Engenharia de Software, do curso de Análise e Desenvolvimento de Sistemas do IFRS, campus Sertão. A dinâmica consistiu no desenvolvimento de uma aplicação, escolhida pela turma utilizando as práticas e os valores do XP. A equipe foi definida, o ambiente da sala de aula foi reinventado e os alunos desempenharam os papéis, junto com a professora. A atividade foi aplicada em três turmas e os resultados demonstram que os alunos conseguiram entender melhor o método, além de vivenciar, na prática, um ambiente real de trabalho.*

1. Introdução

Com o intuito de melhorar o desenvolvimento de software, em 2001, foi criada, por um grupo experiente de desenvolvedores, a Aliança Ágil. Os membros desta aliança elaboraram o manifesto ágil (Beck, 2001), um documento com princípios a serem seguidos na criação de processos ágeis.

O primeiro projeto *eXtreme Programming* (XP) iniciou em março de 1996, sendo esta abordagem um dos vários processos ou métodos ágeis, que já provaram ser muito bem-sucedidos na indústria de software, de diferentes tamanhos e em todo o mundo. Criado por Kent Beck, XP é uma evolução da abordagem incremental e enfatiza o desenvolvimento em incrementos, o envolvimento do cliente no processo e a constante melhoria do código (Beck, 2004).

Atualmente, no mundo acadêmico, uma questão que se depara é como preparar estudantes, de forma adequada, para que possam atuar na indústria de software e trabalhar em equipes de forma ágil. Pensando nisso, definiu-se uma dinâmica para ser utilizada nas aulas da disciplina de Engenharia de Software, ministrada no terceiro nível do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas (ADS), do Instituto Federal do Rio Grande do Sul, Campus Sertão. Esta dinâmica baseia-se no

desenvolvimento de uma aplicação utilizando as práticas e valores do método XP, levando para a sala de aula a simulação de um ambiente de trabalho. A dinâmica já foi aplicada com três turmas e cerca de quarenta alunos já participaram da atividade.

A fim de apresentar o trabalho, a seção dois deste artigo aborda o processo ágil XP, seus conceitos, práticas e valores. A seção 3 mostra a ideia da dinâmica, suas características e fases. Na seção 4 os principais resultados e uma breve discussão são explicados. Por fim, demonstra-se as considerações finais e as referências bibliográficas utilizadas.

2. O processo ágil e*X**treme Programming* (XP)

Segundo Teles (2004), o XP é um processo de desenvolvimento ágil que busca assegurar que o cliente receba o máximo de valor a cada dia de trabalho da equipe de desenvolvimento. O processo é baseado em torno de práticas e valores que, em sintonia, asseguram que o cliente receba um software de valor, em curto espaço de tempo. Surgiu, junto com outros processos ágeis, numa tentativa de evitar os problemas frequentes advindos do uso de processos tradicionais ou prescritivos, como atrasos nos cronogramas, extrapolação de custos, insatisfação do cliente e cancelamentos prematuros dos projetos.

O trabalho em equipe é moldado pelos valores fundamentais que são *feedback*, comunicação, simplicidade e coragem. Em conjunto, estes valores enfatizam a necessidade de aprendizado mútuo, compartilhamento, aproximação dos envolvidos, a implementação de apenas aquilo que é suficiente para atender as necessidades e a coragem de acreditar que a utilização das práticas e valores permitem evoluir com segurança e agilidade.

As práticas (Figura 1) são utilizadas para aplicar os valores durante o desenvolvimento de software. Estas práticas devem ser utilizadas em conjunto, pois se complementam, de forma que os pontos fracos de cada uma são superados pelos pontos fortes de outras.

Figura 1. Práticas XP

Fonte: Adaptado de Teles (2004).

Em XP os requisitos são expressos como histórias do usuário que são implementados diretamente como uma série de tarefas, o desenvolvimento é incremental e as entregas são frequentes e em curtos períodos. Mudanças devem ser

encaradas como rotina e não como algo que atrapalha o projeto. O desenvolvimento baseia-se fortemente no trabalho em equipe através de práticas como programação em pares, código coletivo e ritmo sustentável.

A participação do cliente deve ser intensa no projeto, sendo peça fundamental para o sucesso do projeto. Para Pressman (2006), à medida que o trabalho de desenvolvimento prossegue, o cliente pode adicionar histórias, mudar o valor de uma história existente, subdividir histórias e eliminá-las. A equipe XP então reconsidera todas as versões remanescentes e modifica os seus planos adequadamente.

Além do cliente, uma equipe XP é formada por pessoas que exercem papéis: gerente de projetos, *coach*, analista de teste ou testador, redator e desenvolvedor. O gerente de projetos é responsável pelo relacionamento da equipe com o cliente, tem a visão do todo e preocupa-se com aspectos administrativos e burocráticos. O papel do *coach* é assegurar que a equipe respeite e utilize as práticas e valores e o analista de testes é responsável por testar e garantir a qualidade do sistema. A atribuição do redator é manter a documentação do sistema atualizada e o desenvolvedor, por sua vez, analisa, projeta e codifica o sistema, ou seja, é o papel que efetivamente constrói o software.

3. A dinâmica na disciplina de engenharia de software

A engenharia de software é uma disciplina que se preocupa 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, 2013). Atualmente, um dos tópicos abordados em disciplinas de engenharia de software de cursos de graduação é métodos ágeis. Questionou-se então, como abordar o assunto de forma prática e que os alunos pudessem entender como sua aplicação pode ser feita em um ambiente diferente da sala de aula, simulando um ambiente de trabalho.

Pensou-se em uma dinâmica que pudesse aliar a teoria e a prática atráves de um estudo de caso. Sendo assim, iniciou-se com uma explanação sobre o surgimento dos métodos ágeis e sobre os princípios ágeis e após, focou-se no processo XP. A seguir, foi realizada uma pesquisa pelos alunos e uma discussão de aproximadamente dois períodos sobre o processo. Definiu-se então, que os alunos desenvolveriam uma aplicação simples para verificar, na prática o que o XP propõem na teoria. Por serem turmas de um tamanho relativamente pequeno (dez a treze estudantes), a equipe XP foi formada por todos os alunos e pela professora. A atividade foi dividida em três fases principais: planejamento, execução e avaliação.

3.1. Planejamento

No planejamento a primeira etapa foi um *brainstorming* para escolha da aplicação a ser desenvolvida. A seguir, foram distribuídos os papéis, sendo que para cada aluno (ou mais de um em alguns casos) foi designado um papel. Os papeis de gerente de projetos, desenvolvedor, *coach*, cliente, redator e testador foram distribuídos considerando o perfil dos estudantes, ou seja, procurou-se priorizar as habilidades e competências de cada um. A professora exerceu o papel de cliente e como a disciplina se desenvolve em laboratório de informática, definiu-se que a dinâmica teria que ser um outro local, uma sala de aula, respeitando o que o XP propõem sobre o ambiente de trabalho.

Ainda, na fase de planejamento, foram definidas todas as tecnologias a serem utilizadas no desenvolvimento da aplicação, como hardware, banco de dados e linguagem de programação. As tecnologias foram instaladas previamente em notebooks

e todo o material de apoio foi providenciado pela professora: reserva de sala (com projetor, quadro branco, mesas, cadeiras), canetões, fichas para as histórias, *post-its*, celular (para gravar) e máquina fotográfica. Também foi providenciado, pelos alunos, um lanche para que não precisassem sair da sala durante a fase de execução. Todos os padrões de desenvolvimento foram definidos pela equipe como nomes de tabelas, formulários, variáveis, interface com o usuário, etc.

Nesta fase também foram escritas as histórias do usuário (exemplo na Figura 2) e realizada a modelagem do banco de dados no quadro, que foi posteriormente fotografado pelo redator para integrar a documentação do projeto. Cada estudante comprometeu-se em estudar o seu papel como forma de preparação para a fase de execução.

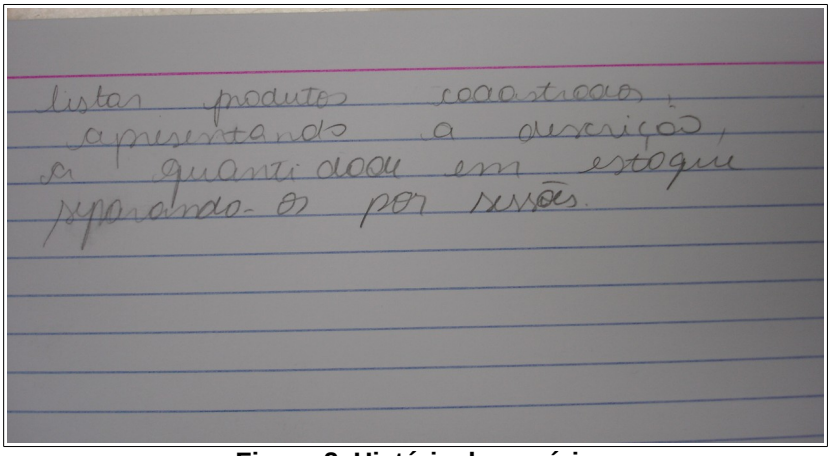


Figura 2. História do usuário

Fonte: Primária.

3.2. Execução

A fase de execução consiste na realização da dinâmica na data marcada. Para isso foi montado um cronograma, visto que a disponibilidade de tempo era de um total de quatro horas. O cronograma contempla as seguintes atividades: organização da sala (30 minutos), *stand up meeting* de planejamento e distribuição das histórias (15 minutos), desenvolvimento das histórias (1 hora e 15 minutos), *stand up meeting* de acompanhamento (15 minutos de reunião para relato de andamento e/ou dificuldades e troca de papeis entre os pares), continuação do desenvolvimento das histórias (1 hora e 15 minutos), *stand up meeting* de encerramento (15 minutos), reorganização da sala (15 minutos).

3.3. Avaliação

Na fase de avaliação, marcada para uma semana após a execução, os alunos responderam um questionário, cujo objetivo era verificar a percepção deles sobre a dinâmica e sobre o XP. O questionário de avaliação foi dividido em duas partes, sendo que na primeira respondiam sobre a simulação do projeto XP: pontos fortes, dificuldades enfrentadas no geral, o que foi possível aplicar do XP, o que poderia melhorar e atribuição de uma nota para a atividade. A segunda parte abordou o papel que cada um exerceu na simulação do projeto: descrição do papel exercido e o que fez durante a simulação do projeto, de que forma contribuiu com o grupo, dificuldades enfrentadas e autoavaliação. Por fim, os alunos poderiam fazer observações que

julgassem pertinentes.

4. Discussão e resultados

A dinâmica já foi aplicada em três turmas diferentes, entre os anos de 2014 e 2016. A primeira turma desenvolveu um controle para a maratona de engenharia de software, outra dinâmica adotada pela professora. A segunda optou por um controle de um supermercado, abordando os módulos de clientes e vendas. Na terceira turma foi desenvolvido um controle para uma loja de calçados.

O ambiente organizado para a dinâmica foi adequado, a forma de disposição das mesas no centro da sala de forma que todos pudessem se enxergar, facilitou o trabalho dos pares de programação. A utilização do quadro facilitou a interação da equipe, onde foram disponibilizados os modelos elaborados, fixados os padrões e o cronograma, demonstrando-se bastante útil para registrar as decisões dos programadores e as informações que surgiram ao longo do desenvolvimento do trabalho. O quadro de histórias concluídas, em andamento e por fazer, baseado no modelo *Kanban* (Aquino, 2016), também foi montado. Os padrões definidos pelos desenvolvedores deram produtividade ao trabalho. A organização do lanche funcionou, cada um parou conforme sua necessidade. A Figura 3 e a Figura 4 apresentam parte do ambiente de uma das dinâmicas.

Figura 3. O ambiente de trabalho e a equipe

Fonte: Primária.

Com relação aos papeis optou-se por deixar os desenvolvedores como testadores também. O redator também pôde exercer outro papel, pois os registros não necessitam ser realizados a todo momento. Os pares interagiram de acordo com o esperado, mas houve certa resistência no momento da inversão de papéis dos pares. É preciso considerar que no método a troca ocorre após um turno ou dia de trabalho. Os desenvolvedores entenderam adequadamente o sistema proposto pelo cliente, que pôde fazer suas atividades normais e ainda atender as dúvidas dos desenvolvedores. A presença do cliente para esclarecer os requisitos do sistema (histórias) foi essencial e os cartões das histórias foram constantemente atualizados com anotações necessárias.

Figura 4. O ambiente de trabalho e a equipe

Fonte: Primária.

Houve momentos que os desenvolvedores utilizaram o projetor para compartilhar "descobertas" com os demais participantes. A maioria dos pares trabalhou em sintonia e houve interação entre os pares, que se ajudaram nas dúvidas. O gerente de projetos circulou bastante orientando a equipe e o *coach* se preocupou com o andamento dos trabalhos, o cumprimento das práticas e valores. O redator acompanhou o gerente de projetos e registrou, com o detalhe necessário, o andamento do trabalho, mostrando que houve sintonia entre eles. As reuniões em pé foram efetivas e produtivas (Figura 5).

Figura 5. *Stand up meeting*

Fonte: Primária.

Os valores de *feedback*, comunicação, simplicidade e coragem foram bem marcantes na dinâmica, os estudantes puderam perceber a importância deles para o bom andamento das atividades. As práticas cliente presente, jogo de planejamento, programação em pares, *stand up meeting*, desenvolvimento guiado pelos testes, código coletivo, código padronizado, *desing* simples, metáfora e integração contínua foram todas aplicadas com sucesso. O *refactoring* foi incentivado durante todo o tempo, buscando o melhoramento do código. Já a prática de ritmo sustentável não ficou tão evidente devido ao curto espaço de tempo da dinâmica. Com relação a prática de *releases* curtos puderam verificar que é viável desenvolver, com produtividade, e entregar algo de valor ao cliente num curto espaço de tempo.

A documentação produzida mostrou-se suficiente e adequada como o processo indica. Foi composta pelas histórias do usuário, pelas fotos do quadro, onde estavam o cronograma, os padrões, o quadro das histórias e o modelo de dados. Além disso, todas as reuniões foram gravadas e também foi possível gerar relatórios disponíveis pela linguagem de programação utilizada.

Nas três aplicações da dinâmica foi possível concluir aproximadamente 80% do que tinha sido planejado, ficando os 20% restantes em andamento.

5. Trabalhos relacionados

Alguns trabalhos do uso de métodos ágeis em sala de aula tem sido propostos na literatura. Um deles é proposto por Gomes Filho et al (2014), que aplica conceitos ágeis na disciplina de Métodos Ágeis para os alunos de graduação do curso de Informática. O objetivo é demonstrar como o uso desses conceitos ajuda a criar um ambiente de aprendizagem estimulante e eficaz.

Gomes Filho et al (2015), também desenvolveu um método baseado na simulação de um ambiente de desenvolvimento de sistemas utilizando diversas práticas ágeis em sala de aula, que culmina com num evento anual denominado Sabadágil.

Ainda, o trabalho de Anslow e Maurer (2015) apresenta uma experiência do desenvolvimento de aplicações web, com base em métodos ágeis, ao longo de 13 semanas, para um cliente real, usando a abordagem “aprender fazendo”.

A diferença do trabalho aqui apresentado é que esta é uma prática desenvolvida durante uma ou duas aulas, de forma que os estudantes possam desenvolver parte de uma aplicação, utilizando os conceitos estudados, simulando o ambiente XP. Além disso, os estudantes participam das decisões do projeto: qual aplicação a ser desenvolvida, escolha dos papeis e das tecnologias utilizadas, a modelagem dos dados e gerência de riscos. Desta forma é possível simular um ambiente real de desenvolvimento e os participantes são ativos na dinâmica.

6. Considerações finais

O espaço de sala de aula propicia a formação de saberes e competências. No entanto, é preciso também, preparar os estudantes para o desempenho profissional. Muito tem sido discutido sobre o distanciamento da academia do mundo do trabalho. Por um lado, a necessidade de possibilitar uma sólida formação teórico-prática, de outro às necessidades do mundo profissional. Como conciliar as duas questões, ainda é um desafio. Cursos superiores de tecnologia, são cursos cuja vocação é atender a demandas específicas do mercado de trabalho. Para isso, é preciso adotar metodologias de ensino que procurem contemplar essa importante lacuna existente.

O objetivo deste artigo foi demonstrar, de forma sucinta, uma experiência adotada na disciplina de Engenharia de Software do curso de ADS, utilizando o processo ágil XP.

Após aplicar a dinâmica em três turmas, pode-se concluir que a prática tem-se mostrado efetiva. A receptividade dos alunos é grande, ficam muito espantados com a produtividade atingida e como é importante o trabalho cooperativo. Além disso, é possível proporcionar um diálogo entre os diversos conteúdos abordados no curso, com destaque para banco de dados, linguagens de programação, engenharia de software e gerência de projetos, enfatizando assim a interdisciplinaridade.

A partir do questionário elaborado foi feita uma avaliação da prática. O instrumento contém questões abertas que permitem extrair dados qualitativos. Os estudantes destacaram a rapidez no desenvolvimento, a boa comunicação proporcionada pelo processo, o *feedback* como forma de resolver os problemas no momento que acontecem, a importância da presença do cliente, a possibilidade de ter orientação do *coach* e do gerente de projetos, a produtividade e o comprometimento nas *stand up meetings*, a definição de padrões para se ter um código padronizado, a eficiência do *refactoring*, a adequação da documentação produzida, através de imagens, relatórios e vídeos.

Alguns problemas também aconteceram, como lentidão da rede, ausência de algum estudante, incompatibilidade do par, falta de motivação de alguns membros da equipe, nível de conhecimento diferente da equipe. Estes problemas foram identificados e tratados pelo gerente de projetos de forma a não atrasar o projeto. Assim, os estudantes aprenderam que nem tudo acontece como o planejado, são os riscos que ocorrem em um ambiente de trabalho, que precisam ser identificados, analisados e mitigados.

Os estudantes manifestaram entusiamo e todos recomendam que esta atividade continue a ser aplicada na disciplina. No entanto, encontra-se em fase de definição um método de avaliação que contemple questões objetivas, de forma a permitir uma avaliação mais precisa da dinâmica.

Como planos futuros pretende-se realizar a dinâmica com outras turmas, buscando o aperfeiçoamento. Também encontra-se em estudo, uma forma de utilizar o processo ágil Scrum nos trabalhos práticos da disciplina.

Referências

ACM/IEEE. Software Engineering Curriculum (2013). “Guidelines for Undergraduate Degree Programsin Software Engineering”.

Anslow, C. Maurer, F. (2015) “An Experience Report at Teaching a Group Based Agile Software Development Project Course”. SIGCSE’15, Kansas City, MO, USA.

Aquino, R. (2013). “Aplicação de alguns princípios do Kanban em Lean TI”. http://www.leanti.com.br/artigos/5/aplicacao-de-alguns-principios-do-kanban-em-lean-ti.aspx. Acesso em Outubro de 2016.

Beck, K. (2004). Programação Extrema (XP) Explicada. Porto Alegre: Bookman.

Beck, K. et al. (2001). “Manifesto for Agile Software Development”, http://agilemanifesto.org/. Acesso em Fevereiro de 2016.

Gomes Filho, A. F. et al. (2014). “Usando métodos ágeis para ensinar métodos ágeis.” In: WORKSHOP BRASILEIRO DE MÉTODOS ÁGEIS, 5., 2014, Florianópolis. Anais. São José dos Campos: INPE, p.1–12.

Gomes Filho, A. F. et al. (2015). “Sabadágil: métodos ágeis de desenvolvimento de software aplicados ao ensino” Revista Brasileira de Informática na Educação, Porto Alegre, v.23, n.3, p.144–163.

Pressman, R. S. (2006). Engenharia de Software. São Paulo: McGraw-Hill.

Teles, V. M. (2004). Extreme Programming. São Paulo: Novatec.

Web-Atlas: Uma Ferramenta Web para Modelagem de Características

Fernando F. Lima[1], Luciano Marchezan[1], Marcelo Schmitt Laser[2], Elder Rodrigues[1], Maicon Bernardino[1]

[1]Universidade Federal do Pampa (UNIPAMPA)
Código Postal 97.546-550 – Alegrete – RS – Brasil

[2]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Código Postal 90.619-900 – Porto Alegre – RS – Brasil

{fernando.fortlima,lucianomarchezan94,marcelo.laser}@gmail.com
elderrodrigues@unipampa.edu.br, bernardino@acm.org

Abstract. *This work presents Web-Atlas, a tool developed to assist the process of teaching-learning feature modeling of Software Product Lines (SPL). Web-Atlas is a stand-alone Web tool that searchc to provides an intuitive interface to the user. The tool provides basic features, such as: create, edit, save, load and validate feature models. With these features and characteristics, we believe that the tool may be used to support the teaching of SPL as well as be applie idn real projects in the industry.*

Resumo. *Este trabalho apresenta a Web-Atlas, uma ferramenta desenvolvida com o objetivo de auxiliar no processo de ensino-aprendizagem de modelagem de características em Linhas de Produtos de Software (LPS). A Web-Atlas é uma ferramenta Web stand-alone, a qual busca proporcionar uma interface intuitiva ao usuário. A ferramenta fornece funcionalidades básicas, tais como: criar, editar, salvar, carregar e validar os modelos de características. Com estas funcionalidades e características, acredita-se que a ferramenta possa ser utilizada no apoio ao ensino de LPS, bem como sua aplicação em projetos reais na indústria.*

1. Introdução

A área de Engenharia de Linhas de Produto de Software (ELPS) [Apel et al. 2013], como todas as outras engenharias, requer o uso de ferramentas para auxiliar no projeto e desenvolvimento de Linha de Produto de Software (LPS) ou na re-engenharia de uma família de sistemas para uma LPS. Estas ferramentas podem ser divididas entre *plugins*, como por exemplo a PlugSPL [Rodrigues et al. 2012] e FeaturePlugin [Antkiewicz and Czarnecki 2004], e sistemas *stand-alone*, tais como Software Product Line Online Tools (S.P.L.O.T.) [Mendonca et al. 2009] e webSPIFF [Bidarra et al. 2001] que são disponibilizadas via Web. Todas as ferramentas apresentam as funcionalidades essenciais para a modelagem de características, como importação e exportação de arquivos, disponibilidade online, e intuitividade na *interface* gráfica para o usuário. Sendo assim, algumas destas ferramentas são complexas ou não apresentam algumas funcionalidades para auxiliar a modelagem de modelos, tais como: representação gráfica ou validação.

Neste trabalho é apresentada a Web-Atlas, uma proposta de ferramenta que desempenha funções no auxilio da modelagem de caraterísticas, tais como: criar, editar, gravar, abrir e validar diagramas de características, disponibilidade online, e representação gráfica e textual do diagrama. A ferramenta Web-Atlas permite a criação de modelos utilizando a notação gráfica chamada Análise de Domínio Orientada a Características (*Feature-Oriented Domain Analysis* - FODA) [Kang et al. 1990].

Este trabalho está organizado em cinco seções. A Seção 2 apresenta os conceitos básicos e características de Linha de Produto de Software (LPS), na definição do que são modelos de característica e os motivos de seu uso, o que são diagramas de características e como são representados, bem como as definições e propriedades da notação FODA. A Seção 3 apresenta ferramentas similares e discussões sobre suas funcionalidades. A Seção 4 expõe os requisitos levantados para o desenvolvimento da ferramenta, incluindo discussões e decisões tomadas, as funcionalidades presentes, e explora um exemplo de uso. Por último, a Seção 5 aponta as considerações finais e trabalhos futuros.

2. Fundamentação Teórica

O conceito de Linha de Produtos de Software (*Software Product Line - SPL*) é baseado no conceito de Linha de Produto comum às Engenharias, ou seja, é um conjunto de softwares que compartilham de características semelhantes, com o objetivo de produzir softwares diferentes que utilizam deste mesmo conjunto de características [SEI 2017]. A LPS é composta por duas características: plataforma e variabilidade [Pohl et al. 2005]. Plataforma é qualquer base tecnológica sobre a qual outras tecnologias ou processos são construídos. Plataforma também pode ser definida como o "núcleo do sistema", contendo as características comuns a todos os sistemas de uma família de software. Plataformas auxiliam no planejamento e projeto dos artefatos que serão usados em dois ou mais produtos. Variabilidade se trata da flexibilidade, ou seja, da habilidade de criar artefatos que serão reusados em uma família de software [Kang et al. 1990]. Em outras palavras, é a habilidade de um sistema de dar suporte a produção de um conjunto de artefatos que possuem pontos diferentes entre eles [Bachmann and Clements 2005].

Um modelo de características documenta as características de uma linha de produto, seus relacionamentos, e é uma abordagem que expressa a variabilidade e semelhanças dentro de uma família de softwares. De acordo com Kang *et al.*, "Características são aspectos visíveis ao usuário ou características do domínio" [Kang et al. 1990]. Os modelos de características são usados no processo de desenvolvimento de LPS, representam todas as possibilidades de combinações entre as características de uma família para o desenvolvimento de produtos de software.

Um diagrama de características é uma notação gráfica para especificar graficamente um modelo de características, podendo ser construídos e apresentados de acordo com diferentes notações [Apel et al. 2013]. Neste trabalho, é adotada a representação em estrutura de árvore, em que os nós representam as características e as arestas são os relacionamentos pai-filhas. A representação de diagramas de características pode ser feita com diversas notações, tais como: a notação FODA [Kang et al. 1990] e *Generative Programming* [Czarnecki et al. 2004]. A notação FODA foi apresentada com o objetivo principal de representar, de forma gráfica, um modelo que permite a visualização de todas as configurações que podem ser produzidas a partir do mesmo conjunto de características.

Esta notação é composta pelas seguintes propriedades: **a)** *Raiz*: é usada para representar o conceito do domínio, ou seja, para definir o conceito genérico. Por meio dela serão especificadas as características e criadas as configurações para cada produto de software dentro da família; **b)** *Características*: são representadas como obrigatórias, opcionais ou alternativas. As obrigatórias são características que estão presentes em qualquer configuração. As opcionais são características que podem ou não ser selecionadas pelo usuário. As alternativas formam um grupo de características, sendo possível a seleção de apenas uma dessas características; **c)** *Grupos Alternativos*: restringem que o usuário escolha apenas uma das características contidas em um grupo; **d)** *Regras de Composição*: representam duas restrições textuais, "requer" e "mutuamente exclusivo com"; **e)** *Razão*: apresenta de forma textual motivos ou justificativas para auxiliar na tomada de decisão.

A Figura 1 ilustra estas características por meio de um exemplo da composição de um carro. Neste exemplo, a raiz, nomeada de *Carro*, representa o conceito do domínio. *Transmissão* e *Potência*, são exemplos de características obrigatórias para o sistema. As características *Manual* e *Automático* são especializações da *Transmissão*; elas representam um grupo alternativo, ou seja, uma e somente uma delas deve ser escolhida. *Ar Condicionado* é apresentado como característica opcional no sistema; esta característica pode ou não ser selecionada.

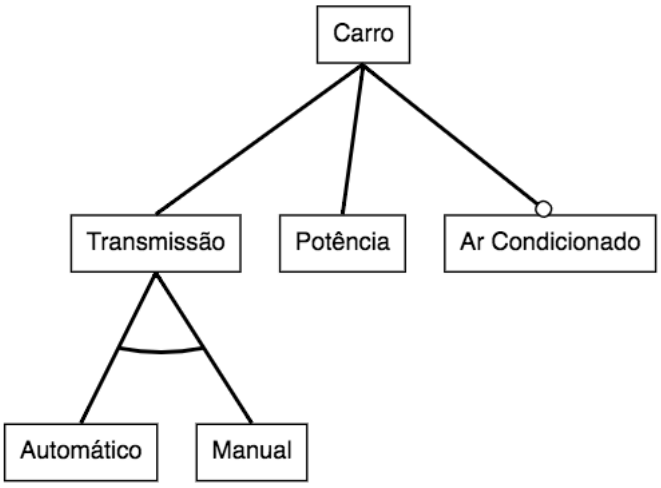


Figura 1. Exemplo da Notação FODA (Adaptado de [Kang et al. 1990])

3. Trabalhos Relacionados

Para o levantamento dos requisitos e funcionalidades requeridos pela ferramenta foram apontadas algumas ferramentas que podem ser usadas para auxiliar na LPS.

O WebSPIFF [Bidarra et al. 2001] é um sistema Web comercial para modelagem colaborativa de modelos de características. O sistema tem uma arquitetura cliente-servidor composto por um sistema de modelagem e um gerenciador de sessões. O servidor fornece a colaboração além de funcionalidades mais avançadas, tais como: gerenciamento de sessões e validação dos modelos. Enquanto que o lado cliente fornece as funcionalidades mais utilizadas, tais como a visualização do modelo e *interface* interativa de especificação.

A FeatureIDE [Kastner et al. 2009] se trata de um *framework open source* que suporta os processos da linha de produto em uma infra-estrutura de ferramenta. O resultado é uma ferramenta que da suporte ao *AHEAD Tool Suite*, tornando mais fácil a adoção de resultados de pesquisas para projetos da academia ou da indústria. O *framework* final dá suporte a múltiplas linguagens e ferramentas.

A FeaturePlugin [Antkiewicz and Czarnecki 2004] foi apresentada como um *plugin* para o Eclipse IDE, que suporta a notação baseada em cardinalidade [Czarnecki et al. 2004]. A *interface* de usuário para a apresentação dos modelos de características permite um meio eficiente na criação e edição de modelos com tamanhos relativamente grandes. Porém, a ferramenta suporta a edição somente via teclado.

A S.P.L.O.T. [Mendonca et al. 2009] é uma ferramenta Web para configuração e pensamento lógico para LPS. O sistema oferece uma *interface* Web, a qual utiliza um grande conjunto de algoritmos para configuração e *logic solvers* relacionado a modelos de características. Além disso, o sistema também mantém um repositório de modelos de características online para compartilhamento de conhecimento.

A PlugSPL [Rodrigues et al. 2012] é um ambiente automatizado que dá suporte a LPS baseadas em *plugins*. O ambiente é dividido em três módulos: Projeto de LPS, Configuração do Produto e Geração do Produto. A proposta se difere do que já existe, pois visa dar suporte a LPS baseadas em *plugins*, podendo importar e exportar modelos de características para outras ferramentas e gerar produtos de maneira eficiente.

As ferramentas citadas apresentaram certas características, tais como: validação dos modelos, de qual forma a ferramenta é disponibilizada, a possibilidade de realizar trabalhos colaborativos, construção e representação de modelos gráficos ou textuais. Sendo assim, a Web-Atlas possui características que correspondem com a maioria das características implementadas pelas outras ferramentas, exceto pelo trabalho colaborativo.

4. Web-Atlas

A Web-Atlas é uma ferramenta que visa auxiliar no aprendizado acadêmico na área de modelagem de características de linhas de produtos de software. Foi desenvolvida visando apresentar uma *interface* intuitiva e ser disponível online. Para seu desenvolvimento, foram levantados alguns requisitos mínimos para o funcionamento da ferramenta (RQ). Estes requisitos são apresentados a seguir: **RQ1.** A ferramenta deve estar disponível para os usuários via Web para facilitar o acesso e não necessitar a instalação de programas e *plugins*; **RQ2.** A ferramenta deve dar suporte aos navegadores: Google Chrome (versão 4.0), FireFox (versão 2.0), e Safari (versão 3.1), incluindo também os navegadores para dispositivos móveis. Este requisito é necessário para alcançar maior número de usuários e evitar que o navegador venha a ser um obstáculo para a utilização da ferramenta; **RQ3.** A ferramenta deve possibilitar que o usuário crie, de forma gráfica, um modelo de características utilizando a notação FODA; **RQ4.** A ferramenta deve possibilitar que o usuário salve localmente o modelo criado de forma textual (formato .json). Salvar de forma textual possibilitará que o modelo seja carregado ou possa ser modificado sem a necessidade de conexão com a Internet. **RQ5.** A ferramenta deve possibilitar que o usuário salve localmente o modelo criado de forma gráfica (formato .png). Salvar de forma gráfica ajuda a visualizar o modelo de forma clara e simples, permitindo o uso da imagem em outros documentos; **RQ6.** A ferramenta deve disponibilizar a opção de

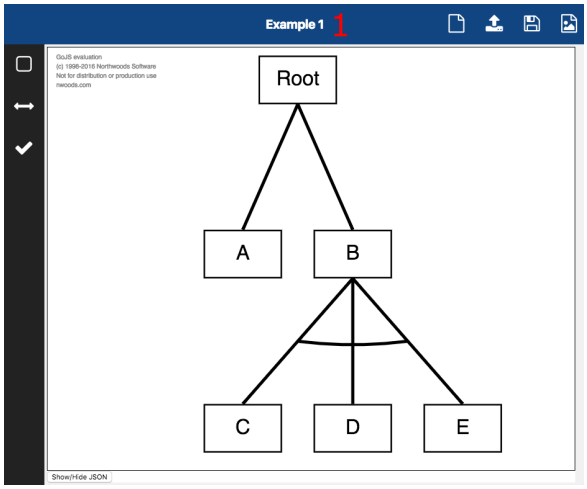
fazer *upload* de um modelo criado. Este requisito possibilita que um modelo criado, independente do usuário, possa ser carregado no sistema e modificado; **RQ7.** A ferramenta deve fornecer ao usuário a opção de validar a estrutura do modelo de características. Este requisito é necessário para auxiliar na modelagem de modelos por usuários inexperientes na notação ou em ELPS; **RQ8.** As telas de comunicação com o usuário devem ser minimalistas, contendo somente o que for necessário ao usuário para que a *interface* seja intuitiva e simples de usar.

Grande parte do esforço do levantamento de requisitos foi dedicado ao processo de seleção da notação e funcionalidades básicas disponibilizadas pela ferramenta. Foram considerados que este seja um ponto importante, visto que a ferramenta deve auxiliar uma pessoa que não possua experiência em ELPS possa representar, graficamente, uma família de software por meio de um processo de modelagem de características. Portanto, foi decidido que a primeira notação a ser disponibilizada seria a notação FODA, por conta de sua facilidade de implementação e quantidade de propriedades. Vale ressaltar que esta notação contém o menor número de propriedades dentre todas as estudadas.

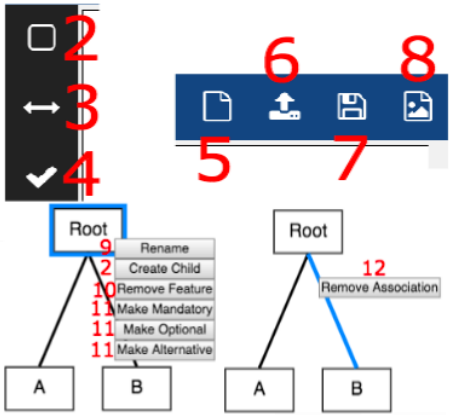
Um conjunto de funcionalidades básicas foram definidas para a ferramenta visando criar uma versão funcional, tais como criação, mudança do tipo e exclusão das características, e criação e exclusão de relacionamentos. Com estas funcionalidades implementadas e testadas, foram utilizadas validações informais com usuários durante a fase de desenvolvimento e as funcionalidades eram incluídas ou excluídas a cada validação.

4.1. Funcionalidades

As funcionalidades implementadas na Web-Atlas foram definidas a partir dos requisitos descritos nesta seção. A seguir são apresentadas três figuras representando as telas e funcionalidades da ferramenta. A Figura 2a apresenta a tela principal da ferramenta, em que estão presentes a barra de ferramentas de modelagem na lateral, menu superior da direita e a área central para modelagem. A Figura 2b mostra o menu lateral e o menu superior da direita. As figuras possuem uma numeração em vermelho, que identifica as funcionalidades juntamente às suas descrições:



(a) Tela Inteira



(b) Menu e Barra de Ferramentas

Figura 2. Tela e Menus

1. *Renomear Modelo*: permite que o usuário possa nomear seu modelo. O sistema restringirá algumas funcionalidades caso o modelo não tenha sido nomeado; **2.** *Criar Característica*: faz com que o sistema crie uma característica no modelo, esta característica, por padrão, é do tipo obrigatória. Esta função pode ser acessada no menu lateral ou clicando diretamente no modelo com o botão esquerdo. Porém, caso o usuário selecione uma característica e use o botão direito, o sistema criará uma nova característica com associação com a que estava selecionada anteriormente; **3.** *Criar Associação*: permite que o usuário crie associações entre as características. Quando selecionada esta função, o sistema solicita ao usuário o nome da característica pai e filha, criando assim a associação. Essa função pode ser acessada pelo menu lateral ou clicando com o botão direito sobre o modelo; **4.** *Verificar Modelo*: realiza a verificação das regras da estrutura de árvore e as propriedades da notação FODA. É verificado se há características sem associação, mais de uma raiz, um grupo de alternativas formado por apenas uma característica, entre outras restrições; **5.** *Criar Modelo*: permite que um novo modelo de características seja criado, ou seja, ele começa um novo documento, apagando todo o progresso de modelagem realizado anteriormente; **6.** *Carregar Modelo*: permite ao usuário abrir um modelo que foi criado e salvo como .json [Bray 2014]. Sendo assim, o usuário pode realizar alterações nos modelos; **7.** *Salvar Modelo*: faz com que o sistema mapeie tudo que foi criado no modelo pelo usuário, realize a conversão de linguagem JavaScript para JSON e transfira para a máquina do usuário um arquivo de extensão .json; **8.** *Exportar Modelo como Imagem*: realiza a captura de todos os elementos inseridos no modelos, criando assim uma imagem com extensão .png e salvando-a para a máquina do usuário; **9.** *Renomear Característica*: permite que o usuário modifique o nome da característica. Há duas formas de acessar essa função, clicando duas vezes sobre uma característica ou clicando com o direito sobre a característica e selecionando a opção "Renomear Característica"; **10.** *Remover Característica*: remove uma característica do modelo. Caso esta característica tenha associações filhas e uma associação pai, essas associações serão também removidas, mantendo as características separadas no modelo; **11.** *Alterar o Tipo da Característica*: permite que o usuário altere o tipo da característica. Os tipos podem ser: *Mandatory* (Obrigatório), *Alternative* (Alternativo) e *Optional* (Opcional). Para acessar essa função, deve-se clicar com o direito sobre uma característica e selecionar uma das três opções: *Make Mandatory*, *Make Optional*, *Make Alternative*. **12.** *Remover Associação*: possibilita ao usuário excluir uma associação do modelo, sem a necessidade de remoção das características pai e filha.

4.2. Exemplo de Uso: Linha de Produto de Celular

Com o objetivo de demonstrar as funcionalidades básicas da ferramenta Web-Atlas utilizou-se como exemplo um diagrama de funcionalidades de uma LPS de celular[1]. Este exemplo tem por objetivo mapear as característica de multimídia de um celular, tais como organização, ações, e gerenciamento dessas multimídias. O primeiro passo é utilizar a função de nomear o modelo, pois algumas funções necessitam do nome para criar arquivos externos. Com o modelo nomeado, a próxima etapa é usar a função de renomear as características para definir o nome do domínio a ser representado, neste caso, o nome atribuído foi "Multimídia para Celular".

Após a definição do domínio, são utilizadas as funções de criar e renomear as ca-

[1]Modelo retirado do repositório da S.P.L.O.T. [Mendonca et al. 2009]

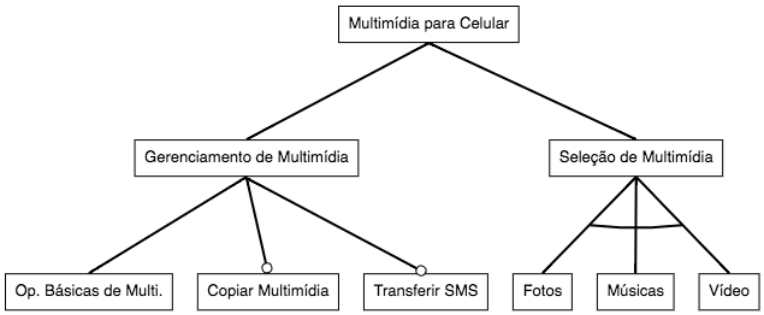


Figura 3. Modelo representando o domínio Multimídia para Celular

racterísticas, definindo assim as características mais genéricas e as filhas da raiz, sendo elas "Gerenciamento de Multimídia" e "Seleção de Multimídia". Nas próximas etapas, deve-se repetir este mesmo procedimentos, para especializar estas características. Para a especialização (filhas) da característica "Gerenciamento de Multimídia" (pai), foram definidas as filhas "Operações Básicas de Multimídia", "Copiar Multimídia", e "Transferir SMS", que são características opcionais, portanto, deve-se usar a função "*Make Optional*" para modificá-las para opcionais. A última característica filha da raiz é definida como pai de um grupo de características alternativas, sendo elas "Foto", "Vídeo" e "Música", em que utilizou-se a função "*Make Alternative*" para torná-las alternativas, então, o modelo finalizado é apresentado na Figura 3.

Após a finalização da representação das características do domínio, há a opção de utilizar a função "*Check Model*", para validar o modelo criado. Esta validação verifica certos critérios da notação, tais como: o modelo deve possuir somente uma raiz, um grupo alternativo não pode conter somente uma característica, uma característica não pode possuir dois pais, não podem haver características sem pai, não podem haver características sem nome, e a raiz não pode ser alternativa e nem opcional. Com o modelo validado, o próximo passo é salvar o modelo ou exportar como imagem. Caso selecione a função "*Save as .json*", o modelo será convertido para um arquivo .json e salvo para o usuário. O JSON gerado é em um formato específico utilizado pela Web-Atlas, como o apresentado na Figura 4. Caso seja escolhida a função "*Save as .png*", o modelo será transformado em uma imagem e transferido para o usuário.

```
{ "class": "go.GraphLinksModel","linkLabelKeysProperty": "labelKeys","nodeDataArray": [
{"key":"Multimídia para Celular", "category":"mandatory"},
…
"category":"LinkLabel"},{"key":"Alt1", "category":"LinkLabel"}],"linkDataArray": [
{"from":"Seleção de Multimídia-Vídeo", "to":"Seleção de Multimídia-Música", "labelKeys":
[ "Alt1" ],"category":"linkToLink"},
…
{"from":"Seleção de Multimídia", "to":"Vídeo", "labelKeys":[ "Seleção de Multimídia-
Vídeo" ]}]}
```

Figura 4. Exemplo simplificado do JSON gerado pela Web-Atlas

5. Conclusão

Inicialmente, neste trabalho foram apresentados os conceitos básicos de LPS, modelos e diagramas de características, incluindo as propriedades da notação FODA. Realizando-se o levantamento da características essenciais das ferramentas relacionadas e da Web-Atlas,

percebe-se que há apenas uma característica que não está implementada na Web-Atlas, sendo ela a possibilidade de trabalho colaborativo. Apenas a WebSPIFF contém todas as características implementadas, porém, sua licença é comercial. Outras ferramentas possuem características próprias, tais como a S.P.L.O.T., que mantém um repositório online com modelos criados por seus colaboradores, que podem servir como exemplos de uso ou aprendizado.

A Web-Atlas é a evolução da primeira versão da Atlas, desenvolvida visando auxiliar a área acadêmica, sendo possível sua utilização na área industrial. Esta ferramenta tem como funcionalidade criar, editar, salvar, abrir e exportar os modelos de características baseados na notação FODA. Para trabalhos futuros, as principais metas são: a implementação de novas notações, um estudo sistemático para verificar quais são as notações em destaque na comunidade de LPS, a validação com usuários e casos de testes, e a implementação de transformação entre os modelos suportados pela ferramenta.

Referências

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

Apel, S., Batory, D., Kästner, C., and Saake, G. (2013). *Feature-Oriented Software Product Lines: Concepts and Implementation*. Springer-Verlag Berlin Heidelberg.

Bachmann, F. and Clements, P. C. (2005). Variability in software product lines. Technical report, Carnegie-Mellon Univ. Pittsburgh PA software engineering Inst.

Bidarra, R., van den Berg, E., and Bronsvoort, W. F. (2001). Web-based collaborative feature modeling. In *SMA*, pages 319–320, New York, NY, USA. ACM.

Bray, T. (2014). The javascript object notation (json) data interchange format.

Czarnecki, K., Helsen, S., and Eisenecker, U. (2004). Staged configuration using feature models. In *SPLC*, pages 266–283. Springer.

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., and Peterson, A. S. (1990). Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document.

Kastner, C., Thum, T., Saake, G., Feigenspan, J., Leich, T., Wielgorz, F., and Apel, S. (2009). FeatureIDE: A tool framework for feature-oriented software development. In *ICSE*, pages 611–614. IEEE.

Mendonca, M., Branco, M., and Cowan, D. (2009). S.P.L.O.T.: Software Product Lines Online Tools. In *ACM SIGPLAN*, pages 761–762, New York USA. ACM.

Pohl, K., Böckle, G., and van Der Linden, F. J. (2005). *Software product line engineering: foundations, principles and techniques*. Springer Science & Business Media.

Rodrigues, E., Zorzo, A., de Oliveira, E., Gimenes, I., Maldonado, J. C., and Domingues, A. (2012). PlugSPL: An Automated Environment for Supporting Plugin-based Software Product Lines. In *SEKE*, pages 647–650.

SEI, T. S. E. I. (2017). Software product lines. `http://www.sei.cmu.edu/productlines/`. Acessado: 30-09-2017.

Aplicando Técnicas de V&V na Fase de Projeto do Modelo em Cascata

Naihara Sousa Amorim, Gilleanes Thorwald Araujo Guedes

[1]Universidade Federal do Pampa (UNIPAMPA)
Campus de Alegrete RS – Brasil Curso de Engenharia de Software

naiharamosouza@gmail.com, gilleanesguedes@unipampa.edu.br

Abstract. *The Waterfall Model has well defined steps that occur only once, since it is a non-iterative model. A problem perceived during the Waterfall Model study is that there is no specific activity in the Design stage that verifies the system models created in it, which can lead to the construction of a system with defects. Therefore, the purpose of the article is to integrate Verification and Validation techniques in the Design phase, in order to improve the construction of the system models of the project. In this way, the process will have more technical precision, contributing to the quality of the final product.*

Resumo. *O Modelo em Cascata possui fases bem definidas e que ocorrem apenas uma vez, visto que é um modelo não iterativo. Um problema percebido durante o estudo do Modelo em Cascata é a inexistência de uma atividade específica na fase de Projeto, que verifique os modelos produzidos. Sendo assim, a proposta deste artigo é, integrar na fase de Projeto técnicas de Verificação e Validação, a fim de inspecionar os modelos produzidos durante o projeto. Desta maneira pretende-se que o processo tenha mais precisão técnica, contribuindo para a qualidade do produto final.*

1. Introdução

A Engenharia de Software é uma área de estudo que busca melhorar o desenvolvimento de software com o objetivo de torná-lo mais eficaz, mais robusto e com melhor qualidade. Com este objetivo pesquisadores dessa área propuseram vários processos de desenvolvimento. Um dos primeiros processos propostos foi o Modelo em Cascata.

As fases do modelo em Cascata possuem um encadeamento, sendo que, uma etapa não deve ser iniciada antes que uma etapa anterior seja concluída. Na prática, esses estágios se sobrepõem e alimentam uns aos outros de informações. O modelo em Cascata é também um processo dirigido a planos, ou seja, suas atividades são planejadas antes de serem executadas. A fase de Projeto no Modelo em Cascata, é um processo de vários passos que é centralizado em quatro atributos diferentes do sistema: estrutura de dados, arquitetura do software, detalhes procedais e caracterização das interfaces [Sommerville 2011]. O processo de projeto representa os requisitos de uma forma que permita a codificação do produto (é uma prévia etapa da codificação). Da mesma maneira que a análise dos requisitos, o projeto deve ser documentado. Para atingir este objetivo, o projetista deve mapear as estruturas e funcionalidades identificadas na análise de requisitos dentro do contexto e das restrições da arquitetura, de forma a tornar possível a construção do software, isto é feito principalmente por meio da criação de modelos.

Todavia, Segundo [Sommerville 2011] problemas de modelagem, podem levar a sistemas mal estruturados, quando os problemas de projeto são contornados por artifícios de implementação. Uma das possíveis maneiras de mitigar possíveis defeitos nos artefatos é a aplicação de técnicas de Verificação e Validação(V & V). Estas técnicas visam melhorar a execução do processo e reduzir a quantidade de retrabalho, bem como evitar que soluções inadequadas possam ser implementadas nas etapas seguintes. Assim com o objetivo de aumentar a qualidade do produto, propõe-se integrar ao Modelo em Cascata a utilização de técnicas de V & V, que serão aplicadas nos modelos construídos durante a fase de Projeto.

Este artigo possui a seguinte estrutura: na seção (2) o Modelo em Cascata é apresentado em detalhes com ênfase na fase de Projeto. A seção (3) descreve algumas técnicas de Verificação e Validação que podem ser aplicadas aos modelos de um projeto. Na seção (4) é proposta a inserção de técnicas de Verificação e Validação no modelo cascata durante a fase de projeto. Na última seção são apresentadas as conclusões do artigo.

2. Modelo em Cascata

Um dos Modelos de processo mais tradicionais na Engenharia de Software é o Modelo em Cascata, que tem como principal característica um conjunto de etapas e atividades bem definidas. Este modelo segue um fluxo linear e só avança para a próxima fase quando a fase anterior estiver terminada e revisada [Wazlawick 2012].

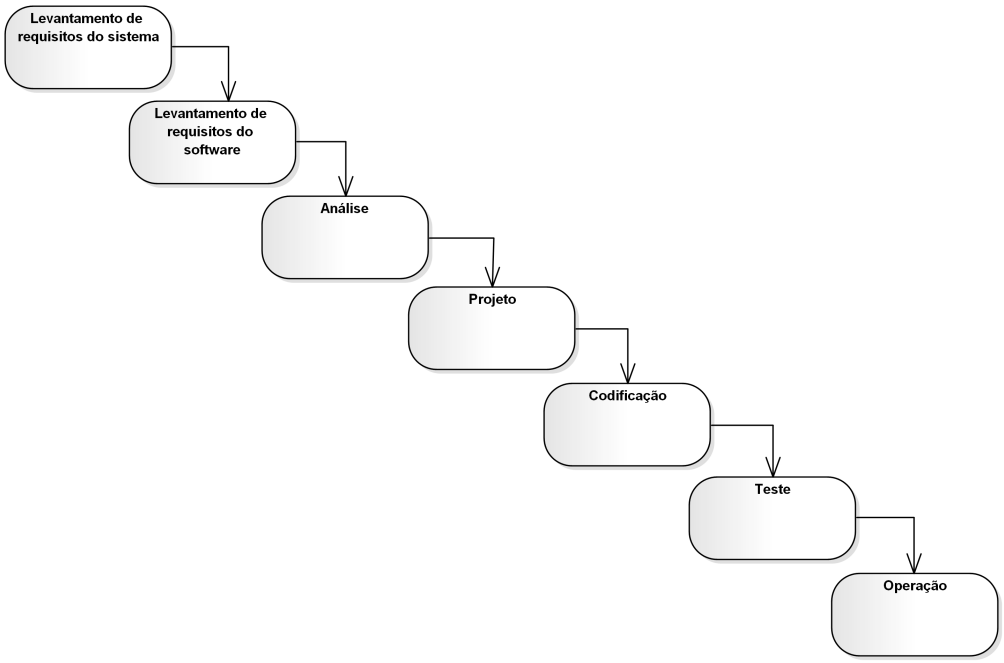


Figura 1. Modelo Cascata [Wazlawick 2012]

O Modelo baseia-se na filosofia BDUF (Big Design Up Front), que propõe que antes de produzir linhas de código deve-se fazer um trabalho detalhado da fase de análise e projeto. A utilização do Cascata sugere que todas as etapas do processo sejam bem documentadas e bem definidas, com destaque para as fases anteriores ao desenvolvimento

de linhas de código. Estas etapas são as que mais produzem artefatos de software e são de extrema importância para que o produto final satisfaça os requisitos do cliente.

Dentro do Modelo em Cascata clássico existem sete (7) fases[Wazlawick 2012]: Levantamento de requisitos do Sistema, Levantamento de Requisitos do Software, Análise, Projeto do Programa, Codificação, Teste, e Operação, conforme pode ser observado na Figura 1.

Na fase de levantamento de requisitos do software as metas e restrições do produto são estabelecidas por meio de consulta aos usuários. A fase seguinte, de levantamento de requisitos do sistema, aprofunda e detalha essas informações, enquanto que a fase de análise verifica inconsistências e conflitos nos requisitos.

Na fase de projeto são identificadas e descritas as abstrações fundamentais do software e seus relacionamentos. Após seu término, na fase de codificação o projeto do software é traduzido em um conjunto de programas ou unidades de programa e, na fase de teste, esse código é testado para determinar se o software funciona adequadamente de acordo com o que foi solicitado. Finalmente na fase de operação, o sistema passa a ser utilizado. Em geral está é a fase mais longa do ciclo de vida.

O enfoque deste trabalho concentra-se na fase de projeto. A figura 2 representa por meio do Modelo BPMN (Business Process Model and Notation), os passos desta etapa como definidos por [Boehm 1981].

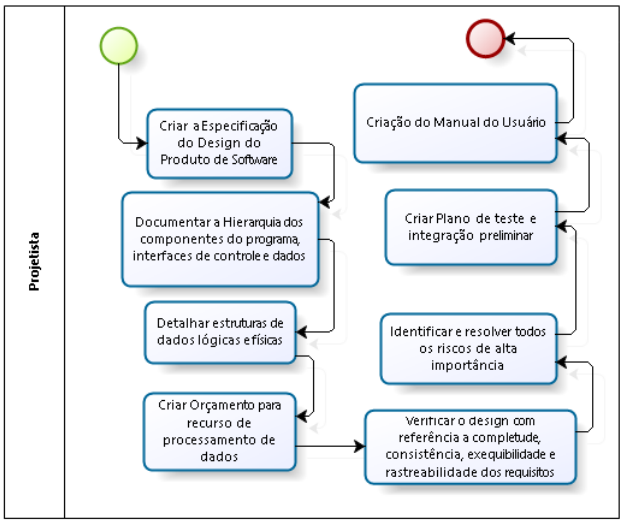


Figura 2. Atividades da etapa de projeto [Boehm 1981]

3. Técnicas de Verificação e Validação(V & V)

A qualidade é hoje um grande motivador em todas as áreas de atividade humana, todos querem oferecer e receber produtos e serviços com qualidade [Tsukumo et al. 1997]. A cada dia é maior o número de empresas que buscam melhorias em seus processos de desenvolvimento de software. Além do aumento da produtividade e da diminuição do retrabalho, elas buscam um melhor relacionamento com seus clientes por meio do melhor planejamento e gestão de suas atividades de desenvolvimento e da diminuição no número de defeitos nos produtos entregues [Jomori et al. 2004].

O processo de Garantia da Qualidade utiliza-se de atividades de V&V, com o intuito de encontrar erros em todas as etapas do desenvolvimento. Essas atividades são muito importantes, pois cuidam de analisar diversos pontos do processo de desenvolvimento, impedindo que um erro se propague para as fases posteriores do projeto [Melo 2009].

Sobre o propósito da V&V, pode-se dizer que o objetivo é confirmar que cada produto de trabalho do processo ou do projeto atende apropriadamente os requisitos especificados. Já com relação à validação, seu propósito é confirmar que um produto ou componente do produto atenderá a seu uso pretendido quando colocado no ambiente para o qual foi desenvolvido [Longen 2013].

De acordo com a [IEEE 1990] "verificação e validação(V&V) é o processo que consiste em se determinar: se os requisitos do sistema ou componentes estão completos e corretos; se os produtos de cada fase de desenvolvimento preenchem os requisitos ou condições impostas pela fase anterior; se o sistema final ou componente está de acordo com os requisitos especificados".

Verificação é considerada uma técnica estática que avalia a qualidade do documento e corrige defeitos presentes no documento sem a necessidade de exercitar uma aplicação. Se defeitos não são detectados e corrigidos na fase inicial de desenvolvimento, o custo será muito alto, porque os defeitos serão propagados para a próxima fase do desenvolvimento. Já a validação é considerada uma técnica dinâmica porque envolve exercitar uma implementação. Técnica estáticas podem ser usadas em todos os estágios do processo de desenvolvimento, enquanto técnicas dinâmicas são usadas quando um protótipo ou um programa executável está disponível[Sommerville 2011].

Inspeção de software é um tipo particular de revisão que possui um processo de detecção de defeitos rigoroso e bem definido [Fagan 1976]. È considerada um tipo de verificação estática, que tem o objetivo de detectar defeitos antes que a fase de teste inicie, contribuindo para melhorar a qualidade do software em relação ao custo beneficio [Debaud 1990]. Inspeções foram criadas por Michael Fagan da IBM [Fagan 1976].O processo para inspeção formal contém atividades e papéis bem-definidos, dados do produto, o produto a ser inspecionado e uma infra-estrutura de suporte [NASA 1993].

Segundo o padrão [IEEE 1998], o processo de inspeção contém características que verificam se o elemento de software está conforme o padrão aplicado e identificam desvios de padronização e especificação. Pode ser usada em todos os estágios do desenvolvimento de software e são aplicadas em um artefato em particular, como especificação de requisitos, projeto e código.

Segundo [Bertini 2006] a inspeção é um método que contribui para garantir a qualidade do produto de software. Todas as etapas do processo de desenvolvimento de software são suscetíveis à incorporação de defeitos, que podem ser detectados pela inspeção e posteriormente removidos.

Concluindo de acordo com [Melo 2009] a inspeção de modelo é uma das técnicas que cujo objetivo é encontrar erros a partir da leitura. Além disso, a inspeção é uma técnica usada durante revisões formais. Sua utilização tem como intuito, por exemplo: descobrir erros de função, de lógica ou implementação para qualquer representação do software e também garantir que o software foi representado de acordo com padrões pré-

definidos, garantindo, assim, que o sistema seja desenvolvido de maneira uniforme.

As atividades do processo de inspeção é formal e composto por seis atividades: planejamento; visão geral; preparação; reunião da inspeção; re-trabalho; e acompanhamento.

4. Integração de V&V sob Modelo em Cascata durante a fase de Projeto

A Verificação e Validação sob Modelo Cascata agregam valor ao processo, principalmente pelo fato de que há grande volume de documentação, em destaque nas etapas de análise e de projeto, gerando diversos tipos de artefatos, o que contribui para uma melhor construção do produto final. Como o Modelo preza a qualidade dos artefatos gerados, técnicas de Verificação e Validação potencializam a qualidade do software antes mesmo de ser codificado.[Wazlawick 2012]

4.1. Integração de Modelos na Fase de projeto

Na fase de projeto é onde a maioria de modelos utilizados para a construção do projeto são criados. Os documentos produzidos em cada fase podem ser modificados para refletirem as alterações feitas em cada um deles [Sommerville 2011]. Portanto percebe-se que erros na modelagem do sistema podem ser encontrados nas etapas seguintes, quando teoricamente a etapa de projeto deveria estar revisada e finalizada. Os problemas encontrados causam retrabalho não apenas na fase de projeto, mas também em suas etapas posteriores.

Em nossa proposta pretendemos focar fortemente nas técnicas de V&V. As técnicas serão aplicadas sobre os modelos a fim de identificar possíveis defeitos na modelagem. Caso seja descoberta alguma irregularidade no projeto, o problema será documentado e terá seu impacto mapeado em outros artefatos além de, claramente, ser corrigido pelo projetista. A Figura 3 descreve detalhadamente as etapas da fase de projeto integrada à técnicas de V&V.

A primeira etapa de nossa proposta é a fase de planejamento, na qual são definidos os participantes da inspeção, seus respectivos papéis e responsabilidades. Nessa etapa o material de inspeção é repassado aos inspetores. A atividade de visão geral consiste em uma reunião, durante a qual são fornecidas à equipe de inspeção as informações necessárias à inspeção. A reunião de visão geral ocorrerá se a equipe de inspeção não possuir familiaridade com o produto. Durante a atividade de preparação, os artefatos são verificados e validados com o objetivo de encontrar defeitos e, caso algum seja encontrado, classificá-los de acordo com o tipo de defeito. A atividade de preparação, de acordo com [Cabral 2007], é considerada a mais importante de uma inspeção, porque seu foco principal é a detecção de defeitos.

A fase de preparação na nossa proposta deve ser organizada individualmente. Após isso, durante a atividade denominada de reunião de inspeção ocorre a consolidação de defeitos encontrados durante a fase de preparação. O objetivo da atividade de retrabalho é corrigir os defeitos encontrados, os defeitos classificados com prioridade alta serão corrigidos de imediato, enquanto os de prioridade baixa só o serão caso o custo e o prazo permitirem. Após verificar se todos os defeitos foram corrigidos, o Analista de Modelo deve aprovar o artefato e assim registrar o fim da inspeção e, caso o produto não for aprovado, deverá notificar o projetista.

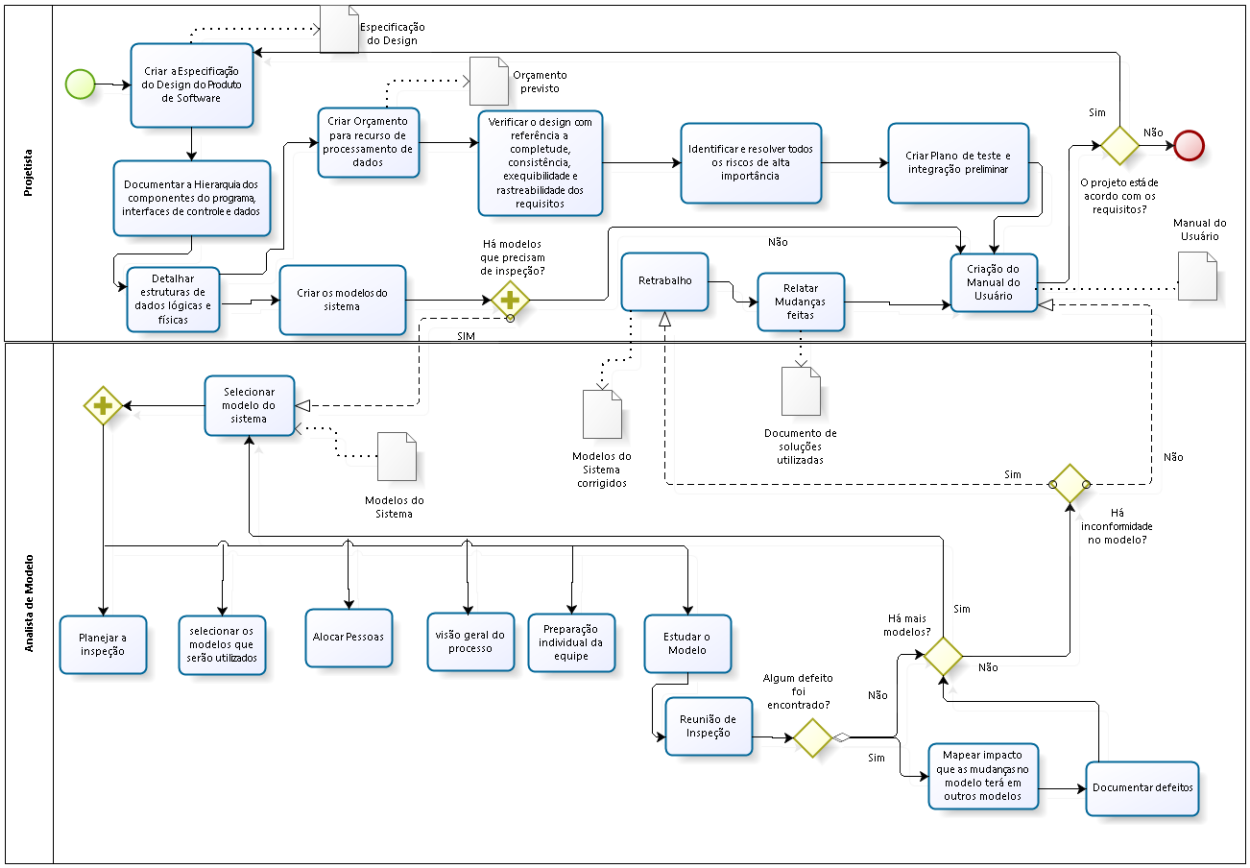


Figura 3. Modelo Cascata com Verificação e Validação na Fase de Projeto

Após as inspeções o projetista deverá ser informado sobre as possíveis inconsistências dos artefatos do sistema e terá a tarefa de corrigi-los. Este processo é reexecutado até que todos os modelos considerados importantes pelo Analista de Modelo sejam verificados. Assim os Modelos de Sistema são refeitos em uma atividade específica, evitando o retrabalho em futuras atividades do processo.

4.2. Analista do Modelo

Para a realização da adaptação do processo é adicionado um novo papel, o Analista de Modelo.

O Analista de Modelo é responsável por selecionar os modelos mais relevantes para o projeto e aplicar a técnica de Inspeção de modelo como as OORT's [Travassos et al. 2002]. Caso seja encontrado algum problema sua responsabilidade é mapear e documentar as possíveis causas dos defeitos e reportar para o Projetista.

O processo de inspeção OORT's de software pode ser aplicado a qualquer tipo de artefato do sistema, incluindo o documento de requisitos. Em relação a Diagramas de Sequência e Casos de Uso, pode-se, por exemplo, verificar se os Diagramas de Sequência descrevem uma combinação apropriada de objetos e mensagens que trabalham em conjunto para capturar a funcionalidade descrita pelo caso de uso. Já com relação a Diagramas de Sequência e Classe é possível, por exemplo, verificar se um determinado Diagrama de Classes descreve as classes e suas associações de forma que os comportamentos espe-

cificados nos Diagramas de Sequência estejam representados corretamente. Mas para que isso possa acontecer, é fundamental o uso de técnicas específicas e adequadas no decorrer de toda Verificação e Validação [Travassos et al. 2002]

5. Conclusão

A atividade de criação de modelos de sistemas não é uma tarefa trivial, ela requer muito cuidado, visto que um defeito produzido em algum modelo pode acarretar em graves problemas na construção do sistema. Acreditamos que técnicas de V&V devem ser aplicadas na fase de projeto com o objetivo de identificar possíveis defeitos de modelagem.

Para isso deve-se exigir mais esforço na etapa de projeto para aplicar técnicas de Verificação e Validação. Essas técnicas estão relacionadas, particularmente, à área de Qualidade de Software, que tem como objetivo auxiliar o time de desenvolvimento a construir produtos com um nível de qualidade aceitável.

De acordo com a nossa visão, essa proposta possui pontos positivos e negativos em relação à integração de Verificação e Validação no modelo cascata.

Os pontos positivos identificados foram cinco, a saber: (1) diminuir a quantidade de retrabalho em etapas seguintes à etapa de projeto; (2) proporcionar maior qualidade na documentação do projeto; (3) mapear problemas detectados com antecedência; (4) garantir maior conformidade dos artefatos; (5) proporcionar maior qualidade no produto final.

Como pontos negativos consideramos que: (1) realizar inspeções sobre o projeto são consideradas caras, embora a longo prazo elas sejam na verdade uma forma de economizar custos de manutenção; e (2) um novo membro da equipe de desenvolvimento deve ser alocado para ser o papel de Analista de Modelo, o que tende a deixar o processo mais custoso em relação ao tempo e orçamento.

Muitas técnicas de Verificação e Validação já são utilizadas nos processos de construção de software. Entretanto, essa proposta tem como objetivo conduzir as técnicas de Verificação e Validação dentro da etapa de Projeto, fazendo com que os modelos de sistema criados nessa fase sejam produzidos de acordo com as boas práticas da Engenharia de Software. Em suma, essa proposta visa reduzir diversos problemas que uma má modelagem pode acarretar em outras etapas do modelo cascata, melhorando a qualidade do processo em geral.

Como forma de validação, pretendemos aplicar essa proposta na Universidade Federal do Pampa (UNIPAMPA) em turmas do curso de Engenharia de Software, particularmente durante as disciplinas de Resolução de Problemas, que são disciplinas em que um problema maior, próximo à realidade do mercado, é apresentado às turmas. Pretendemos tabular os resultados e compará-los com os resultados de turmas que utilizaram o modelo em cascata padrão para determinar se a aplicação do processo de desenvolvimento em cascata com forte ênfase na aplicação de técnicas de verificação e validação na fase de projeto demonstra ser superior ao processo em cascata padrão.

Referências

Bertini, Tereza G. Kirner, M. I. M. I. A. L. (2006). *Técnicas de Inspeção de Documentos de Requisitos de Software: um Estudo Comparativo*. WER06 - Workshop em Enge-

nharia de software.

Boehm, B. W. (1981). *Software Engineering Economics*. Prentice Hall.

Cabral, M. S. (2007). Técnicas de leitura para inspeção da especificação inicial de requisitos.

Debaud, J.M., L. (1990). ”an encompassing life cycle survey of software inspectons”.

Fagan, M. (1976). ”design and code inspections to reduce errors in program development”.

IEEE, C. S. P. (1990). standard glossary of software engineering terminology.

IEEE, C. S. P. (1998). Ieee standard for software reviews and audits.

Jomori, S. M., VOLPE, R., and Zabeu, A. C. P. (2004). Qualidade de software. *Revista TECHOJE,(IETEC-Instituto de Educação Tecnológica), ano XIII. Disponível em:¡ http://www. ietec. com. br/ietec/techoje/techoje/tecnologiadainformacao/2004/07/01/2004_07_01_0001. 2xt/-template_ interna*.

Longen, S. J. (2013). Desenvolvimento de um processo de verificação e validação para produtos de software.

Melo, S. M. (2009). Inspeção de software. *University of São Paulo: São Carlos, SP*.

NASA (1993). Software formal inspections guidebook.

Sommerville, I. (2011). *Engenharia de Software, 9ª edição*. Pearson Education, Inc.

Travassos, G. H., Shull, F., Carver, J., and Basili, V. (2002). Reading techniques for oo design inspections. Technical report.

Tsukumo, A. N., Rêgo, C. M., Salviano, C. F., Azevedo, G. F., Meneghetti, L. K., Costa, M. C., CARVALHO, M. B. d., and Colombo, R. M. (1997). Qualidade de software: visões de produto e processo de software. *II ERI–SBC, Piracicaba, São Paulo, Brasil*.

Wazlawick, R. S. (2012). *Engenharia de Software para Sistemas de informação:Conceitos e Práticas que fazem sentido*. Pearson Education, Inc.

Segurança na Web: análise black-box de scanners de vulnerabilidades

Isadora Garcia Ferrão[1], **Diego Kreutz**[1]

[1]Universidade Federal do Pampa - Campus Alegrete (UNIPAMPA)
Av. Tiarajú, 810 - Ibirapuitã - 97.546-550 - Alegrete - RS - Brasil

isadora-gf@hotmail.com, Diego.Kreutz@unipampa.edu.br

Abstract. *Vulnerability scanners are essential tools for monitoring eminent security risks on Web systems. Unlike the existing literature, this paper presents a detailed black-box analysis of a significant set of free vulnerability scanners. The results allow to identify the main differences between existing scanners, contributing to a better selection of these important tools when performing vulnerability scans on Web systems.*

Resumo. *Os scanners de vulnerabilidades são ferramentas essenciais para monitorar riscos de segurança eminentes em sistemas Web. Diferentemente da literatura existente, este trabalho apresenta uma análise black-box detalhada de um conjunto significativo de scanners de vulnerabilidades gratuitos. Os resultados permitem identificar as principais diferenças entre scanners existentes, contribuindo para uma melhor seleção dessas importantes ferramentas na hora de realizar varreduras de vulnerabilidades em sistemas Web.*

1. Introdução

Sistemas *Web* fazem parte do cotidiano da maioria das pessoas. Segundo estatísticas, a cada ano milhares de novos sistemas são implementados e lançados na *Web* [Baldessar 2014]. Entretanto, apesar de a maioria desses sistemas possuírem um grande impacto social ou econômico, um conjunto significativo deles são lançados com vulnerabilidades críticas de segurança [Vieira et al. 2009]. A principal causa dessa alarmante constatação é o fato de os programadores se preocuparem essencialmente com as funcionalidades do produto final, pois possuem restrições de tempo para entrega do produto e não possuem os conhecimentos necessários sobre segurança de sistemas. Isto leva a um cenário propício à exploração maliciosa de dados e componentes de sistemas como comprovam relatórios de segurança atuais de grandes empresas, com equipes altamente especializadas em segurança de sistemas, como a Symantec e a Cisco.

Existem diferentes vulnerabilidades e ataques exploratórios que podem prejudicar os sistemas. Entre as mais comuns estão os ataques de injeção de código, *cross-site scripting* (XSS), quebra de autenticação e gerenciamento de sessão, referência insegura e direta a objetos, configuração incorreta de segurança, exposição de dados sensíveis, falta de função para controle do nível de acesso, *cross-site request forgery*, utilização de componentes vulneráveis conhecidos e redirecionamentos e encaminhamentos inválidos [OWASP 2017]. Na prática, os atacantes utilizam diferentes ferramentas para identificar e explorar estas vulnerabilidades dos sistemas, como *scanners* de vulnerabilidades disponíveis no mercado e *scripts* automatizados de teste e invasão.

Os *scanners* de vulnerabilidades são ferramentas especialmente desenvolvidas para diagnosticar defeitos de instalação, configuração e segurança de sistemas. Na engenharia de *software*, mais especificamente no ciclo de vida do *software*, este tipo de ferramenta é um instrumento que pode contribuir significativamente na automatização dos testes de robustez e segurança dos sistemas *Web*. Empresas e órgãos governamentais utilizam constantemente este tipo de ferramentas para detectar e corrigir vulnerabildades de sistemas. Como exemplo prático, a área de tecnologia da UNIPAMPA utiliza *scanners* de vulnerabilidades para testar periódicamente máquinas e sistemas. No momento em que vulnerabilidades são detectadas, usuários, desenvolvedores e especialistas de segurança são notificados para que as medidas apropriadas sejam tomadas.

A maioria dos *scanners* de vulnerabilidades de sistemas *Web* são usados como ferramentas para testes automatizados, de caixa preta, ou seja, sem acesso ao código-fonte, com o objetivo de identificar falhas e vulnerabilidades [Fong and Okun 2007]. As ferramentas de caixa preta simulam um ataque exploratório e fornecem relatórios contendo o local e as especificações das vulnerabilidades encontradas [Bau et al. 2010]. Essas ferramentas se tornaram tão importantes que *StartUps* estão sendo criadas com um único objetivo: prover varredura de vulnerabilidades para empresas e governos. Alguns exemplos são a On-security (`www.on-security.com`), Pentest-Tools (`pentest-tools.com`), Valency Networks (`www.valencynetworks.com`) e Veracode (`www.veracode.com`), especializadas em contratos de varredura de vulnerabilidades em sistemas *online*. Não apenas as empresas privadas, mas também instituições governamentais e forças armadas estão cada vez mais preocupados com a qualidade dos seus sistemas em termos de robustez e segurança, gerando contratos e oportunidades de negócio para empresas de segurança, em especial empresas especializadas em detecção de vulnerabilidades.

Apesar da existência de inúmeros *scanners*, que tem como objetivo detectar vulnerabilidades de sistemas *Web*, estudos demonstram que há disparidade entre as ferramentas existentes em termos de abrangência e níveis de exploração das vulnerabilidades [Rocha et al. 2012, Doupé et al. 2010, Vieira et al. 2009]. Os resultados apresentados na literatura indicam claramente a necessidade de não apenas uma única ferramenta, mas sim um conjunto de *scanners* para garantir uma boa cobertura na detecção de vulnerabilidades de sistemas *Web*. Outro aspecto importante de ser ressaltado é o fato de os estudos existentes serem relativamente antigos, isto é, não trazem o que há de mais recente em termos de ferramentas e técnicas de detecção de vulnerabilidades e, também, não explorarem a grande variedade e diversidade de ferramentas livres e abertas. Na verdade, alguns estudos exploram, em paticular, ferramentas comerciais.

Este trabalho tem como objetivo principal realizar uma análise *black-box* de *scanners* de vulnerabilidades livres e abertos. Para a análise foram selecionados dez dos principais *scanners* disponíveis na *Internet*, incluindo Uniscan, Paros Proxy, Zed Attack proxy, Nessus, Arachni, Grabber, Wapiti, Ratproxy, Skipfish e Vega [Rocha et al. 2012, Offensive Security 2017, INFOSEC 2017]. Estas ferramentas foram instaladas em uma máquina virtual Linux, ou sistema hospedeiro, e executadas sobre o mesmo alvo, uma máquina virtual do projeto BWA *(Broken Web Applications)* da OWASP [OWASP 2016]. No decorrer da análise comparativa das ferramentas, o trabalho procura responder três questões: *1- Quais vulnerabilidades são detectadas por cada scanner? 2- Qual a cober-*

tura de cada scanner? 3- Quais as vulnerabilidades mais frequentemente identificadas pelos scanners?

Os 10 *scanners* detectaram 28 vulnerabilidades nos sistemas da máquina virtual BWA. Entretanto, a média de vulnerabilidades detectadas pelos *scanners* foi baixa, isto é, aquém do esperado, com excessão do Zed Attack Proxy. Este *scanner* detectou 60.71% das vulnerabilidades, enquanto que o segundo melhor detectou apenas 35,71% e o pior deles ficou limitado a meros 3,57%. Estes resultados estão condizentes com outros trabalhos similares, ou seja, há discrepância significativa entre as ferramentas. Portanto, pode-se inferir que é imperativo avaliar os *scanners* de vulnerabilidades antes de decidir quais deles utilizar para manter os sistemas *Web* da empresa ou instituição livres de *bugs* passíveis de exploração por atacantes. No estudo realizado, a combinação dos *scanners* Zed Attack Proxy, Paros Proxy, Nessus, Skipfish e Vega oferece uma cobertura de 100% das vulnerabilidades presentes nos sistemas testados.

O restante deste texto está organizado como segue. A seção 2 apresenta a metodologia utilizada no processo de avaliação das ferramentas selecionadas. Na seção 3 são apresentados e discutidos os resultados empíricos dos testes realizados com *scanners* de vulnerabilidades. Por fim, são apresentados os trabalhos relacionados na seção 4 e a conclusão na seção 5.

2. Metodologia

Os *scanners* de vulnerabilidades tem evoluído significativamente com o passar do tempo. Da mesma forma, os testes realizados por estas ferramentas tem se tornado cada vez mais sofisticados. Como resultado, continuam sendo eficazes aliados na detecção de vulnerabilidades em sistemas.

O tipo mais comum de testes para *scanners*, também utilizado neste trabalho, é o caixa preta [Bau et al. 2010, Rocha et al. 2012, Doupé et al. 2010, Vieira et al. 2009, Perlman et al. 2016]. Os testes de caixa-preta são realizados de forma automatizada e sem acesso direto a detalhes de infraestrutura e codificação do sistema alvo.

Para realização dos testes, foram selecionadas as ferramentas Uniscan, Paros, Zed Attack Proxy, Nessus, Ratproxy, Grabber, Wapiti, Andiparos, Skipfish e Vega. Os principais motivos da escolha foram: serem gratuitas e de código aberto, não existirem estudos que comparem estas ferramentas, similaridade de objetivos entre elas e sites de classificação de ferramentas de segurança [Mundo Hacker 2017, Terminal Root 2017].

Os testes foram executados utilizando uma máquina Dell *Inspiron Special Edition*, Intel core i7, sexta geração, memória de 16 GB e 1 TB de HD + 8 GB de SSD. As ferramentas foram instaladas em uma máquina virtual (MV) com o sistema operacional Kali Linux versão 4.9.0. Como alvo para avaliar os *scanners*, foi utilizada uma máquina virtual OWASP BWA. Esta MV é constituída por um sistema operacional GNU/Linux, versão 1.2 e um conjunto de aplicativos *Web* (em diferentes linguagens de programação, como PHP, Perl e Python) vulneráveis especialmente preparados para testes de segurança.

3. Resultados

A tabela 1 sumariza os resultados obtidos com os testes de caixa-preta. As ferramentas são identificadas pelos seguintes índices: [1] Uniscan, [2] Paros, [3] Zed Attack Proxy,

[4] Nessus, [5] Ratproxy, [6] Grabber, [7] Wapiti, [8] Andiparos, [9] Skipfish e [10] Vega. Na tabela aparecem os nomes das vulnerabilidades, as ferramentas que a detectaram e o nível de gravidade da vulnerabilidade, respectivamente. O nível de gravidade foi definido conforme consta nos relatórios de saída dos *scanners*. Vale ressaltar que todos que detectaram uma determinada vulnerabilidade identificavam o mesmo nível de gravidade. Em outras palavras, não houve nenhuma discrepância entre os *scanners* em termos de criticidade das vulnerabilidades detectadas.

A maioria das vulnerabilidades são de criticidade alta ou média. As vulnerabilidades de criticidade alta representam 35,7% do total de vulnerabilidades detectadas. As vulnerabilidades com gravidade alta podem comprometer a confidencialidade, integridade e disponibilidade de um sistema. Da mesma forma, a mesma porcentagem foi alcançada pelas vulnerabilidades de criticidade média. Portanto, a soma das vulnerabilidades de gravidade média e alta perfaz 71,4% do total.

Para atacar uma vulnerabilidade do tipo mediana, o invasor precisa de acesso especializado, interação do usuário, ou outras circunstâncias que estão além do controle do invasor [Viegas 2016]. Uma vulnerabilidade de alta gravidade é mais fácil de ser explorada, pois não precisa de interação do usuário ou acesso especializado. De qualquer forma, é importante frizar que todas as vulnerabilidades de alta e média criticidade devem ser tratadas imediatamente para evitar incidentes, como vazamento de informações e indisponibilidade de sistemas.

As vulnerabilidades *external redirect, remote OS command injection, x-frame-options header not set, format string error, buffer overflow, x-content-type-options header missing, incorrecting caching directives, external content embedded on a page, unix operating system unsupported version detection, ssh weak algorithms supported, samba badlock vulnerability e shell injection* foram detectadas por apenas uma única ferramenta, como pode ser observado na tabela 1. Isto indica claramente a necessidade de múltiplos *scanners* de vulnerabilidades para garantir uma verredura mais completa e eficaz de sistemas *Web*.

É interessante observar que alguns *scanners*, de cobertura bastante baixa, como Nessus, Skipfish e Vega, detectaram vulnerabilidades específicas, que as demais ferramentas não detectaram. Já a ferramenta Zed Attack Proxy, além de detectar a maior porcentagem de vulnerabilidades, também detectou o maior número de vulnerabilidades de alta gravidade. Entretanto, sozinha ela não oferece uma cobertura completa (100% das vulnerabilidades) para o cenário da BWA. Mas, em conjunto com a Paros Proxy, Nessus, Skipfish e Vega, pode-se atingir uma cobertura de 100%.

A tabela 2 apresenta alguns resultados e métricas relativos aos processos de detecção de cada ferramenta, como a porcentagem em relação a todas as vulnerabilidades detectadas, o tempo de execução e a taxa de detecção das vulnerabilidades que possuem gravidade de nível alto. Como pode ser observado na tabela, o tempo de execução não tem a ver necessariamente com a cobertura. Os *scanners* Uniscan e Vega, apesar de possuirem um alto tempo de execução, não detectaram um número significativo de vulnerabilidades. Este elevado tempo de execução ocorre devido ao fato de estas ferramentas possuírem um número maior (e mais diversificado) de combinações de testes para determinadas vulnerabilidades, como *SQL injection*. Apenas as ferramentas Zed Attack e Skipfish detectaram

50% ou mais das vulnerabilidades de criticidade alta.

Tabela 1. Vulnerabilidades detectadas pelas ferramentas

Vulnerabilidade	Web Scanners	Gravidade
Sql Injection	[1,2,3,5,7,9,10]	Alto
Directory Browsing	[2,3,8,9,10]	Médio
Password Autocomplete in browser	[2,3,10]	Alto
Cross site scripting	[2,3,4,5,7,8,10]	Médio
Lotus Domino default files	[2,8]	Médio
Cross site scripting without brackets	[2,8]	Médio
Obsolete file	[2,8]	Baixo
Private ip disclosure	[2,3,8]	Baixo
Session ID in URL rewrite	[2,8]	Baixo
Obsolete file extended check	[2,8]	Baixo
Remote file inclusion	[3, 9]	Alto
Local file include	[1,3,7,9]	Alto
External Redirect	[3]	Alto
Remote OS Command Injection	[3]	Alto
Path Traversal	[3,4,9]	Alto
Application Error Disclosure	[3,4,10]	Médio
X-Frame-Options Header Not Set	[3]	Médio
Format string error	[3]	Médio
Buffer overflow	[3]	Médio
Web Browser XSS Protection Not Enabled	[3,9]	Baixo
Cookie No HttpOnly Flag	[3,4,6,9,10]	Baixo
X-Content-Type-Options Header Missing	[3]	Baixo
Incorrecting caching directives	[9]	Alto
External content embedded on a page	[9]	Baixo
Unix Operating System Unsupported Version Detection	[4]	Alto
SSH Weak Algorithms Supported	[4]	Médio
Samba Badlock Vulnerability	[4]	Médio
Shell injection	[10]	Alto

Vale também ressaltar que ferramentas como os *scanners* de vulnerabilidades também realizam ataques de DoS contra sistemas. A exemplo, uma dessas ferramentas foi utilizada recentemente para testar a robustez do Moodle da UNIPAMPA. O resultado do ataque foi uma indisponibilidade de aproximadamente 10 minutos do sistema. Somente após o endereço de origem do ataque ser bloqueado nos sistemas de segurança da instituição e o serviço ser re-estabelecido manualmente, o ataque deixou de ter efeito. Claro que, no caso, bastaria o atacante trocar de endereço IP para continuar o ataque.

No exemplo da UNIPAMPA, o problema está relacionado à implementação de um sistema moderno, amplamente utilizado e com vários ciclos evolutivos de melhoramento de *software*. Este tipo de ataque pode ser evitado através de técnicas e recursos de programação simples. Por exemplo, com a linguagem de programação PHP é possível verificar o tempo entre requisições subsequentes de forma simples. Em combinação com mapas do *ModRewrite* do servidor *Web* Apache, o sistema pode gerar, em tempo de execução, uma lista negra de IPs identificados como potenciais origens de ataques. O Apache irá utilizar os mapas para bloquear os ataques de DoS contra o sistema *Web*, o Moodle, no caso. Entretanto, é necessário que os engenheiros de *software* tenham formação apropriada em segurança de *software*, o que ainda não é algo comum, infelizmente.

Tabela 2. Principais resultados e métricas

Ferramenta	Porc. de deteção	Tempo de execução	Detecção gravidade alta
Uniscan	7,14 %	9 horas	20 %
Paros	35,71 %	1 hora e 03 minutos	20 %
Zed Attack	60,71 %	1 hora e 05 minutos	70 %
Nessus	21,43 %	15 minutos	20 %
Ratproxy	7,14 %	12 minutos	10 %
Grabber	3,57 %	5 minutos	0 %
Wapiti	10,71 %	2 horas	20 %
Andiparos	28,57 %	33 minutos	0 %
Skipfish	32,14 %	30 minutos	50 %
Vega	25,00 %	72 horas	30 %

Outra forma de conter ataques deste tipo é através de mecanismos avançados de tolerância a faltas e intrusões. Pesquisas e desenvolvimentos recentes tem mostrado que mecanismos como replicação de máquina de estados e componentes seguros podem ser utilizados para amenizar significativamente os efeitos de ataques de exaustão de recursos, ataques de DoS e intrusões que tenham como objetivo comprometer a confidencialidade e corretude do serviço [Kreutz et al. 2016].

4. Trabalhos Relacionados

Existem diferentes estudos que analisam e comparam *scanners* de vulnerabilidades [Bau et al. 2010, Vieira et al. 2009, Fong et al. 2008]. Estas pesquisas analisam ferramentas comerciais ou são consideravelmente desatualizadas. Em outras palavras, não há estudos recentes que analisam e comparam *scanners* de vulnerabilidades, em especial as opções livres e abertas.

Entre os *scanners* de vulnerabilidades avaliados na literatura, podem ser citados o *HP WebInspect* e o *IBM Rational AppScan*. Estes *scanners* se destacam pela boa cobertura de detecção de vulnerabilidades em cenários controlados.

Algumas das questões que os pesquisadores frequentemente procuram responder incluem: (1) Qual a cobertura dos *scanners* de vulnerabilidades quando usados em um ambiente de serviços *web*?, (2) Qual a taxa de falsos-positivos?, (3) Quais são os tipos mais comuns de vulnerabilidades em ambientes de serviços da *web*? e (4) Quais vulnerabilidades são as mais detectadas pelas ferramentas?. Neste trabalho foi respondido três das quatro questões. A única questão não respondida é a sobre falsos-positivos. A resposta a esta pergunta implicaria em analisar o código fonte de todas as dezenas de aplicações *Web* da máquina virtual BWA. Vale também ressaltar que falsos-positivos não ocorrem com tanta frequência em *scanners* de vulnerabilidades. Além disso, os falsos-positivos não representam um problema, uma vez que é melhor prevenir do que remediar. Em outras palavras, no pior caso falsos-positivos implicam em uma segunda verificação de propriedades, funcionalidades e códigos dos sistemas. Isto apenas ajuda a ratificar a corretude e segurança dos sistemas.

Os cenários mais comuns de avaliação de *scanners* de vulnerabilidades consistem em conjuntos limitados de serviços e sistemas controlados [Vieira et al. 2009]. Esse tipo de sistema é preparado especificamente para testes de segurança. Neste trabalho, foi utilizada a máquina virtual BWA, que possui uma boa coleção de aplicativos *Web* vulneráveis.

Esta máquina faz parte de um dos projetos da OWASP destinada a profissionais e pessoas interessadas em segurança e testes de *software*. Um dos pontos positivos da máquina virtual é que ela possui um conjunto significativo de vulnerabilidades.

Apesar de possuírem o mesmo intuito, os *scanners* detectam diferentes tipos de vulnerabilidades, com exceção da *SQL injection* e *cross site scripting* que, no geral, são detectadas por todas as ferramentas conhecidas. Os trabalhos relacionados detectam e analisam um conjunto limitado (em torno de 78% menos) de vulnerabilidades se comparados a este trabalho. Este cálculo foi baseado pela quantidade de vulnerabilidades detectadas. Enquanto que os trabalhos mencionados anteriormente informam a detecção de 6 vulnerabilidades, neste trabalho foram detectadas 28 vulnerabilidades distintas.

5. Conclusão

Neste trabalho foi analisada e discutida a eficácia de diferentes *scanners* de vulnerabilidades atuais, livres e abertos. Os resultados mostram a importância e a necessidade de se utilizar múltiplas ferramentas para encontrar e corrigir defeitos em sistemas *Web*. Utilizando um ambiente controlado, os *scanners* detectaram inúmeras vulnerabilidades de baixa, média e alta criticidade. É importante ressaltar que as vulnerabilidades de média e alta criticidade representam riscos de segurança críticos eminente para um sistema.

SQL injection e *cross site scripting* são as vulnerabilidades mais frequentemente testadas pelos *scanners*. Apesar disto, e do nível de criticidade, estas vulnerabilidades continuam na lista das mais recorrentes em sistemas *Web*. Este cenário é preocupante. As estatísticas permitem concluir que há uma falta de cuidados e utilização de ferramentas como *scanners* de vulnerabilidades no ciclo de vida de sistemas *online*.

Apesar de possuírem o mesmo intuito, as ferramentas detectam diferentes vulnerabilidades e diferentes níveis de gravidade, como é o caso do Nessus. Apesar do baixo índice de cobertura (21,43%), este *scanner* detectou vulnerabilidades que as outras ferramentas não reconheceram, como *samba badlock vulnerability, ssh weak algorithms supported e unix operating system unsupported version detection*. Outro exemplo é o Skipfish, que, apesar de apresentar uma cobertura de apenas 32,14%, detectou 50% das vulnerabilidades que representam risco de segurança alto.

A ferramenta mais eficaz foi a Zed Attack Proxy, com uma cobertura de 60,71% na detecção das vulnerabilidades da BWA. Entretanto, para atingir uma cobertura 100%, são necessários pelo menos mais quatro *scanners*, o Paros, Nessus, Skipfish e Vega. Estes resultados demonstram a necessidade de utilizar-se várias ferramentas de varredura de vulnerabilidades em sistemas *Web*. Um bom conjunto de *scanners* poderá oferecer uma boa cobertura na detecção de vulnerabilidades, diminuindo significativamente os riscos de comprometer a segurança dos sistemas.

Finalmente, é importante ressaltar que esta pesquisa terá continuidade através de diferentes ações e tarefas, como análise de testes específicos (exemplo: DoS) e utilização de diferentes ambientes controlados (para identificar taxas de falsos positivos e falsos negativos, por exemplo) e não controlados (sistemas disponíveis na *Web*). Outro objetivo futuro consiste em estudar *firewalls* de aplicação, ferramentas que podem ajudar a controlar ataques que visam explorar vulnerabilidades de sistemas *Web*.

Agradecimentos. Em especial ao Gustavo Alves Rodrigues pela contribuição técnica no que diz

respeito a mecanismos de defesa contra ataques de DoS em sistemas Web PHP/Apache.

Referências

Baldessar, M. J. (2014). Noticiário internacional: um mapa de contradições e influências ideológicas e econômicas1.

Bau, J., Bursztein, E., Gupta, D., and Mitchell, J. (2010). State of the art: Automated black-box web application vulnerability testing. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 332–345. IEEE.

Doupé, A., Cova, M., and Vigna, G. (2010). Why johnny can't pentest: An analysis of black-box web vulnerability scanners. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 111–131. Springer.

Fong, E., Gaucher, R., Okun, V., Black, P. E., and Dalci, E. (2008). Building a test suite for web application scanners. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 478–478. IEEE.

Fong, E. and Okun, V. (2007). Web application scanners: definitions and functions. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 280b–280b. IEEE.

INFOSEC (2017). 14 best open source web application vulnerability scanners. `https://goo.gl/DknrqB`.

Kreutz, D., Malichevskyy, O., Feitosa, E., Cunha, H., da Rosa Righi, R., and de Macedo, D. D. (2016). A cyber-resilient architecture for critical security services. *Journal of Network and Computer Applications*, 63(Supplement C):173 – 189.

Mundo Hacker (2017). Ferramentas para scan de vulnerabilidades web. `https://goo.gl/VNVcnP`.

Offensive Security (2017). Kali tools - web applications. `https://goo.gl/QfvN4u`.

OWASP (2016). Broken web applications project. `https://goo.gl/NScjJ`.

OWASP (2017). Top ten 2017 project. `https://goo.gl/snkFmd`.

Perlman, R., Kaufman, C., and Speciner, M. (2016). *Network security: private communication in a public world*. Pearson Education India.

Rocha, D., Kreutz, D., and Turchetti, R. (2012). A free and extensible tool to detect vulnerabilities in web systems. In *Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on*, pages 1–6. IEEE.

Terminal Root (2017). 100 melhores ferramentas open source de segurança. `https://goo.gl/9ksoXR`.

Viegas, J. (2016). Por que o número de ataques virtuais aumentou e como evitá-los. `https://goo.gl/zkNahP`.

Vieira, M., Antunes, N., and Madeira, H. (2009). Using web security scanners to detect vulnerabilities in web services. In *Dependable Systems & Networks, 2009. DSN'09. IEEE/IFIP International Conference on*, pages 566–571. IEEE.

SOFIA: Um Sistema de Suporte para as Inspeções de Software

Thiago C. Krug[1], Juliano P. Pires[2], Vitor Hugo M. dos Santos[2], João Pablo S. da Silva[2]

[1]Unversidade Federal de Santa Maria (UFSM)
Av. Roraima, 1000, Santa Maria, RS, Brasil

[2]Universidade Federal do Pampa (UNIPAMPA)
Av.Tiarajú, 810, Alegrete, RS, Brasil

{thiagockrug, jjulianopires, vitao375, jpabloss}@gmail.com

Abstract. *The inspection of software is a important practice of engineering software to detect preliminary of defects, however, its adoption still faces resistance, because it is difficult to relate the effort with the gain in the quality of the product. To support the process of inspection, it presents in this work the SOFIA, a support system for inspection of software. The inspection of software is a important practice of engineering software to detect preliminary of defects, however, its adoption still faces resistance, because it is difficult to relate the effort with the gain in the quality of the product. To support the process of inspection, it presents in this work the SOFIA, a support system for inspection of software.*

Resumo. *A inspeção de software é uma prática importante de engenharia de software para detecção preliminar de defeitos, porém, a sua adoção ainda enfrenta resistência, pois é difícil relacionar o esforço despendido com o ganho na qualidade do produto. Para dar apoio ao processo de inspeção, apresenta-se neste trabalho o SOFIA, um sistema de suporte para inspeções de software. O SOFIA provê suporte para execução de todo o ciclo de vida de uma inspeção. Além disso, ele possui agentes de software que automatizam as tarefas durante o processo de inspeção, o SOFIA é capaz de apoiar a execução de inspeções de software, indicando potencial para otimizar a realização de inspeções e, por consequência, aumentar a produtividade das equipes.*

1. Introdução

A descoberta precoce de defeitos é uma ótima estratégia para se obter softwares economicamente viáveis [Pressman 2009]. As inspeções de software, introduzidas por *Michael E. Fagan* em 1976 [Fagan 1986], têm se mostrado uma prática muito efetiva para detectar anomalias em projetos de software [Sommerville 2010]. Uma inspeção de software pode ser definida como um processo sistemático para verificar se o software satisfaz suas especificações, exibe os atributos de qualidade especificados, segue as regulamentações, padrões, orientações, planos, especificações e procedimentos, entre outros [IEEE 2008].

Apesar de seus benefícios, a adoção das inspeções de software ainda enfrenta resistência, pois é difícil relacionar o esforço despendido com o ganho na qualidade do produto [da Silva et al. 2013]. O processo de inspeção não é suficientemente adaptado para o contexto do projeto e dos membros das equipes de desenvolvimento atuais. Outro

fator que dificulta a adoção é que o investimento inicial parece ser alto, uma vez que não há ligação entre as atividades da inspeção e as atividades usuais de desenvolvimento [Denger and Shull 2007].

Ferramentas de apoio ao processo de inspeção podem contribuir para a adoção da prática. Tais ferramentas podem minimizar a sobrecarga de controle imposta pela sistematização e integrar as inspeções ao processo de desenvolvimento das equipes. Motivado por isso, apresenta-se neste trabalho o SOFIA, um sistema de suporte para inspeções de software baseado no processo da norma IEEE Std 1028-2008. O SOFIA provê suporte para execução do ciclo de vida de uma inspeção, além de ter agentes de software que automatizam as tarefas de: i) seleção e envio de artefatos para inspeção, ii) organização e distribuição de material para inspeção e iii) monitoramento das correções de anomalias.

O SOFIA é formado por um ambiente *web* desenvolvido em *JavaServer Pages* (JSP) para interação com os usuários, um módulo multiagentes desenvolvido em *Java Agent Development Framework* (JADE) para suporte automatizado na execução das inspeções, uma base de conhecimento desenvolvida em *Web Ontology Language* (OWL) para definir o processo de software e uma base de dados para persistir os resultados das inspeções. Os testes preliminares em ambiente controlado mostraram que o SOFIA é capaz de apoiar a execução de inspeções de software, indicando potencial para otimizar a realização de inspeções e, por consequência, aumentar a produtividade das equipes.

O restante deste trabalho está organizado como segue. Na section 2 é explicado como se deu o desenvolvimento do SOFIA em uma perspectiva de engenharia de software. Por fim, na section 3 são apresentadas as considerações finais e os trabalhos futuros.

2. Desenvolvimento do Sistema

O presente trabalho apresenta o SOFIA, um sistema de suporte para inspeções de software baseado no processo da norma IEEE Std 1028-2008. O SOFIA possui cinco funcionalidades principais: enviar o artefato para a inspeção, reunir os materiais de inspeção, distribuir os materiais de inspeção, possibilitar o inspetor analisar o artefato, e notificar o inspetor líder das correções das anomalias.

2.1. Arquitetura da Solução

Conforme apresentado na Figura 1, o SOFIA é composto por uma base de conhecimento, um ambiente *web* e um banco de dados. O módulo multiagente interage com a base de conhecimento, na qual o conhecimento necessário para auxiliar a tomada de decisões durante o processo de inspeção é computacionalmente representado. Os agentes também interagem com uma base de dados onde são registrados os dados relevantes ao processo. Os dados podem ser apresentados posteriormente pelo ambiente *web* que interage com o usuário, relatando o andamento do processo de inspeção. No momento que o ambiente *web* é inicializado, o módulo multiagente também é iniciado para que ambos possam ser utilizados pelo processo de inspeção em andamento.

O SOFIA foi construído partindo de algumas premissas. Considera-se que o projeto em execução possui um repositório sob o sistema de controle de versão Git[1], contendo

[1]Git é um sistema de controle de versões distribuído feito para manipular desde projetos pequenos até grandes com velocidade e eficiência. http://git-scm.com/.

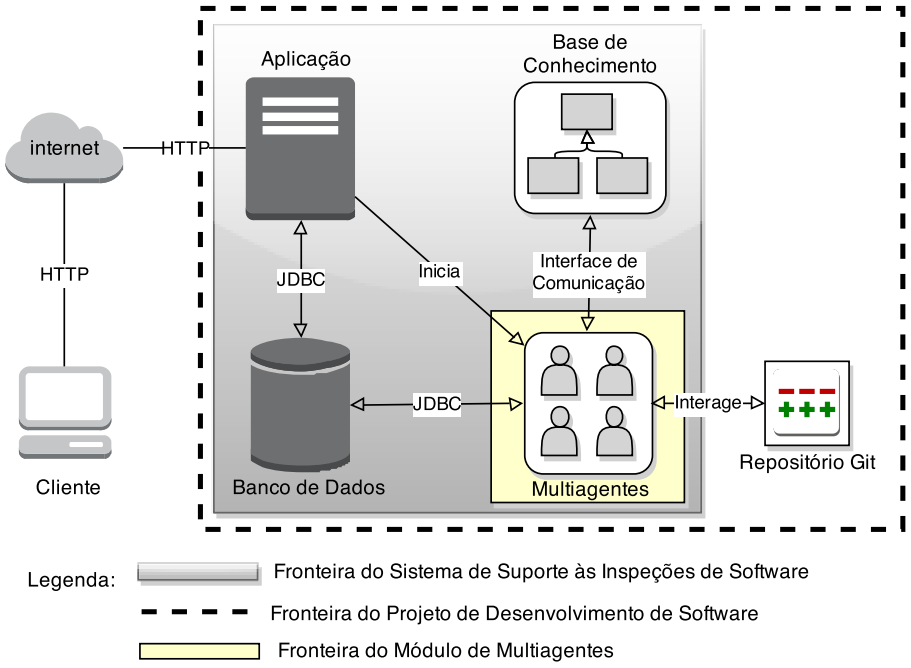


Figura 1. Fronteiras da aplicação.

todos os arquivos do projeto. Esse repositório possui um ramo específico para o envio dos artefatos prontos para a inspeção. O mesmo ramo recebe os envios dos artefatos que são corrigidos por seus autores. Além disso, a base de conhecimento deve ser populada com o conhecimento do processo de desenvolvimento em andamento e seguir as restrições do projeto.

O SOFIA tem por finalidade prover suporte para as atividades do processo de inspeção de software normatizadas pela IEEE 1028-2008, na forma de automatizar fases, etapas ou procedimentos de processos de garantia da qualidade. Na mesma perspectiva, é possível verificar uma tendência na utilização de agentes de software em conjunto com bases de conhecimento ontológicas para o suporte a tais atividades [da Silva et al. 2013] [Lee and Wang 2009] [Wang and Lee 2008]. Entretanto, mesmo em processos novos ou modificados, a necessidade de dispor de uma ferramenta que auxilie no processo é aparente [Mishra and Mishra 2009] [Lucia et al. 2011] [Liu et al. 2012] [Mishra and Mishra 2012]. A utilização de agentes nesse tipo de situação se torna adequada, pois um agente é um sistema computacional que está situado em algum ambiente onde é capaz de realizar ações autônomas neste ambiente para atingir seus objetivos [Wooldridge and Jennings 1995]. Uma definição mais simples é que um agente é qualquer coisa que consegue perceber o ambiente através de sensores e atuar neste ambiente através de atuadores [Russell and Norvig 2009].

A estrutura do módulo multiagente é composta por cinco agentes com responsabilidades distintas, estruturalmente definida como reativa simples. Apesar disso, o módulo multiagente atende a todas as outras propriedades de sistemas multiagentes inteligentes: é *autônomo* quando consulta por dados no ambiente e na base de conhecimento; *pró-ativo*, pois identifica, reúne e distribui os materiais de inspeção; e é *sociável*, pois se relaciona com outros agentes para a obtenção de dados referentes à inspeção.

Para a realização da funcionalidade de enviar o artefato para a inspeção, o autor deve realizar o envio do artefato no ramo das inspeções. Após, o módulo multiagente detecta que esse artefato foi adicionado e consulta a base de conhecimento para descobrir qual o tipo desse artefato. Por fim, o artefato é inserido no banco de dados para que fique disponível para a inspeção.

Outra funcionalidade se dá quando o inspetor líder acessa o ambiente *web*, seleciona os artefatos disponíveis para a inspeção e os membros do time de inspeção, e autoriza a inspeção. Em seguida, o módulo multiagente verifica a adição de uma nova inspeção, e reúne os membros do time e os artefatos selecionados pelo inspetor líder. Conseguinte, o módulo multiagente consulta a base de conhecimento e obtém os itens de revisão pertencentes àquele artefato, realizando dessa forma a funcionalidade de reunir os materiais de inspeção. Por fim, o módulo envia um *e-mail* para cada membro do time informando a nova inspeção e fornecendo um *link* para a página em que se encontra o *checklist*. Assim, distribuindo o material de inspeção e possibilitando os inspetores analisarem o artefato de software.

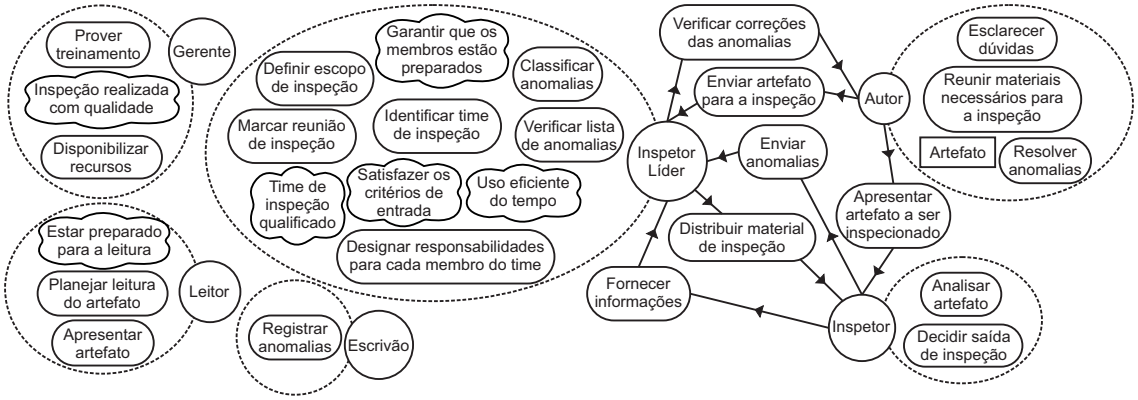


Figura 2. Modelo de dependências entre atores e seus objetivos.

A última funcionalidade trata da verificação das correções das anomalias. Por meio dela, o autor corrige a anomalia encontrada pelos inspetores e realiza o envio dessas correções para o repositório Git. O autor pode informar qual o estado da anomalia que ele está corrigindo na mensagem de envio. Após, o módulo multiagente verifica as mensagens de envio realizadas e encontra a modificação do estado da anomalia. Em seguida, ele encontra e modifica o estado da anomalia de acordo com a mensagem enviada pelo autor, salvando o novo estado da anomalia e enviando um *e-mail* para o inspetor líder.

As funcionalidades descritas pelo módulo multiagente foram selecionadas a partir da análise do ambiente do processo de inspeção de software. Para isso, foi utilizada a metodologia de engenharia de agentes Tropos, a qual possibilitou a obtenção do modelo de dependências entre atores e seus objetivos, como é apresentado na Figura 2.

Tropos é uma metodologia de engenharia de software orientada a agentes que cobre todo o processo de desenvolvimento do software. Seu propósito é a construção refinada e incrementalmente estendida do sistema e do seu ambiente. A metodologia Tropos possui quatro fases de desenvolvimento: (i) *requisitos preliminares*, (ii) *requisitos finais*, (iii) *arquitetura do projeto*, e (iv) *projeto detalhado* [Bresciani et al. 2004]. Elas direcionaram o desenvolvimento desde sua concepção até o projeto dos agentes de software.

2.2. Implementação

A implementação de agentes de software, diferentemente de outros sistemas, requer a utilização de estruturas que possibilitem seu monitoramento e seu controle. Nesse sentido, destaca-se o *framework Java Agent Development* (JADE), que provê uma camada *middleware* de funcionalidades básicas que são independentes das especificação da aplicação e que simplificam o desenvolvimento de aplicações distribuídas que utilizam agentes de software. Além disso, a plataforma JADE provê aos programadores um sistema completamente distribuído habitado por agentes e possui compatibilidade total com as especificações da *Foundation for Intelligent Physical Agents* (FIPA) [Bellifemine et al. 2007].

Cada agente foi implementado seguindo a arquitetura proposta pela JADE, como é apresentado na Figura 3. Os agentes foram divididos por responsabilidades e cada um estende a classe *Agent* pertencente à JADE. O agente *InspectionManager* é equivalente ao agente *Gerente da Inspeção*. Já o agente *ArtifactManager* é o mesmo agente *Gerente dos Artefatos*. Ainda na Figura 3, há o agente *TeamManager*, o qual representa o agente *Gerente do time*. O agente *ScopeManager* representa o agente *Gerente do escopo*. Por fim o agente *AnomalieManager* equivale ao agente *Gerente das anomalias*.

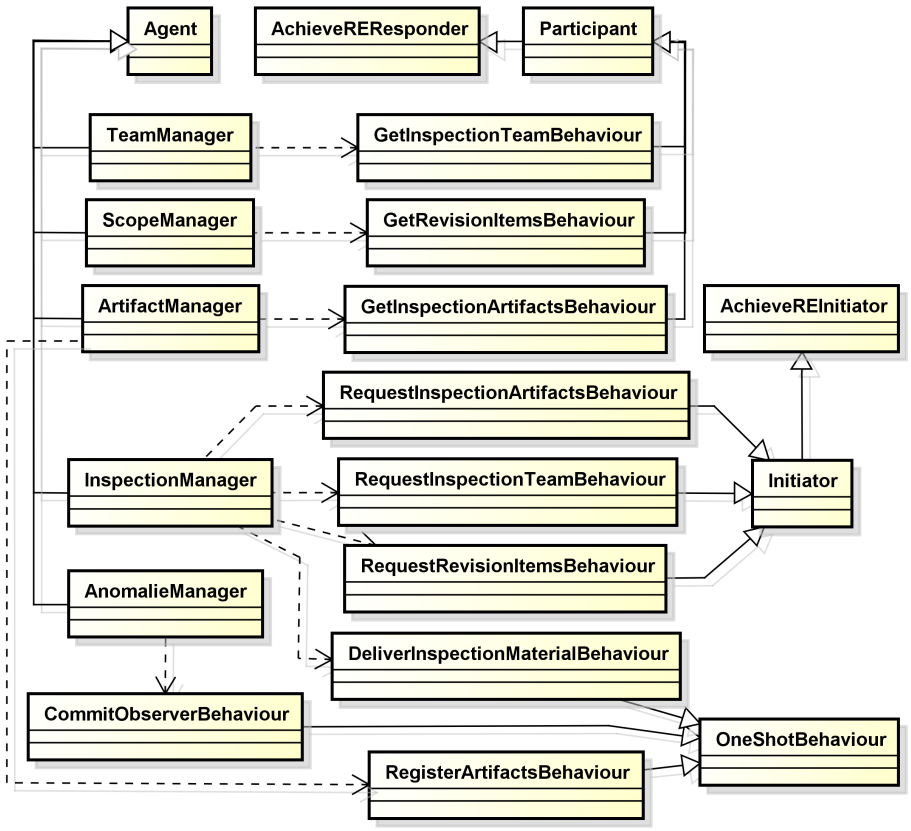


Figura 3. Diagrama de classes dos agentes.

Na Figura 3 são apresentados os comportamentos dos agentes. A classe *OneShotBehaviour*, provida pela JADE, estabelece o comportamento de atomicidade para as demais classes. A classe *CommitObserverBehaviour* implementa o comportamento do agente *Gerente das anomalias* de verificar as correções das anomalias. Já a classe *Re-*

gisterArtifactsBehaviour implementa o comportamento do agente *Gerente dos artefatos* de enviar os artefatos para a inspeção. Por último, a classe *DeliverInspectionMaterial-Behaviour* provê o comportamento de distribuir os materiais da inspeção.

Para que os sensores dos agente possam ser ativados, eles necessitam estar em um ambiente. Os ambientes que os sensores do sistema multiagente observam são o banco de dados e o repositório Git. A cada envio no ramo das inspeções do repositório Git, os agentes *Gerente dos artefatos* e *Gerente das anomalias* são disparados. Porém, nenhum agente atua no ambiente Git, pois é apenas uma forma de visualização que os agentes possuem do ambiente de desenvolvimento. O banco de dados, por sua vez, sofre modificações tanto do módulo multiagente quanto pelo time de inspeção por intermédio do ambiente *web*.

2.3. Testes

Foram realizadas cinco baterias de testes para a verificação do SOFIA. Em cada bateria foram verificadas as unidades de cada classe e a integração entre os componentes dos módulos, além dos testes de comportamento e comunicação entre os agentes.

Para a realização dos casos de teste dos comportamentos dos agentes, foram criados dados no ambiente em que há a sensibilização dos sensores. A partir dos dados no ambiente, os sensores eram disparados, e o agente realizava seu comportamento e atuava modificando o ambiente. Foi verificado se o estado do ambiente após a atuação do agente condizia com o estado esperado.

A comunicação entre os agentes foi testada de maneira semelhante. O ambiente era preparado com dados que disparavam o comportamento sob teste. Após, era inicializada a plataforma JADE juntamente com os agentes que fossem se comunicar. Ao final do teste, foram comparadas as saídas resultantes dos comportamentos dos agentes com o as saídas esperadas. Ao total foram gerados 183 casos de teste.

Os resultados dos casos de teste unitários, integração, comportamento e comunicação entre os agentes foram separados pelas bateria de testes. Em cada bateria foram verificadas a quantidade de falhas encontradas, a quantidade de linhas cobertas e a quantidade de caminhos ou ramos percorridos. Para isso, foi utilizado o Cobertura[2] em conjunto com o JUnit.

O gráfico apresentado na Figura 4 é referente a quantidade total de falhas encontradas durante cada bateria de testes unitários, integração, comportamento e comunicação entre os agentes. Pode-se verificar que houve uma redução das falhas durante as duas primeiras baterias de testes, não havendo nenhuma falha a partir da terceira bateria.

Na Figura 5 é apresentada a percentagem da cobertura das linhas e da cobertura dos ramos por bateria de testes. Houve um aumento da cobertura durante as três primeiras baterias de teste, uma redução na quarta bateria, e um novo aumentando na quinta bateria. Ao final, se chegou a um total de 86% de cobertura de linhas de código-fonte e a 82% de cobertura de ramos.

Ao analisar a Figura 4 e a Figura 5, pode-se concluir que apesar dos casos de teste cobrirem 86% do código-fonte, esses 86% estão corretos de acordo com os resultados

[2]Ferramenta para teste de cobertura de código-fonte em Java. http://cobertura.github.io/cobertura/.

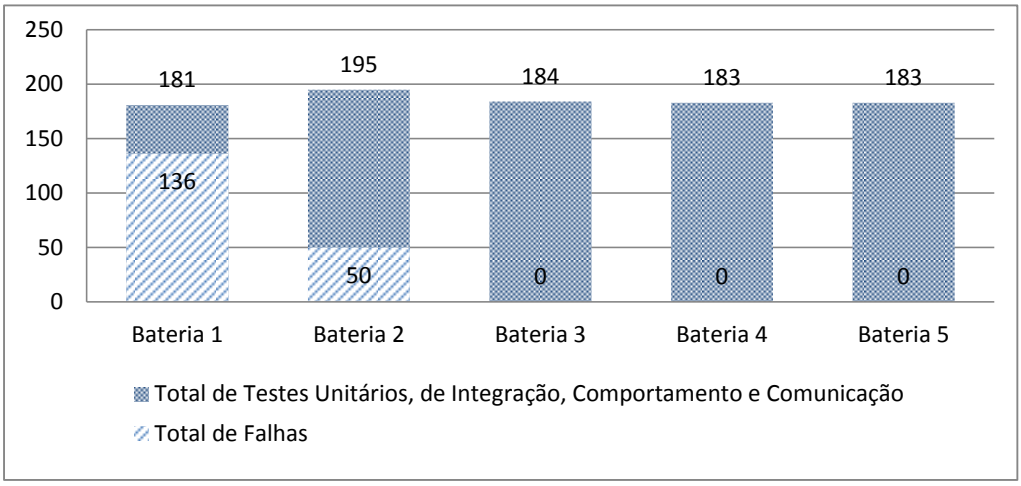


Figura 4. Gráfico de Quantidade de Falhas por Bateria de Teste.

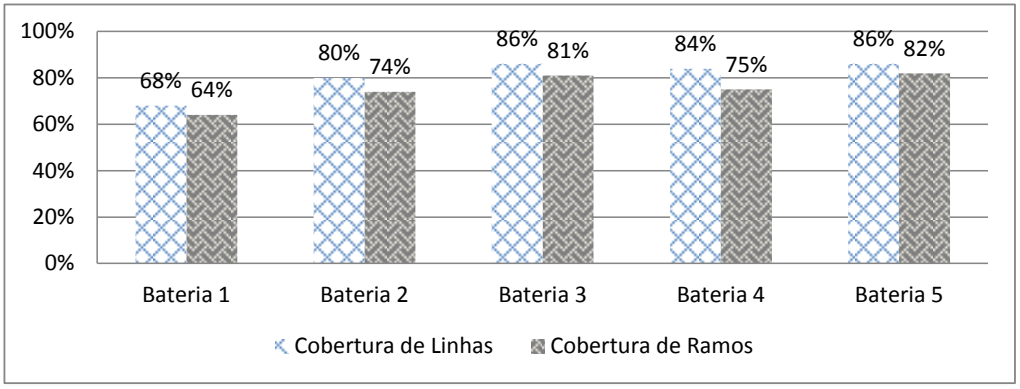


Figura 5. Gráfico de Cobertura do Código por Bateria de Teste.

esperados. Fazem parte dos 86% todos os fluxos principais e secundários do SOFIA.

3. Conclusão

O uso de ferramentas de suporte pode melhorar a relação custo/benefício nas inspeções de software. Neste sentido, apresentou-se neste trabalho o SOFIA, um sistema de suporte para inspeções de software baseado no processo da norma IEEE Std 1028-2008. O SOFIA oferece suporte a inspeções, utilizando agentes para selecionar, organizar e distribuir materiais e para monitorar correções de anomalias.

Nos testes foi possivel verificar que o sistema atende a todas a funcionalidades especificadas, mostrando assim que o sistema está apto para experimentos em um contexto real, para que o mesmo demonstre o quão útil é no contexto de inspeção de software.

A ferramenta foi aplicada somente em ambiente controlado demonstrando resultados positivos. Entretanto, em contextos reais pode ser possível encontrar adversidades não exploradas. Assim, dentre os trabalhos futuros está a aplicação da ferramenta em um contexto real e evolução das funcionalidades que apresentarem menor eficiência.

Referências

[Bellifemine et al. 2007] Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons.

[Bresciani et al. 2004] 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.

[da Silva et al. 2013] da Silva, J. P. S., Dall'Oglio, P., da Silva Pinto, S. C. C., and Bittencourt, I. I. (2013). Um sistema para inspeções de garantia da qualidade baseado em ontologias e agentes. *Revista de Informática Teórica e Aplicada*, VIII(1):1–18.

[Denger and Shull 2007] Denger, C. and Shull, F. (2007). A practical approach for quality-driven inspections. *IEEE Software*, 24(2):79–86.

[Fagan 1986] Fagan, M. E. (1986). Advances in software inspections. *IEEE Transactions on Software Engineering*, 12(7):744–751.

[IEEE 2008] IEEE, I. C. S. (2008). *IEEE 1028: Standard for Software Reviews and Audits*.

[Lee and Wang 2009] Lee, C.-S. and Wang, M.-H. (2009). Ontology-based computational intelligent multi-agent and its application to cmmi assessment. *Applied Intelligence*, 30(3):203–219.

[Liu et al. 2012] Liu, S., Chen, Y., Nagoya, F., and McDermid, J. A. (2012). Formal specification-based inspection for verification of programs. *IEEE Transactions on Software Engineering*, 38(5):1100–1122.

[Lucia et al. 2011] Lucia, A. D., Fasano, F., Scanniello, G., and Tortora, G. (2011). Improving artefact quality management in advanced artefact management system with distributed inspection. *IET Software*, 5(6):510–527.

[Mishra and Mishra 2009] Mishra, D. and Mishra, A. (2009). Simplified software inspection process in compliance with international standards. *Computer Standards & Interfaces*, 31(4):763–771.

[Mishra and Mishra 2012] Mishra, D. and Mishra, A. (2012). A global software inspection process for distributed software development. *Journal UCS*, 18(19):2731–2746.

[Pressman 2009] Pressman, R. S. (2009). *Software Engineering: A Proactitioner's Approach*. McGraw-Hill, New York, NY, USA, 7 edition.

[Russell and Norvig 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[Sommerville 2010] Sommerville, I. (2010). *Softwware Engineering*. Addison-Wesley, Boston, MA, USA, 9 edition.

[Wang and Lee 2008] Wang, M.-H. and Lee, C.-S. (2008). An intelligent ppqa web services for cmmi assessment. In *International Conference on Intelligent Systems Design and Applications*, volume 1, pages 229–234.

[Wooldridge and Jennings 1995] Wooldridge, M. and Jennings, N. R. (1995). Agent theories, architectures, and languages: a survey. In *Intelligent agents*, pages 1–39. Springer Berlin Heidelberg, Berlin.

Redes de Colaboração em Ecossistemas de Software:
Uma Análise baseada em uma Revisão Longitudinal de Literatura

Caio Borges[1], David Barreto[2], Sabrina Marczak[1], Cleidson de Souza[3], Fernando Figueira Filho[4]

[1]MunDDoS – PPGCC – Faculdade de Informática, PUCRS
Caixa Postal 1.429 – 90619-900 – Porto Alegre – RS

[2]Pesquisador Independente – Porto Alegre – RS – Brasil

[3]Faculdade de Computação, UFPA
Rua Augusto Corrêa 01 – CEP 66075-110 – Belém – PA

[4]Coopera – Departamento de Informática e Matemática Aplicada, UFRN
Caixa Postal 1524 – Natal – RN

caio.borges@acad.pucrs.br, david@cgbhome.com,sabrina.marczak@pucrs.br,
cleidson.desouza@acm.org, fernando@dimap.ufrn.br

Abstract. *A Software Ecosystem is the interaction of a set of actors having a technological platform in common and resulting in a number of software solutions or services. The development of this kind of large software system involves business, technical, and social aspects. Manikas (2016)'s literature review revealed that although software ecosystems are gaining importance, the topic is still in development and that some aspects are less discussed than others, such as the social one. In order to highlight the collaborations among authors, including the ones that discuss social aspect, this papers presents the visualization of collaborative networks among authors of software ecosystems papers as listed in Manikas's literature review. Results of such visualizations reveal hidden patterns of collaboration and point out central authors. It also highlights the little attention given to social aspects until now.*

Resumo. *Ecossistema de Software é a interação de um conjunto de atores com base em uma plataforma tecnológica comum que resulta em uma série de soluções ou serviços de software. O desenvolvimento deste tipo de grandes sistemas de software envolve aspectos de negócio, técnicos e sociais. A revisão da literatura de Manikas (2016) revelou que, embora os ecossistemas de software ganhem importância, o tema ainda está em desenvolvimento e alguns aspectos são menos discutidos do que outros, como o social. Para destacar as colaborações entre os autores, incluindo os que discutem o aspecto social, esse trabalho apresenta a visualização de redes colaborativas entre autores de artigos de ecossistemas de software, conforme listado na revisão da literatura de Manikas. Os resultados dessas visualizações revelam padrões ocultos de colaboração e apontam quem são os principais autores. Também destaca a pouca atenção dada aos aspectos sociais até os dias de hoje.*

1. Introdução

Jansen *et al.* (2009) definem que um Ecossistema de Software (do inglês, *Software Ecosystem*) é um conjunto de funcionalidades de negócio que como uma unidade interagem em um mercado de compartilhamento de produtos e serviços. Segundo Manikas *et al.* (2013), é uma interação de um conjunto de atores que estão sobre a mesma plataforma tecnológica, o que resulta em um número de software, soluções e serviços. De acordo com Bosch (2009), um Ecossistema de Software consiste em um conjunto de soluções de software que permite, suporta e automatiza atividades e transações de atores associados em Ecossistemas sociais ou de negócios. A definição que permite um panorama mais realista e geral poderia ser a de Bosch *et al.* (2010), que indica que um Ecossistema de Software consiste em uma plataforma de software, um conjunto interno e externo de desenvolvedores e comunidade de domínio que se dedica em compor soluções que satisfaçam suas necessidades.

Essas definições estão todas corretas acerca do tema, porém as primeiras referem-se a dimensões específicas do que realmente é um Ecossistema de Software (ECOS) e como ele se estrutura. No trabalho apresentado por de Souza *et al.* (2016), os autores se baseiam em uma classificação proposta por Barbosa *et al.* (2013) para discutir ECOS em três dimensões, quais sejam: Técnica (*Technical*), de Negócio (*Business*) e Social (*Social* ou *Relationships*). Campbell *et al.* (2010) também classificam em seu trabalho três dimensões—Arquitetura, de Negócio e Social, e propõem como aprimorar cada aspecto em um ECOS com um conjunto de estratégias.

A dimensão *técnica*, conforme definida por Bosch (2009), diz respeito a um conjunto de soluções de software que suporta e automatiza atividades. Sem dúvidas o aspecto técnico é importante e de acordo com Manikas (2016) já possui um considerável número de publicações. A dimensão *de negócio* está constantemente sendo discutida, pois ela foca na produtividade e valor que se pode gerar a partir de um ECOS. Segundo Jansen *et al.* (2009), esta dimensão diz respeito a discussão de funcionalidades de negócio e a interação das mesmas com o mercado de produtos e serviços. A terceira dimensão é a social, também chamada por alguns autores de relacionamento (*relationship*). Ao definir que um ECOS é um conjunto de autores atuando em uma mesma plataforma e que isso resulta em soluções e serviços, Manikas *et al.* (2013) indicam o quanto o social é importante dentro destes ECOS. São as pessoas que produzem as soluções e interagem nesses meios. Manikas (2016) aponta que esta dimensão é pouco discutida em relação às demais.

Visto que a área é recente e que a literatura ainda está sendo consolidada, este trabalho tem como objetivo *identificar através da visualização gráfica as redes de colaboração dos autores da área, destacando em especial os autores que contribuem para as discussões sobre o aspecto social*. Desta forma, definiu-se as seguintes questões de pesquisa:

QP1: Qual a rede de colaboração entre autores em ECOS?

QP2: Quais são os autores que mais têm contribuído em ECOS?

QP3: Qual a rede de colaboração entre autores em aspectos sociais de ECOS?

QP4: Quais são os autores que mais têm contribuído em aspectos sociais de ECOS?

As duas primeiras questões vem com o objetivo de identificar como está a situação geral da área de ECOS, assim sendo, quais autores mais contribuem com a área e como esses autores têm se articulado com os demais tendo em vista a colaboração em suas publicações. As duas questões seguinte são referentes exclusivamente ao foco social em ECOS, pois o foco de nosso trabalho será investigar um mais de perto essas relações dentro dos aspectos sociais. Isso servirá na identificação dos autores mais produtivos e mais ativos da área.

2. Aspectos Sociais em Ecossistemas de Software

O aspecto social é uma das três dimensões de um ECOS, mas é fundamental entender a sua importância. Manikas *et al.* (2013) relatam que um ECOS não se faz apenas com processos e modelos de negócio, mas sim que necessita do aspecto social. Uma comunidade que utiliza uma plataforma precisa interagir entre si e com a plataforma.

De Souza *et al.* (2016) apontam que sustentabilidade é o principal aspecto de um ECOS e que isso só se torna possível quando se possui um conjunto de desenvolvedores ativos e comprometidos com a plataforma, o que denota a importância dos aspectos sociais nesses ambientes. Mens *et al.* (2011), em sua pesquisa sobre a evolução dos aspectos sociais em ECOS *Open Source*, concluem que os aspectos sociais têm impacto significante neste tipo de sistema e que se deve estudar a comunidade e suas influências sobre os ECOS em si. Sobre as comunidades em ECOS, de Souza *et al.* (2016) indicam que a obtenção e manutenção de novos membros estão entre os maiores desafios.

Em síntese, os aspectos sociais não são apenas uma dimensão importante mas um pilar fundamental na concepção de um ECOS e sua sustentabilidade e evolução.

3. Metodologia para a Elaboração das Redes de Colaboração dos Autores

Optou-se por utilizar o trabalho de Manikas (2016) como base para este estudo por ser a revisão de literatura mais recente sobre o tema. Neste trabalho, o autor identifica 231 artigos sobre ECOS, publicados em conferências e periódicos entre 2007 e 2014, e os classifica de diversas formas (*e.g.*, por aspecto, por evento) ao longo de seu trabalho. Sendo assim, para identificar as redes de colaborações entre os autores dos artigos citados por Manikas (2016), realizou-se o levantamento e a visualização destes autores usando técnicas de análise de redes sociais (do inglês, *Social Network Analysis* – SNA) conforme detalhado nesta seção.

Uma rede de colaboração identifica um conjunto de atores e algum tipo de interação (e.g., comunicação, confiança, co-autoria) entre eles. Em geral, este tipo de rede é representada por um grafo, onde os atores são representados pelos nodos e a relação entre eles por arcos (ou arestas). No presente trabalho, uma rede de colaboração indica os autores que colaboraram entre si na escrita de artigos sobre o tema de ECOS. Assim, para todos os artigos, sendo cada um composto de *n* autores, cada autor é representado por um nodo e para cada autor (e.g., A_x, A_y, A_z) tem um arco não direcional deste autor com todos os *n* autores que colaboraram com ele na produção do referido artigo (ou demais artigos que este autor faça parte).

Essa técnica de produzir uma visualização de redes de colaboração foi utilizada também por demais autores, como por exemplo das Dores *et. al.* (2014), como parte da sua revisão de literatura sobre estimativa de esforço em projetos de desenvolvimento de

Tabela 1. Exemplo da Preparação dos Dados dos 231 Artigos Base

Código	Autores	Título	Ano	Local Publicado
1	Aaltonen, T., Mikkonen, T., Peltola, H., Salminen, A.	From mashup applications to open data ecosystems	2014	Proceedings of The International Symposium on Open Collaboration. OpenSym '14. ACM, New York, NY, USA, pp. 15:1–15:8 .
2	Aarnoutse, F., Renes, C., Snijders, R., Jansen, S.	The reality of an associate model: comparing partner activity in the eclipse ecosystem	2014	Proceedings of the 2014 European Conference on Software Architecture Workshops. ECSAW '14. ACM, New York, NY, USA, pp. 8:1–8:6 .
3	Albert, B. , Santos, R. , Werner, C.	Software ecosystems governance to enable it architecture based on software asset management	2013	Proceedings of the 7th IEEE International Conference on Digital Ecosystems and Technologies (DEST), 2013, pp. 55–60 .
...	...	...	...	...
N	**Autores**	**Título N**	**Ano**	**Congresso N**

Tabela 2. Autores de cada artigo

Código Artigo - Autores						
1	Aaltonen, T.	**2**	Aarnoutse, F.	**3**	Albert, B.	**N**	Autor N
	Mikkonen, T.		Renes, C.		Santos, R.		Autor N
	Peltola, H.		Snijders, R.		Werner, C.		Autor N
	Salminen, A.		Jansen, S.				Autor N

Tabela 3. Classificação dos Autores

Grupo A	ID Artigos	Grupo B	ID Artigo	Grupo C	ID Artigo
Aaltonen, T.	[1,114]	Peltola, H.	[1]	An, H.	[8]
Alspaugh, T.	[4,187]	Aarnoutse, F.	[2]	Andresen, K.	[9]
Alves, A.M.	[5,6,207]	Renes, C.	[2]	Brockmann, C.	[9]
...	**[...]**	...	**[...]**	...	**[...]**
N	N	N	N	N	N

Tabela 4. Tabela de publicações entre dois autores

	Aaltonen, T.	Alspaugh, T.	Alves, A.M.	...	Autor X
Aaltonen, T.	0	0	0	...	X
Alspaugh, T.	0	0	0	...	X
Alves, A.M.	0	0	0	...	X
...	...	...	...	...	X
Autor Y	X	X	X	X	X

software, trabalho do qual nos inspirou a realizarmos visualizações com a mesma técnica. Outro trabalho que também utilizou essa técnica foi o de Durelli *et al.*(2011), no qual foi realizado um mapeamento sistemático dentro da área de teste de software, utilizando em uma seção de seu artigo uma rede de colaboração para identificar co-autorias e relações entre autores (identificando grupos de autores).

Para a criação e visualização das redes de colaboração dos autores de ECOS segundo os artigos listados no trabalho de Manikas (2016), realizou-se três grandes passos seguindo o procedimento proposto por das Dores *et al.* (2014), sendo eles: 1) preparou-se os dados referentes aos artigos em tabelas Excel (Seção 3.1); 2) adaptou-se o código fonte escrito por das Dores *et al.* (2014) para a geração do *script* de criação das redes usando a ferramenta R (Seção 3.2); e gerou-se as visualizações das redes (Seção 4 – os resultados em si).

3.1. O Processo de Elaboração das Tabelas de Colaboração dos Autores

Quanto a preparação dos dados, realizou-se as seguintes atividades:

- Atividade 1: Baixou-se as referências para cada um dos 231 artigos citados por Manikas (2016).
- Atividade 2: Para cada uma das referências, criou-se uma nova linha em uma tabela Excel denominada "Artigos Extraídos" contendo as seguintes informações: ID do artigo (código), autores, título do artigo, ano de publicação e evento onde foi publicado (local), conforme ilustrado na Tabela 1.
- Atividade 3: Após, foi construída uma nova tabela contendo o ID (1-231) de cada artigo e os seus respectivos autores conforme a Tabela 2. Esta separação foi feita para ser possível visualizar em cada artigo a colaboração existente entre os autores.
- Atividade 4: Na sequência, foi criada uma terceira tabela denominada "Classificação dos Autores" (vide Tabela 3) com o objetivo de identificar quais autores são os mais significativos, buscando a quantidade de publicações de artigos e interligação entre os autores. Essa tabela agrupa autores e o ID dos artigos que publicaram, conforme segue, dividindo-os em três grupos:
    - Grupo A) os autores entre os que publicaram mais de um artigo;
    - Grupo B) os autores que publicaram apenas um artigo, mas em conjunto algum autor do grupo A;
    - Grupo C) os autores que publicaram apenas um artigo, mas não em conjunto algum autor do grupo A ou B.
- Atividade 5: Por fim, gerou-se uma matriz de co-autoria (vide Tabela 4) na qual se representa o volume de co-autoria entre autores. Ou seja, a cada dois autores A_x e A_y, na célula de intersecção entre os mesmos é indicado a quantidade de artigos em comum. Essa tabela serviu como entrada para o software que está descrito na Seção 3.2, o qual gerou a visualização das redes de colaboração.

Com esse procedimento foi possível gerar uma tabela organizada de entrada para o próximo passo que demanda uma matriz para desenhar as relações entre os autores.

3.2. Utilizando a ferramenta R para criação das redes de colaboração

A ferramenta R[1], que tem como propósito ser uma ferramenta de apoio a Computação Estatística, funciona através de um código formulado em sua linguagem pré estruturada que recebe um arquivo ".csv" como a matriz apresentada na Tabela 4 e permite operação sobre esses dados a fim de gerar nossa visualização em rede.

Para gerar uma SNA dos autores e suas colaborações entre si foi utilizado esta ferramenta, que através do código descrito na Figura 1 recebe como entrada um arquivo ".csv", que no caso foi a matriz expressada na Tabela 4.

Com essa matriz e código gerou-se as visualizações disposta na Seção 4.

4. Resultados

QP1: Qual a rede de colaboração em ECOS?

[1] Disponível no site: https://www.r-project.org/ - Acessado em: 21/06/2017

```
> install.packages("igraph")
> library(igraph)
> dados<-read.delim ("C:.../... .csv", row.names = 1, sep = ";")
> dados<-as.matrix(dados)
> g<-graph.adjacency (dados,mode="undirected")
> V(g)$label <- V(g)$name
> tkplot(g, layout = layout.fruchterman.reingold,vertex.color="lightblue")
```

Figura 1. Código usado para gerar SNA (Baseado no código utilizado por das Dores *et al.* (2014))

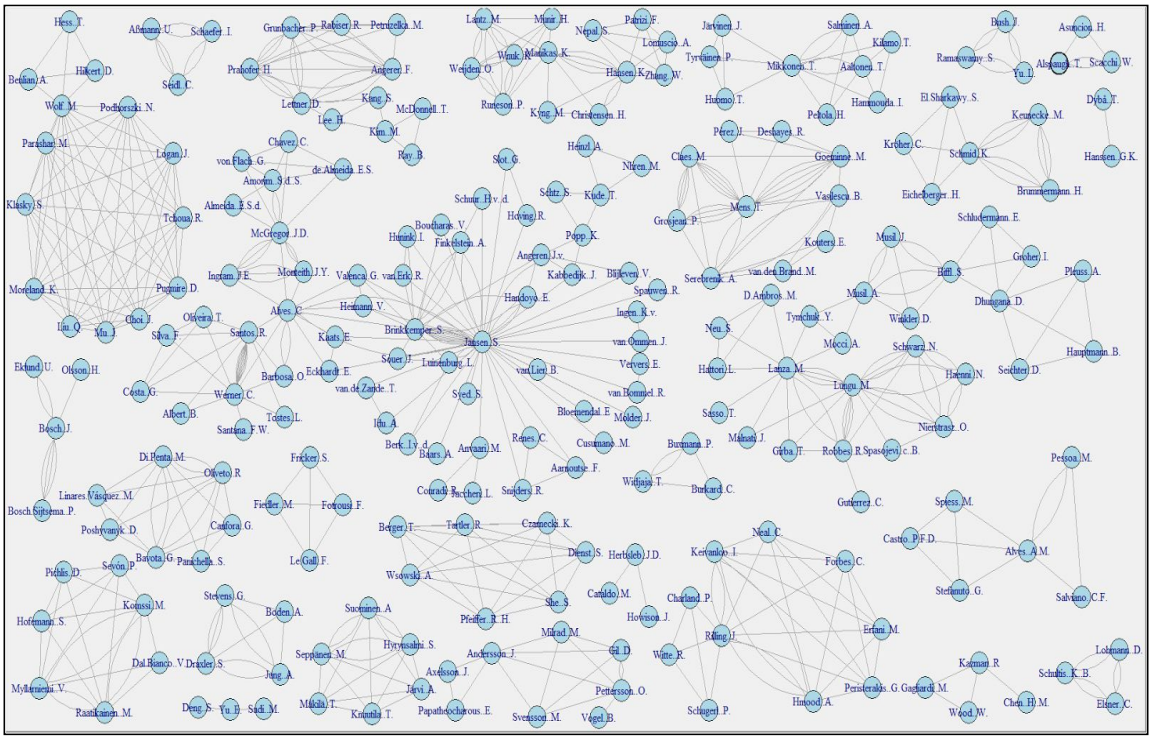


Figura 2. Relação entre os Autores mais Relevantes em Ecossistemas de Software

Pelo processo realizado na Seção 3, a primeira visualização foi realizada para termos um panorama geral da área, mostrando apenas os autores dos grupos A e B explicitados na Seção 3.1 (Figura 2).

QP2: Quais são os autores que mais tem contribuído em ECOS?

Devido a visualização realizada na Figura 2, é possível identificar que, em um panorama geral, o autor Jansen S. é o autor com maior número de publicações sobre o tema de Ecossistemas de Software, possuindo 28 publicações, mais que o dobro do segundo colocado, Bosch J. que possui 11 publicações.

Observando a redes de colaboração gerada é possível identificar alguns padrões em aspecto geral, exemplos são Jansen S. e Brinkkemper S. que possuem 10 publicações entre si. Outros autores extremamente produtivos são Santos R. e Werner C. que possuem 9 publicações entre si. Alguns autores que merecem destaque são Mens T. (9 publicações), Geominne M.(6 publicações), Claes M.(5 publicações) e Grosjean P.(4 publicações) (Figura 3a). Outro autor que merece destaque é Lungu M., que possui 10 publicações (Figura 3b).

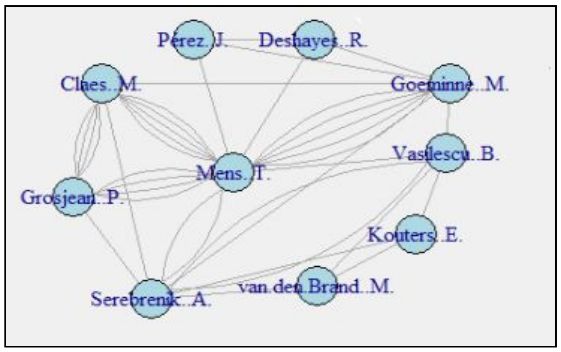


Figura 3a. Mens, T e sua rede de publicações.

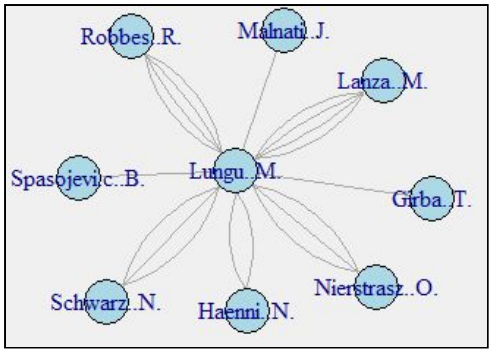


Figura 3b. Lungu M. e autores que colaboraram com o mesmo

Tabela 5. Autores e Número de Publicações (Geral)

Posição - Quantidade de Artigos - Autores				
1º - 28	Jansen, S.			
2º - 11	Bosch, J.			
3º - 10	Brinkkemper, S.	Lungu, M.	Santos, R.	Werner, C.

Na Tabela 5 está disposta uma classificação com os autores mais produtivos em ECOS identificados por Manikas (2016) e a quantidade de suas publicações. Agora que foi realizada as visualizações sobre a situação geral de ECOS, podemos ao invés de utilizar os 231 artigos de ECOS, apenas os 64 que retratam aspectos sociais, com isso, repetir o mesmo processo para responder as próximas questões.

QP3: Qual a rede de Colaboração em aspectos sociais dos ECOS?

Pelo processo realizado na Seção 3, a segunda visualização gerada foi realizada para termos um panorama social, mostrando apenas os autores que tenham contribuído com os aspectos Sociais em ECOS (Figura 4).

QP4: Quais são os autores que mais tem contribuído nos aspectos sociais dos ECOS?

Quanto à classificação de autores que retratam aspectos sociais, a partir da Figura 4 é possível visualizar que: o autor Jansen S. continua sendo o autor com maior número de publicações, possuindo 11 publicações na área, novamente mais que o dobro dos segundos colocados, Brinkkemper S. e Hansen K., que possuem 4 publicações. Na Tabela 6 estão dispostos os primeiros colocados na classificação por publicações em aspectos sociais em Ecossistemas de Software.

Os demais autores foram identificados, porém possuem menos publicações do que os reportados na Tabela 6.

4.1. Resultados Complementares

Nosso trabalho apresenta também dados interessantes para área, que não diretamente vinculados às perguntas de pesquisa mas que melhoram o entendimento do tema. Alguns desses resultados no panorama geral são:

- Comparando a classificação do número de publicações geral com a classificação das publicações em aspectos sociais, pode-se perceber que alguns autores desaparecem no segundo, como por exemplo Bosch J. Isso ocorre porque este autor tem como foco em suas publicações a dimensão de negócio.

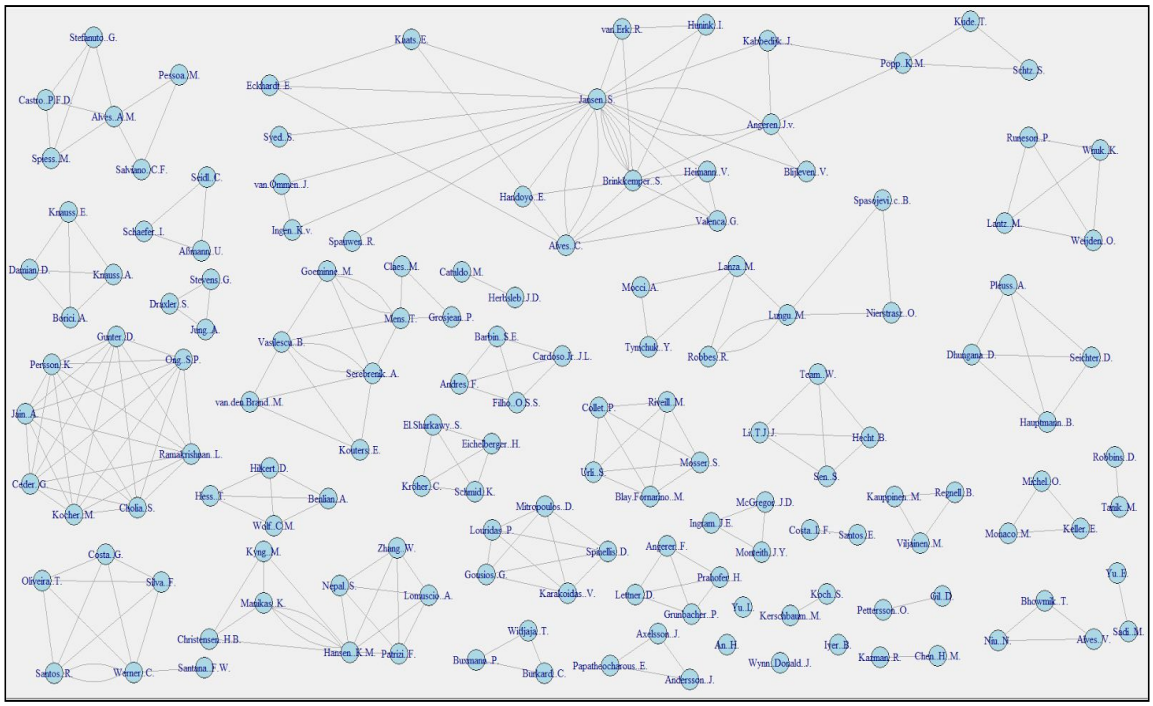


Figura 4. Relação entre os Autores em Aspectos Sociais de Ecossistemas de Software

Tabela 6. Autores e número de Publicações em Aspectos Sociais

Posição - Quantidade de Artigos Sociais - Autores					
1º - 11	Jansen, S.				
2º - 4	Brinkkemper, S.		Hansen, K.		
3º - 3	Angeren, J.v.	Goeminne, M.	Lungu, M.	Manikas, K.	Mens, T.
		Popp, K.		Werner, C.	

- Dos 420 autores das 231 publicações observadas, apenas 26 escreveram sobre os três aspectos de ECOS (Tabela 7).

Alguns resultados complementares em aspectos sociais em ECOS:

- Apesar da área de ECOS ter suas primeiras publicações em 2007, foi apenas em 2009 que houve a primeira publicação voltada a aspectos sociais.
- Jansen, S. continua sendo o pesquisador com maior número de publicações, tendo aproximadamente 39,29% de suas publicações dentro dos aspectos sociais;
- Dos 420 autores das 231 publicações em ECOS, apenas 150 publicaram sobre aspectos sociais (aproximadamente 35,71%), tendo escrito 64 publicações destas 231 (aproximadamente 27,70%).
- Destas 64 publicações de aspectos sociais em ECOS são dos anos de 2009 a 2014, tendo sua maior quantidade em 2014, a quantidade de publicações em aspectos sociais por ano está disposta na Tabela 8.
- Dentro dos aspectos sociais, apenas 20 autores dos 150 escreveram mais de uma publicação sobre aspectos sociais (aproximadamente 13,33%);

Tabela 7. Autores que publicaram nas três dimensões

Autores com mais de uma publicação que possuem contribuições nos três aspectos					
Alves, A.M.	Andersson, J.	Angerer, F.	Brinkkemper, S.	Christensen, H.	Claes, M.
Goeminne, M.	Grosjean, P.	Grunbacher, P.	Hansen, K.	Herbsleb, J.D.	Jansen, S.
Lanza, M.	Lettner, D.	Lungu, M.	Manikas, K.	McGregor, J.D.	Mens, T.
Nierstrasz, O.	Popp, K.	Prahofer, H.	Robbes, R.	Santos, R.	Schmid, K.
Valenca, G.	Werner, C.				

Tabela 8. Publicações Sociais por Ano

Ano	2009	2010	2011	2012	2013	2014
Qt Publicações	1	10	9	6	15	23

- Dois autores possuem mais de uma publicação e todas sobre os aspectos sociais em ECOS, são eles: Kabbedijk, J. e Vasilescu, B.

5. Discussão e Considerações Finais

Ao longo do tempo o número publicações na área de Ecossistemas de Software tem crescido, o que indica que mais pessoas estão envolvidas. Neste trabalho foram identificado diversos autores que têm contribuído com a área bem como as relações entre eles. Através deste trabalho, visualizou-se as redes de colaboração entre os autores da área visando identificar quais autores colaboram com quais e quem são os autores de destaque, ou seja, os que mais contribuem com a escrita de publicações. Deu-se especial atenção aos autores que contribuem com a discussão de aspectos sociais por ser um aspecto pouco explorado se comparado aos demais.

Nosso estudo revelou que apesar de muitos autores dentro de ECOS, poucos deles possuem mais de uma publicação, e poucos deles publicaram também dentro de áreas sociais (como descrito nos resultados na Seção 4). Também se observou que as redes auxiliaram muito na identificação desses autores, por ser visual, basta observar aos autores com as maiores concentrações de arcos, que com isso é possível identificar quantas publicações tem com os demais.

Em suma, neste trabalho foi possível:

- Identificar a quantidade de artigos sobre o assunto ECOS e seus autores;
- Identificar quantidade de artigos sobre o aspecto social em ECOS e seus autores;
- Identificar os principais autores com publicações, num panorama geral e na dimensão Aspectos Sociais.

Como todo trabalho de pesquisa, este também possui limitações. Optou-se for realizar a análise das redes de colaboração baseados na revisão de literatura de Manikas (2016). Os trabalhos publicados depois desta data não são contemplados. Como trabalho futuro poder-se-ia estender a revisão de Manikas e adicionar eventuais novos artigos publicados. Ainda, no estudo presente apenas visualizou-se as redes de colaboração e destacou-se os autores que mais contribuíram com a escrita de artigos. Ao se aplicar métricas de redes sociais, pode-se então avaliar efetivamente o nível de colaboração entre os autores. Acredita-se que as visualizações apresentadas ao longo deste trabalho são contribuições para melhor visualização do panorama atual da área e com eles é possível responder as perguntas levantadas no início da pesquisa.

Agradecimentos

Os autores agradecem ao CNPq (processos 310468/2014-0 e 420801/2016-2) pelo apoio financeiro.

Referências

Barbosa, O., Santos, R. P., Alves, C., Werner, C., & Jansen, S. (2013). Software ecosystems–analyzing and managing business networks in the software industry. Jansen *et al.*[29], Chapter: A Systematic Mapping Study on Software Ecosystems from a Three-Dimensional Perspective, 59-81.

Bosch J. (2009). From software product lines to software ecosystems. In Proceedings of the 13th International Software Product Line Conference (SPLC '09). Carnegie Mellon University, Pittsburgh, PA, USA, 111-119.

Bosch, J., & Bosch-Sijtsema, P. (2010). From integration to composition: On the impact of software product lines, global development and ecosystems. Journal of Systems and Software, 83(1), 67-76.

Campbell, P. R., & Ahmed, F. (2010, August). A three-dimensional view of software ecosystems. In Proceedings of the European Conference on Software Architecture: Companion Volume (ECSA '10), Carlos E. Cuesta (Ed.). ACM, New York, NY, USA, 81-84

das Dores, S. C. N., Ruiz, D. D. A.(2014). Soluções para realização de estimativa de esforço em projetos de desenvolvimento de software: uma revisão sistemática. 2014. Programa de Pós-Graduação em Ciências da Computação, PUCRS, 2014.

Durelli, V. H., Araujo, R. F., Silva, M. A., Oliveira, R. A., & Maldonado, J. C. (2011, September). What a Long, Strange Trip It's Been: Past, Present, and Future Perspectives on Software Testing Research. In Software Engineering (SBES), 2011 25th Brazilian Symposium on (pp. 30-39). IEEE.

Jansen, S., Finkelstein, A., & Brinkkemper, S. (2009, May). A sense of community: A research agenda for software ecosystems. In Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on (pp. 187-190). Vancouver, BC, Canada, 2009. IEEE Computer Society.

Manikas, K., & Hansen, K. M. (2013). Software ecosystems–A systematic literature review. Journal of Systems and Software, 86(5), 1294-1306.

Manikas, K. (2016). Revisiting software ecosystems research: A longitudinal literature study. Journal of Systems and Software, 117, 84-103.

Mens, T., Goeminne, M. (2011). Analysing the evolution of social aspects of open source software ecosystems. In: Proceedings of the Third International Workshop on Software Ecosystems (IWSECO-2011). CEUR-WS, pp. 1–14

de Souza, C. R., Figueira Filho, F., Miranda, M., Ferreira, R. P., Treude, C., & Singer, L. (2016, May). The social side of software platform ecosystems. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (pp. 3204-3214). ACM.

Aquila: uma Linguagem de Domínio Específico para Teste Baseado em Modelos para Projetos Ágeis

Aline Zanin, Avelino F. Zorzo

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)

aline.zanin@acad.pucrs.br, avelino.zorzo@pucrs.br

Resumo. *A aplicação da técnica de Teste Baseado em Modelos (Model-based Testing - MBT) pode trazer diversos benefícios para a melhoria da qualidade de software. Usualmente MBT é aplicado apenas em modelos tradicionais de ciclo de vida de desenvolvimento de software e.g. Cascata, sendo que poucos trabalhos exploram a sua aplicação no contexto de equipes ágeis. Desta forma, neste trabalho apresenta-se uma Linguagem de Domínio Específico, denominada Aquila, que foi projetada para modelagem de testes funcionais em equipes ágeis. Aquila é uma extensão da Linguagem de Domínio Específico Domain-Specific Language (DSL) Gherkin, aonde novas palavras chaves, relacionadas a testes funcionais, são adicionadas para permitir a geração automatizada de scripts. Neste trabalho apresentamos uma revisão de literatura, a arquitetura da linguagem Aquila e um exemplo de uso da aplicação desta DSL.*

Abstract. *Although Model-based Testing (MBT) brings several benefits for improving software quality, usually MBT is only used in traditional software development life cyles, e.g. waterfall, and there is almost no work that shows how to use MBT in agile environments. Hence, this paper presents a Domain-dpecific Language (DSL), called Aquila, which was designed to model functional testing in agile environments. Aquila is an extension of the Ghergin DSL in which new keywords related to agile software development and functional testing are added. In this paper, we present a literature review, Aquila's architecture and a example of use that was used to show how Aquila can be applied.*

1. Introdução e Motivação

A busca crescente por aumento da qualidade nos produtos e serviços entregues aos clientes, faz com que as empresas procurem otimizar os seus processos e ferramentas para tornar possível atender a este propósito. Neste sentido, as técnicas utilizadas para realização de testes de software precisam acompanhar esta tendência e estarem sendo atualizadas e aprimoradas constantemente.

Entre estas técnicas encontra-se Teste Baseado em Modelos (*Model-Based Testing* - MBT). MBT fundamenta-se na criação de artefatos de testes, baseados na reutilização dos modelos criados pelas equipes de desenvolvimento para a realização de outras atividades, como por exemplo: análise e especificação de requisitos, análise de sistemas e programação. A técnica de MBT agrega diversos benefícios às empresas que optam pela sua utilização, estando entre os principais: facilitação da manutenção dos artefatos de testes; melhor rastreabilidade entre requisito de sistema e artefato de teste; e melhor visualização dos requisitos, de forma gráfica, através de modelos [El-Far and Whittaker 2001].

Contudo, embora os benefícios de MBT já sejam conhecidos e difundidos, poucos trabalhos apresentam a aplicação desta técnica em projetos que fazem uso de metodologias ágeis de desenvolvimento de sistemas. Isto se deve ao

fato de MBT trazer alguns desafios quando aplicado neste contexto. Estes desafios dizem respeito, principalmente a gerência do tempo e dos recursos demandados para a criação dos modelos e ao fato deles serem geralmente construídos em ferramentas separadas das ferramentas de desenvolvimento tradicionalmente utilizadas [Katara and Kervinen 2006][Jalalinasab and Ramsin 2012][Entin et al. 2012].

Estes desafios são notados especificamente no contexto de desenvolvimento ágil de software porque as equipes trabalham com ciclos curtos de entregas contínuas, o que exige que o tempo de criação dos modelos seja reduzido para que exista tempo hábil para execução dos testes antes da entrega ao cliente. Além disso, as metodologias ágeis de desenvolvimento embasam sua estrutura no manifesto ágil para o desenvolvimento de sistemas [Fowler and Highsmith 2001] e neste documento, os princípios de comunicação, interação e trabalho colaborativo, acima de documentações extensas, responsabilidades individuais e burocracia contratual, são norteadores. Desta forma torna-se inviável a dedicação de profissionais especificamente para a criação dos modelos.

Visando o cumprimento do princípio básico de colaboração e integração entre os profissionais que atuam no desenvolvimento do sistema e destes com o cliente, é comum em equipes ágeis a utilização de *user stories* para documentação de requisitos [VersionOne 2017]. O uso de *user stories* se popularizou por introduzir uma linguagem seminatural na especificação de requisitos tornando-os compreensíveis para o cliente final do produto, sendo um mecanismo eficiente de comunicação durante todo o processo de desenvolvimento, desde o contato com o cliente até a entrega final. Além disso, atualmente existem no mercado técnicas que fazem uso desta documentação para a produção de artefatos utilizados no desenvolvimento do sistema, esta atividade é chamada de *Living Documentation*. Um exemplo de técnica que explora a utilização de *Living Documentation* é a de *Behavior Driven Development - BDD*, que através da linguagem Gherkin descreve de forma detalhada os cenários que compõem cada uma das *user stories*, mantendo a linguagem natural e efetuando a geração de métodos que guiam a automação de testes.

Contudo, a técnica de BDD não prove a criação completa dos *scripts* de testes, sendo necessário que o testador crie de forma manual cada uma das ações que deseja automatizar, além disso a técnica de BDD não apresenta para o testador sequências de teste, sendo necessário que esse efetue a criação das sequências segundo o seu conhecimento do sistema. Desta forma, aliar a técnica de MBT com a técnica de BDD pode agregar vantagens para equipes ágeis. Para este fim, uma das alternativas é por meio da utilização de Linguagens de Domínio Específico (Domain-specific Languages - DSLs). DSLs são linguagens projetadas para atender as necessidades de um domínio específico e fazer uso de vocabulários que se aproximem deste domínio. Neste contexto, propomos neste trabalho a DSL Aquila, uma DSL que será utilizada para representação de cenários de teste através de linguagem seminatural com a adição de informações de teste e sistema, viabilizando a criação automatizada de modelos para execução de MBT.

Para a criação desta DSL efetuamos primeiramente uma revisão de literatura visando identificar quais desafios e dificuldades no processo de MBT foram descritos na literatura como os potenciais empecilhos da aplicação da técnica. Após percebermos que o principal problema está na criação dos modelos, buscamos identificar como as equipes ágeis documentam requisitos, e nesta busca percebemos que os requisitos são documentados em forma de *user stories* [VersionOne 2017]. Baseado nisso, e considerando o *background* de nosso grupo de pesquisa em teste baseado em modelos [Rodrigues et al. 2010], [Rodrigues et al. 2015], [Bernardino et al. 2016], propomos a arquitetura da DSL Aquila

e a avaliamos preliminarmente através de um exemplo de uso.

2. Trabalhos Relacionados

A fim de embasar nossa pesquisa, primeiramente efetuamos uma busca na literatura por trabalhos que apresentam métodos, abordagens ou processos para realização de MBT para teste funcional de software no contexto de desenvolvimento ágil. Posteriormente, efetuamos uma busca por DSLs para testes funcionais de software. E, por fim, buscamos agrupar todos estes conceitos, buscando trabalhos que apresentem DSLs para teste funcional de software baseado em modelos para o desenvolvimento ágil. A seguir apresentamos os trabalhos localizados, que possuem maior correlação com a DSL Aquila, proposta neste estudo.

Utting Utting *et al.* (2007) [Utting et al. 2007] apresentam em seu trabalho uma ferramenta para geração automatizada de *scripts* de testes a partir de modelos. Contudo, esta ferramenta não considera a utilização de Gherkin.

Entin *et al.* (2015) [Entin et al. 2015] apresentam a criação automatizada de modelo com base nos requisitos escritos na linguagem Gherkin. O que difere a DSL Aquila deste trabalho é que, neste trabalho é utilizado Gherkin em sua estrutura pura, e por este motivo os modelos gerados não são ricos em detalhes, são replicação literal das informações inseridas nos cenários Gherkin.

Li *et al.* (2016) [Li et al. 2016] apresentam uma ferramenta que a partir de modelos UML efetua a criação de cenários para a ferramenta Cucumber. Neste trabalho a partir dos modelos são gerados *scripts* de testes e a partir destes *scripts* são gerados os cenários para serem executados na ferramenta Cucumber. Este trabalho se assemelha com a linguagem Aquila porque ambos fazem uso de Gherkin, contudo, o processo utilizado pela DSL Aquila é o inverso, sendo gerados modelos a partir de Gherkin e não Gherkin a partir de modelos.

Elallaoui *et al.* (2015) [Elallaoui et al. 2015] se propõem em seu trabalho a gerar, através de *user stories*, diagramas de sequências. Embora este trabalho se assemelhe com a linguagem Aquila por efetuar a geração de modelos, a linguagem Aquila se propõe a gerar diagramas de atividades com um nível de detalhamento elevado, permitindo a aplicação de MBT. Neste trabalho os autores efetuam apenas a geração de diagramas de sequência o que não compreende dados minimalistas sobre o sistema que permitam a aplicação de MBT.

Elallaoui *et al.* (2016) [ELA 2016] propõem a geração de *scripts* de teste baseado em diagramas de sequência. O processo apresentado neste trabalho considera a criação de modelo de sequencias simples, migrado para um modelo de sequências com informações de testes, sendo que, posteriormente estes modelos completos que foram gerados são transformados em código de teste automatizado.

Dwarakanath *et al.* (2017) [Dwarakanath et al. 2017] apresentam uma DSL para geração de *scripts* de teste funcional baseado em Gherkin. Esta DSL tem propósito similar a DSL Aquila, contudo, não é destinada à aplicação de MBT, e com isso não permite aos projetos que a utilizam, beneficiar-se das vantagens de MBT.

Embora existam vários trabalhos que mencionem DSL, testes funcionais ou testes baseado em modelos aplicado em ambientes ágeis, no melhor de nosso conhecimento, não encontramos nenhum trabalho que aborde a utilização de DSL para modelagem de teste baseado em modelos para equipes ágeis. Visando minimizar esta lacuna, propomos

uma nova DSL, que através da geração de modelos, baseada em requisitos escritos em linguagem seminatural, permite a união dos conceitos supra citados e agrega os benefícios de MBT para o contexto de times ágeis de desenvolvimento de sistemas.

3. Aquila

Visando propor uma alternativa facilitada para a realização de MBT, e considerando um enfoque principal em equipes que atuam com metodologias ágeis de desenvolvimento de software, propomos neste trabalho a DSL Aquila. Esta DSL propõe estender a DSL Gherkin para tornar possível a criação automática de modelos para aplicação de MBT a partir dos cenários escritos em linguagem seminatural. Isto deverá ser feito através da agregação de palavras chaves que representam informações de sistema. Dentre essas informações se encontram referências às ações que precisam ser executadas no sistema para realizar um teste, por exemplo: preenchimento correto dos campos presentes na interface gráfica de determinada funcionalidade. Com a geração automatizada de modelos com base nos cenários, se torna possível a automação da criação de *scripts* de teste, bem como, facilita-se a manutenção destes artefatos, uma vez que a atualização de um determinado requisito de sistema pode ser replicada para o *script* de teste equivalente de forma automatizada.

A geração automatizada de *scripts* de teste a partir de modelos é um processo já existente e aplicado por diversos pesquisadores, entre eles: [Domingues et al. 2016][Bouquet et al. 2008][Lasalle et al. 2011]. Contudo, estes trabalhos consideram a criação manual dos modelos ou a geração de modelos a partir de código fonte, não considerando a geração de modelos a partir de cenários Gherkin. A criação manual de modelos, em processo de desenvolvimento ágil, costuma ser citada como um dos principais pontos que dificultam a implantação de MBT, principalmente devido aos ciclos curtos de entrega contínua aplicados nas metodologias ágeis [Entin et al. 2011]. Já a geração de *scripts* de teste a partir de código fonte apresenta desvantagens uma vez que, pode ocasionar a realização de testes tendenciosos, sendo que, um dos princípios do teste de software segundo [ISTQB] é realizar os testes com base nos requisitos do sistema e não com base no sistema implementado. Contudo, a desvantagem supra citada não se aplica nos casos de testes de regressão aonde visa-se garantir que um sistema continua em funcionamento após uma alteração de código [ISTQB].

Diferente do que propõem estes dois tipos de estratégias, a DSL Aquila permite a geração de *scripts* de teste baseado exclusivamente nos requisitos especificados, nos padrões estabelecidos e no conjunto de entradas e saídas esperadas para o sistema. Esta característica faz com que os artefatos de teste gerados representem aquilo que o sistema deve fazer e não aquilo que o sistema implantado faz, aumentando assim a eficiência dos testes. Nas próximas seções apresenta-se a definição do domínio, arquitetura e requisitos desta DSL.

3.1. Análise de Domínio

O desenvolvimento de uma DSL, segundo [Van Deursen et al. 2000] costumeiramente envolve três etapas distintas, sendo elas: Análise, Implementação e Utilização. Dentro da etapa de análise, a atividade principal é a definição do domínio do problema, ou seja a definição do contexto em que a DSL será empregada. Segundo [Neighbors 1984] a definição de domínio compreende entender as necessidade e requisitos de um conjunto de sistemas que possuem características similares, sendo o profissional responsável por isso, o analista de domínio ou *domain analyst*. No contexto da DSL Aquila definiu-se como

domínio específico **teste de software funcional para sistemas web em equipes de desenvolvimento ágil de software**.

3.2. Requisitos da Linguagem

Nesta seção são descritos os requisitos que formam esta DSL. Cada um dos requisitos presenta uma característica que esta DSL precisa ter para atingir o propósito para o qual foi criada. Estes requisitos dizem respeito ao domínio desta DSL (testes funcionais de software).

Requisito 1. A DSL deverá ser textual e deve ser estruturada baseada em linguagem seminatural.

O termo seminatural se trata de um subconjunto da Linguagem Natural e em razão de que deseja-se evitar a ambiguidade inerente das linguagens naturais, ou seja, o duplo significado de que determinadas palavras podem assumir em um mesmo contexto. A linguagem seminatural utilizada neste trabalho será proposta na linguagem estrangeira inglês para facilitar a aplicação da DSL em âmbito internacional. A escrita de requisitos no formato de linguagem seminatural, através da DSL Gherkin, é um padrão difundido em equipes de desenvolvimento ágil de software. Por este motivo a DSL Aquila deve seguir esta mesma estrutura de escrita visando a fácil utilização e aprendizado pelos profissionais das equipes ágeis que já estão adaptados a escrever requisitos neste formato.

Requisito 2. A DSL construída deve permitir a escrita de informações de testes funcionais de software.

O grande objetivo do desenvolvimento desta DSL é construir uma ligação entre a escrita de requisitos em linguagem seminatural e a modelagem funcional de testes para aplicação de MBT. Desta forma, é de extrema importância que a DSL proposta tenha agregada em sua estrutura a inserção de informações de testes como dados essenciais para a construção de modelos comportamentais que serão a saída da linguagem.

Requisito 3. A DSL construída deve permitir a adição de informações de sistema que permitam uma rastreabilidade entre os requisitos e o sistema desenvolvido.

Para que possa ser atendido o objetivo central do desenvolvimento desta DSL, no que diz respeito a criação de modelos comportamentais acrescidos de informações de testes e sistema que permitam a aplicação da técnica de MBT, se faz necessário incluir informações de sistema na estrutura da DSL proposta. Essas informações dizem respeito ao *front-end* das aplicações web que serão testadas. Isto é importante, uma vez que, para que se possa automatizar a criação de *scripts* de teste, é necessário que exista uma forma de identificar os campos que estão presentes na interface da aplicação e correlaciona-los com os modelos comportamentais.

Requisito 4. A DSL proposta deve permitir a realização de um *parser* para modelos comportamentais *e.g.* diagramas de atividades da UML.

Todas as informações referentes a ações de usuário e padrões de sistema que forem inseridas na DSL Aquila precisam ser refletidas automaticamente para os modelos comportamentais gerados. Uma vez que, este trabalho tem por objetivo criar modelos que permitam a geração de *scripts* automatizados de teste, a DSL deverá gerar diagramas de atividades da UML. A especificidade por este diagrama se dá como requisito por, entre os diagramas comportamentais, ser o que permite a inserção de maiores detalhes sobre as ações do usuário no sistema e os resultados esperados para cada uma delas.

Requisito 5. A DSL proposta deve permitir a geração de *scripts* de teste para aplicações web, independentemente da plataforma ou linguagem de programação na qual as aplicações foram programadas.

A DSL proposta deverá atender a um domínio específico, o da realização de testes funcionais, porém, não deverá se limitar a nenhuma plataforma ou linguagem de programação. A DSL terá seu domínio no âmbito de requisitos, gerando modelos que permitam a criação de *scripts* independentes do código fonte da aplicação. Desta forma os *scripts* gerados pela aplicação da técnica de MBT nos modelos gerados pela DSL serão classificados como *scripts* de teste de caixa preta.

3.3. Decisões de Projeto

Para atender os requisitos citados acima, se faz necessário estabelecer alguns padrões e definições arquiteturais para a DSL. Neste contexto algumas decisões estratégicas foram tomadas de forma a tornar possível atingir estes objetivos.

Decisão de Projeto 1: A DSL Aquila será projetada como extensão da DSL Gherkin.

A DSL Aquila foi projeta como uma extensão da DSL Gherkin, visando aproveitar os conhecimentos que os profissionais que atuam em equipes ágeis de desenvolvimento já possuem sobre esta DSL. Isto permite reduzir a curva de aprendizagem e facilitar a adaptabilidade para utilização da DSL Aquila. Além disso, optou-se por estender Gherkin, por ela ser compatível com testes funcionais uma vez que, descreve comportamentos. Aquila complementa Gherkin adicionando maiores informações e detalhamentos.

Decisão de Projeto 2: A DSL Aquila será composta por Palavras Chave.

A DSL Aquila tem como principal característica a inserção de informações de testes em cenários Gherkin. Desta forma, se faz necessário estabelecer um conjunto de palavras chaves que devem ser utilizadas para representar o comportamento do usuário no sistema e consequentemente serem refletidas em artefatos de testes. A Tabela 1 apresenta as palavras chaves definidas para a representação destes comportamentos. Na coluna da esquerda são representadas as palavras chaves e na coluna da direita o comportamento representado por cada uma elas.

Decisão de Projeto 3: A DSL Aquila deverá ser utilizada seguindo uma estrutura de arquivos no projeto de desenvolvimento.

Para a facilitação da geração dos testes e a melhor representação dos requisitos em modelos, estabeleceu-se uma estrutura para a criação dos arquivos que contém as funcionalidades escritas no formato Aquila. Para esta padronização levamos em consideração algumas convenções já definidas, sendo elas as seguintes: uma *feature* descreve uma funcionalidade em teste e pode agrupar diversos *scenários*, sendo um *scenario* a especificação mais detalhada de uma *feature*. Desta forma, na estrutura da DSL Aquila, todos os cenários referentes a uma *features* devem ser descritas em um único arquivo e cada *feature* representará um modelo.

Decisão de Projeto 4: A fim de estabelecer a rastreabilidade entre requisitos e modelo fica definido o seguinte:

1. A palavra GIVEN estabelece o início do modelo de uma *feature*;
2. A Palavra WHEN representa a criação de uma nova atividade no modelo;
3. A Palavra AND dentro da cláusula WHEN representa a criação de uma nova atividade no modelo;

Tabela 1. Palavras Chave Linguagem Aquila

Tag	Definição
<>	Marca caminho de tela
{}	marca a chamada para outro cenário
[]	marca nome de um campo específico
click	Indica o click em um botão
use-valid-data-on	Indica o preenchimento de um campo específico com valor válido
use-invalid-data-on	Indica o preenchimento de um campo específico com valor inválido
use valid data	indica que serão preenchidos valores válidos para todos os campos
save	indica que os dados devem ser salvos
show-sucess-message	indica que deve ser exibida uma mensagem de sucesso
show-error-message	indica que deve ser exibida uma mensagem de erro
input-data	indica que haverá entrada de dados
checked	indica seleção em checkbox
choose	indica seleção em radio
enables	indica que o sistema deverá habilitar determinado campo
disables	indica que o sistema deverá desabilitar determinado campo
successfully-saved	Indica que os dados submetidos foram salvos com sucesso
dont-fill-out	indica que um determinado campo não será preenchido
select-data	indica seleção em list
GIVEN	estado inicial do sistema
WHEN	ação
THEN	resultado de uma ação
AND	conectivo para mais de uma ação ou resultado

4. Quando utilizada a palavra USE-VALID-DATA na cláusula WHEN, deverá ser criada uma atividade no modelo para cada um dos campos presentes na interface da funcionalidade;
5. Quando utilizado USE-VALID-DATA e a aplicação possuir campos que permitem seleção, deverá ser criado no modelo um ponto de decisão onde sejam permitidos fluxos distintos de acordo com o selecionado. Por exemplo: se a aplicação possuir um *radio button* "rdbTipo" para representar Pessoas Físicas e Pessoas Jurídicas, deverá ser criado no modelo um *decision point* com dois fluxos, um cobrindo os campos referente a pessoa física e outro os campos referente a pessoa jurídica;
6. Campos que pertencem a um fluxo específico devem conter um padrão de nomenclatura com prefixo que indica o fluxo a qual pertencem;
7. Quando utilizada a palavra USE-INVALID-DATA-ON[nome do campo] deverá ser criado um ponto de decisão no modelo antes do campo para a criação do *scenario* negativo;
8. Quando utilizada a palavra DON'T-FILL-OUT[nome do campo]deverá ser criado um ponto de decisão no modelo antes do referido campo para a criação do *scenario* negativo. Caso o ponto de decisão já exista, deverá ser criado apenas uma atividade para a validação do fluxo sem preenchimento do campo;
9. Quando o caso de teste exigir a seleção de um valor específico no campo de *radio button* será utilizada a palavra CHOOSE. Neste caso será criado no modelo um fluxo específico para a opção descrita no CHOOSE. O mesmo acontece para a seleção de valores específicos em *checkbox*, através da palavra CHECKED e para seleção de valores específicos em listas, através da palavra SELECT-DATA;
10. As situações descritas na cláusula THEN, não serão convertidas em atividades nos modelos, e sim em resultados esperados para os testes.

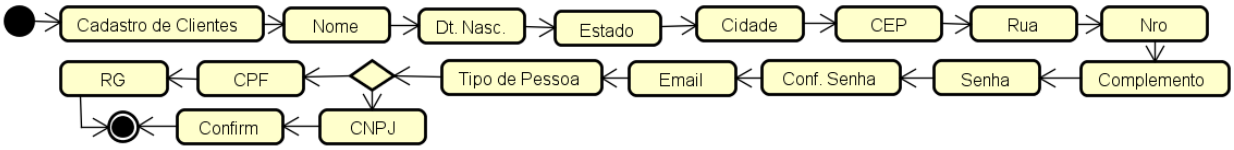


Figura 1. Fluxo Principal Cadastro de Clientes Gerado a Partir da DSL Aquila (cenário 1)

4. Exemplo de uso

A fim de realizarmos uma avaliação preliminar da DSL Aquila, realizamos a modelagem de teste funcional de um software. Este software trata-se de um cadastro de pessoas que tem um fluxo principal e oito fluxos alternativos, sendo eles; inserção de data de nascimento inválida; CEP inválido; email inválido e confirmação de senha inválida, não preenchimento de dados obrigatórios. Para exemplificação da utilização da DSL Aquila foram modelados estes fluxos supracitados e fluxo de seleção de valores em *radio button*. Na Tabela 2 descrevemos o fluxo principal, o fluxo de data de nascimento inválida e o fluxo de Salvar Dados sem Preencher Campos Obrigatórios. Os demais fluxos podem ser vistos modelados utilizando a DSL Aquila no seguinte link: https://goo.gl/Ewb1pe

Tabela 2. Cenários Aquila

1	Fluxo Principal representa o comportamento do sistema quando todos os campos são preenchidos com informações válidas.	GIVEN Customer Registration Page WHEN I **use-valid-data** AND I **click [enviar]** THEN Data is **successfully-saved**
2	Fluxo Alternativo Preenchimento inválido do Campo Data de Nascimento	GIVEN Customer Registration Page WHEN I **use-invalid-data-on[data de nascimento]** AND I **click [enviar]** THEN The data **is-not-saved**
3	Fluxo Alternativo Salvar Dados sem Preencher Campos Obrigatórios	GIVEN Customer Registration Page WHEN I **use-valid-data** AND I **choose[pessoa fisica]** AND I **dont-fill-out[CPF]** THEN The data **is-not-saved** AND The System **show-error-message**

4.1. Modelo Comportamental Gerado Após a Modelagem de Teste Utilizado a Ferramenta Aquila

O primeiro cenário escrito com a DSL Aquila, demonstra o comportamento do fluxo principal do sistema, aonde todos os campos são preenchidos com valores válidos. Este modelo é o ponto de partida para a criação de todos os demais e pode ser visualizado na Figura 1. Os demais cenários refletem fluxos alternativos, um exemplo pode ser visto na Figura 2. A Figura 3 representa o modelo geral da aplicação em teste, compreendendo todos os fluxos alternativos.

5. Conclusões e Trabalhos Futuros

Apresentamos neste trabalho a DSL Aquila e um exemplo de sua utilização. Através deste exemplo de uso podemos concluir preliminarmente que é possível efetuar a criação

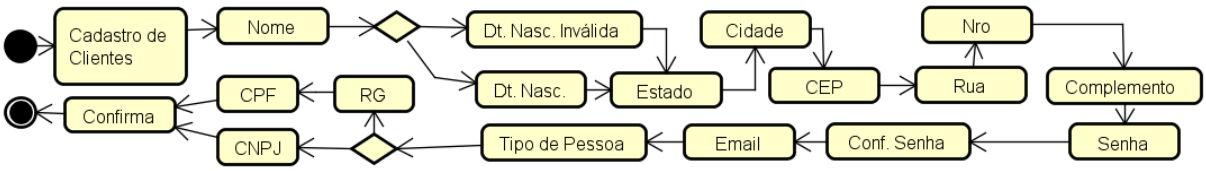


Figura 2. Fluxo Alternativo Efetuar Cadastro com Data de Nascimento Inválida (cenário e)

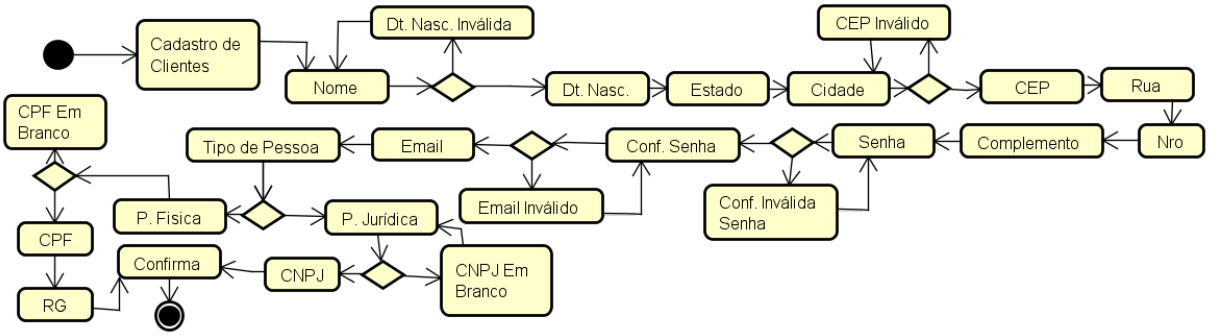


Figura 3. Modelo que Descreve a União de Todos os Cenários Contemplando Fluxo Principal e Fluxos Alternativos

de modelos para aplicação da técnica de MBT de acordo com os requisitos especificados no formato Aquila. Futuramente pretende-se expandir o exemplo de uso, efetuar a validação em um cenário real de uma empresa que trabalha com desenvolvimento de sistemas, bem como realizar reuniões com especialistas em desenvolvimento ágil de software para garantir que a DSL Aquila agrega benefícios para realização de testes em times ágeis e que a extensão realizada na linguagem Gherkin mantém as qualidades que a linguagem Gherkin possui, e não afeta a usabilidade desta linguagem.

Referências

(2016). Automated model driven testing using andromda and uml2 testing profile in scrum process. *Procedia Computer Science*, 83(Supplement C):221 – 228.

Bernardino, M., Zorzo, A. F., and Rodrigues, E. M. (2016). Canopus: A domain-specific language for modeling performance testing. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*.

Bouquet, F., Grandpierre, C., Legeard, B., and Peureux, F. (2008). A test generation solution to automate software testing.

Domingues, A., Rodrigues, E. M., and Bernardino, M. (2016). Autofun: An automated model-based functional testing tool. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing*. ACM.

Dwarakanath, A., Era, D., Priyadarshi, A., Dubash, N., and Podder, S. (2017). Accelerating test automation through a domain specific language. In *Software Testing, Verification and Validation (ICST), 2017 IEEE International Conference on*.

El-Far, I. K. and Whittaker, J. A. (2001). Model-based software testing. *Encyclopedia of Software Engineering*.

Elallaoui, M., Nafil, K., and Touahni, R. (2015). Automatic generation of uml sequence diagrams from user stories in scrum process. In *Intelligent Systems: Theories and Applications (SITA), 2015 10th International Conference on*, pages 1–6. IEEE.

Entin, V., Winder, M., Zhang, B., and Christmann, S. (2011). Combining model-based and capture-replay testing techniques of graphical user interfaces: An industrial approach. In *Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*.

Entin, V., Winder, M., Zhang, B., and Christmann, S. (2012). Introducing model-based testing in an industrial Scrum project. In *Proceedings of the 7th International Workshop on Automation of Software Test*.

Entin, V., Winder, M., Zhang, B., and Claus, A. (2015). A process to increase the model quality in the context of model-based testing. In *Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on*.

Fowler, M. and Highsmith, J. (2001). The agile manifesto. *Software Development*, 9(8):28–35.

ISTQB, I. S. T. Q. B. Syllabi. http://www.istqb.org/downloads/syllabi.html. acessado em 11/06/2017.

Jalalinasab, D. and Ramsin, R. (2012). Towards model-based testing patterns for enhancing agile methodologies. In *SoMeT*.

Katara, M. and Kervinen, A. (2006). Making model-based testing more agile: a use case driven approach. In *Proceedings of the Haifa Verification Conference*.

Lasalle, J., Peureux, F., and Fondement, F. (2011). Development of an automated mbt toolchain from uml/sysml models. *Innovations in Systems and Software Engineering*, 7(4).

Li, N., Escalona, A., and Kamal, T. (2016). Skyfire: Model-based testing with cucumber. In *Software Testing, Verification and Validation (ICST), 2016 IEEE International Conference on*.

Neighbors, J. M. (1984). The draco approach to constructing software from reusable components. *IEEE Transactions on Software Engineering*, SE-10(5):564–574.

Rodrigues, E. M., de Oliveira, F. M., Costa, L. T., Bernardino, M., Zorzo, A. F., Souza, S. d. R. S., and Saad, R. (2015). An empirical comparison of model-based and capture and replay approaches for performance testing. *Empirical Software Engineering*, 20(6).

Rodrigues, E. M., Viccari, L. D., Zorzo, A. F., and Gimenes, I. (2010). PLeTs - Test automation using software product lines and model based testing. In *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering*.

Utting, M., Perrone, G., Winchester, J., Thompson, S., Yang, R., and Douangsavanh, P. (2007). The modeljunit model-based testing tool. acessado em 11/06/2017.

Van Deursen, A., Klint, P., Visser, J., et al. (2000). Domain-specific languages: An annotated bibliography. *Sigplan Notices*, 35(6):26–36.

VersionOne (2017). State of agile report. http://stateofagile.versionone.com/. acessado em 11/06/2017.

Utilização do Algoritmo de QSM para automação de testes a partir de cenários Gherkin

Aline Zanin, Henry C. Nunes, Avelino F. Zorzo, Flavio M. de Oliveira

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681 - Prédio 32 - Porto Alegre - RS - Brasil

aline.zanin@acad.pucrs.br, henry.nunes@outlook.com,

avelino.zorzo@pucrs.br, flavio.mo.oliveira@gmail.com

Abstract. *One of the present challenges in software development is the representation of requirements in a clear and intuitive way and at the same time make it productively viable in software development and testing - which imply in intensive use of automation. To ease this work, various techniques are utilized, as an example, the requirements writing in languages as Gherkin and the creation of formal models. In this work, we aim to unify the advantages offered by both techniques through the proposal of a model for automated generation of test scripts. This model includes the generation of Finite State Machines (FSM), with Gherkin scenarios, using the Query-driven State Merging (QSM) algorithm. Using the resulting FSM, we can use Model-based Testing (MBT) techniques for generation of a more robust set of test cases, which can be used to generate test scripts, in an automatic way too. In this work, we performed a literature review and verified the proposed model through an example of use. In this example, we perceived the viability of the model and the contribution for academy and industry.*

Resumo. *Um dos desafios presentes no desenvolvimento de software é a representação de requisitos de forma clara e intuitiva, por um lado, e ao mesmo tempo viabilizar produtividade no desenvolvimento e testes de sistemas - o que implica o uso intensivo de automação. Para facilitar este trabalho, diversas técnicas são utilizadas, por exemplo, a escrita de requisitos em linguagens como Gherkin e a criação de modelos formais. Neste trabalho, buscamos unir as vantagens oferecidas por ambas as técnicas por meio da proposta de um modelo para geração automatizada de scripts de teste. Este modelo contempla a geração de Máquinas de Estados Finitos (MEFs), a partir de cenários Gherkin, utilizando o algoritmo Query-driven State Merging (QSM). A partir da MEF gerada, utilizamos técnicas de Teste Baseado em Modelos (Model-based Testing - MBT) para gerar um conjunto robusto de casos de teste, a partir dos quais geram-se scripts de teste, de forma também automática. Para o desenvolvimento deste trabalho, nos baseamos em uma revisão de literatura, e posteriormente validamos o modelo proposto por meio de um exemplo de uso. Neste exemplo de uso percebe-se a viabilidade da utilização deste modelo e a contribuição para academia e para indústria.*

1. Introdução

O desenvolvimento de software é um trabalho complexo que envolve diversas habilidades e práticas. Este trabalho é costumeiramente realizado por mais de uma pessoa, formando

equipes de desenvolvimento. Nestas equipes, todos os envolvidos precisam comunicar-se de forma eficaz, para que possa ser realizado um trabalho colaborativo e atinja-se êxito nos projetos de desenvolvimento de sistemas. Além disso, a comunicação dos membros da equipe com o cliente é de extrema importância, uma vez que software é desenvolvido visando atender alguma necessidade de algum segmento ou cliente específico.

Desta forma, são necessários recursos para que o profissional responsável por elicitar os requisitos do software junto ao cliente, possa representá-los de forma clara e intuitiva. Isto porque é necessário que o mesmo requisito, utilizado pelas equipes de desenvolvimento para guiar seu trabalho, seja visível e compreensível pelo cliente. Desta forma, o cliente pode verificar se o software que está sendo produzido é o software desejado.

Neste contexto diversas são as técnicas utilizadas para facilitar a representação de requisitos. Uma delas é a utilização de descrição em linguagem semi-natural por meio da Linguagem de Domínio Específico (DSL - Domain-specific Language) Gherkin. Esta representação tem sua escrita facilitada, dada a simplicidade e a proximidade com o idioma comumente utilizado. Esta prática é consolidada especialmente em equipes ágeis, por meio da técnica de Desenvolvimento Guiado por Comportamento (*Behavior-driven Development -BDD*). Esta técnica permite a utilização de uma documentação viva que estrutura o desenvolvimento do sistema de acordo com os requisitos escritos. Contudo, esta técnica não efetua a geração automatizada de *scripts* de teste com alta cobertura e não propicia ao cliente e aos desenvolvedores, uma visão gráfica do fluxo do sistema. Em geral, os testes definidos a partir de especificações em Gherkin limitam-se a testes de aceitação.

Estas duas características não compreendidas pela técnica de BDD, podem ser supridas pela aplicação da técnica de Teste Baseado em Modelos (*Model-based Testing – MBT*). Esta técnica consiste na geração de artefatos de teste a partir de modelos comportamentais, geralmente modelos de transição de estados. Em particular, no contexto deste estudo, trabalhamos com Máquinas de Estados Finitos (MEFs). Para que esta técnica possa ser aplicada, os requisitos do sistema precisam ser descritos em modelos comportamentais. Desta forma, além de propiciar a geração de artefatos de teste, a técnica propicia ao cliente e à equipe de desenvolvimento uma visão mais robusta de como será o fluxo comportamental do sistema que está sendo desenvolvido.

Entretanto, a técnica de MBT possui algumas desvantagens que em determinados contextos de projetos dificultam a sua aplicação. Uma delas é o tamanho dos modelos criados. Isto porque, para funcionalidades complexas, o modelo que precisa ser criado para representá-las se torna extenso e sua compreensão é dificultada. Além disso, para uma equipe que já utiliza Gherkin em um contexto de BDD, construir uma MEF equivale a um retrabalho e consequente aumento no custo do projeto.

Neste contexto, este trabalho propõe uma estratégia para utilização conjunta da DSL Gherkin, e de MBT. Esta estratégia permitirá uma melhor representação dos requisitos do sistema e a geração automatizada de artefatos de teste. Buscando contribuir com a redução do problema supracitado de geração de modelos demasiadamente complexos, propomos a geração automática de MEFs a partir de especificações Gherkin. Utilizamos o algoritmo *Query-driven State Merging – QSM* [Dupont et al. 2008] que propõe a indução de máquinas de estados finitos de forma que representem adequadamente a funcionalidade que está sendo modelada e testada, porém de forma otimizada reduzindo a

quantidade de estados da MEF, evitando retrabalho e, posteriormente, gerando artefatos de teste com alta cobertura.

Assim, este artigo se propõe a responder a seguinte questão de pesquisa: Aplicar o algoritmo de QSM, em MEF geradas a partir de requisitos escritos no formato Gherkin, pode otimizar os artefatos de testes gerados pela técnica de MBT? Para responder a esta pergunta, realizamos primeiramente uma revisão de literatura buscando identificar trabalhos relacionados e que pudessem contribuir com o modelo proposto. Posteriormente, efetuou-se a criação de um *parser* para geração de MEF a partir de Gherkin. Este *parser* baseou-se na proposta central de Gherkin, onde informações representadas da cláusula *when* representam ações, e valores representados na cláusula *then* representam resultados. A aplicação do algoritmo de QSM, por sua vez, baseou-se no proposto por [Dupont et al. 2008] e a geração de sequência de testes no método de geração de sequência de testes Wp [Fujiwara et al. 1991]. Por fim as sequências geradas pelo método Wp são convertidas em *drivers* de teste.

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta a fundamentação teórica que embasa este trabalho; a Seção 3 apresenta a estratégia proposto por este trabalho; a Seção 4 apresenta um exemplo de uso da estratégia e a Seção 5 apresenta as considerações finais e trabalhos futuros.

2. Fundamentação Teórica

Teste Baseado em Modelos *(Model-based Testing (MBT))* [El-Far and Whittaker 2002] é uma técnica de testes que consiste na utilização de modelos para a geração de artefatos de testes. MBT pode ser aplicado para todos os tipos de testes, sendo que, conforme o tipo de testes que será empregado, deve-se definir qual o melhor modelo para utilizar. Para a realização de testes funcionais, que é o foco deste trabalho, são exemplos de modelos que podem ser utilizados: diagrama de atividades da UML, diagrama de casos de uso da UML e especificações formais como as Máquinas de Estados Finitos (MEF).

Para todas as formas de utilização, MBT permite reduzir custos com geração de *scripts* de teste e correção de defeitos, permitindo ainda criar uma rastreabilidade entre modelo e requisito, de forma que, quando ocorre uma alteração de requisito de sistema, se torna possível recriar os *scripts* de teste de forma automatizada, mitigando o problema gerado pelo alto custo de manutenção dos artefatos de teste.

Neste mesmo contexto de buscar automatizar os processos de criar melhor rastreabilidade entre requisito e produto, diversas equipes tem utilizado a escrita de requisitos utilizando a linguagem de domínio específico (*Domain-specific Language - DSL*) Gherkin [Wynne et al. 2017]. Gherkin permite que os requisitos sejam escritos utilizando linguagem semi-natural, aproximando-se da linguagem do usuário e fazendo uso de palavras chaves pré-definidas que auxiliam na descrição do comportamento esperado para o sistema, sendo elas: GIVEN (representando o estado anterior do sistema), WHEN (representando uma ação que será executada), THEN (representando o resultado esperado). Esta DSL, em conjunto com ferramentas que dão suporte a sua utilização *e.g.* Cucumber permite a aplicação da técnica de *Behavior-driven Development (BDD)*, que proporciona a rastreabilidade dos requisitos com o código implementado.

Contudo, embora Gherkin e MBT sejam ferramentas que tem em comum, realizar automação de alguns processos e melhorar a rastreabilidade dos requisitos, elas costumeiramente não são aplicadas em conjunto. Neste sentido o algoritmo *Query-driven*

State Merging (QSM) pode ser uma importante ferramenta para o relacionamento destas duas técnicas. O algoritmo de QSM é um algoritmo de indução que tem por objetivo gerar MEFs a partir de um modelo de entrada. Neste trabalho, o modelo de entrada é uma especificação em Gherkin.

Para a geração das MEFs, o algoritmo considera um processo de indução onde inicialmente são inseridos na MEF os cenários positivos e posteriormente os cenários negativos. Para cada cenário negativo inserido, o algoritmo considera um processo de indução onde são verificadas e excluídas redundâncias de estados. O resultado é uma MEF que mapeia os comportamentos descritos na especificação de entrada.

2.1. Trabalhos Relacionados

A fim de complementar os nossos estudos buscamos trabalhos diretamente relacionados, contudo, não localizamos trabalhos que apresentem utilização de Gherkin e QSM e por este motivo relacionamos trabalhos que apresentam Gherkin ou QSM e que explorem soluções para realização de testes de software.

Seijas *et al.* [Seijas et al. 2016] no trabalho *Model extraction and test generation from JUnit test suites*, descreve uma técnica para realizar inferência em MEFs que modelam testes unitários de sistemas legados e analisa como gerar novos testes a partir destes modelos. A técnica é implementada utilizando a ferramenta James Tool que aplica a técnica em códigos Java e códigos de teste.

Sivanandan *et al.* (2014) [Sivanandan and B 2014], apresentam o trabalho *"Agile development cycle: Approach to design an effective Model-based Testing with Behaviour-driven automation framework"*. Este trabalho apresenta uma estratégia para geração de cenários no formato Gherkin a partir de modelos. De forma similar o trabalho "Skyfire: Model-based Testing With Cucumber" proposto por Li *et al.* (2016) [Li et al. 2016] também gera cenários Gherkin a partir de modelos, neste caso diagramas da UML e não especificação formal em MEFs.

Entin *et al.* (2015) [Entin et al. 2015] apresentam o trabalho "*A process to increase the model quality in the context of model-based testing"*, neste trabalho os autores apresentam a criação automatizada de modelo com base nos requisitos escritos na linguagem Gherkin. O trabalho apresentado pelos autores se assemelha com o trabalho que apresentamos neste estudo por gerar *script* a partir de cenários Gherkin, contudo, este trabalho não utiliza o algoritmo de QSM para otimizar os artefatos gerados.

3. Proposta de modelo para aplicação de MBT a partir de cenários Gherkin

O algoritmo QSM, conforme mencionado anteriormente, foi originalmente proposto para efetuar a transição de diagramas de atividade para máquinas de estados. Nesta seção demonstramos como realizamos a migração de cenários Gherkin para MEF.

3.1. Criação de cenários Gherkin

Nesta etapa aplica-se a criação dos cenários no formato Gherkin em sua estrutura tradicional, conforme a interpretada pela ferramenta Cucumber [Cucumber 2017], não sendo previsto por este trabalho nenhuma alteração nesta etapa. Nesta representação, os requisitos do sistema são separados por funcionalidades (*features*) e por cenários (*scenarios*). Uma *feature* representa a funcionalidade que está sendo desenvolvida e cada cenário descreve um comportamento esperado para aquela funcionalidade. Um cenário sempre possui um título e é formado pelas cláusulas GIVEN, WHEN e THEN, podendo ser acrescido

de cláusulas AND. GIVEN representa o estado atual do sistema no momento em que o cenário é executado, WHEN representa uma ação que será feita no sistema e está sendo descrita neste cenário e THEN representa o resultado do sistema para esta ação.

3.2. Geração de Sequências e criação de MEFs

Para cada *feature*, que contempla um conjunto de cenários, é criada uma MEF simulando um fluxograma em formato de árvore. O início da *feature* representa o estado inicial da MEF e a partir dele, cada um dos cenários Gherkin derivará um ramo da árvore. Para este fim, são criadas sequências que representam a ordem em que os comportamentos descritos em cada cenário serão executados. No momento da geração da MEF, cada item da sequência irá ser inserido na transição anterior a um estado e representar a entrada de um estado, sendo que, os estados são criados numerados em ordem crescente. A transição de saída de cada estado irá receber como valor a confirmação da entrada, isto porque, entradas representam ações e saídas, resultados esperados. Dado que todas as sequências criadas a partir do Gherkin são positivas, podemos considerar o resultado esperado para cada item delas como sendo a confirmação da ação (por exemplo: entrada abrir, saída aberto).

3.3. Eliminação de Não Determinismos

Em algumas situações podem ser geradas MEF com situações de não determinismo, ou seja, situações em que uma mesma entrada leva a dois ou mais estados distintos. Desta forma para resolver este problema são unidos os estados distintos em um único estado e as transições são removidas.

3.4. Redução da MEF por meio de induções

Esta etapa tem por objetivo remover redundâncias na MEF e com isso otimizar os artefatos de testes produzidos posteriormente. A redução da MEF por meio de induções é realizada por meio de múltiplas tentativas de união de estados. O processo de indução inicia sempre pelo estado de menor valor hierárquico sendo este estado comparado a todos os outros estados da MEF verificando as uniões possíveis.

A fim de verificar se uma união é possível, são geradas sequências contendo o prefixo do estado de menor hierarquia, e todos os sufixos do estado que está sendo comparado com este. Estas sequências representam fluxos do sistema modelado na MEF e são chamadas de *Queries*. Após gerar as *Queries* o algoritmo gera um conjunto de perguntas ao usuário sobre a aplicabilidade destas *Queries*. Caso o usuário considere positivas todas as *Queries*, ou seja, considere que com a união dos estados da MEF os fluxos resultantes são válidos, os estados são unidos, senão a MEF permanece inalterada.

3.5. Geração de Sequências de Teste com o método Wp

Um vez gerada a MEF ela é submetida como entrada para a aplicação do método Wp. O método Wp é um método de geração de sequências de testes, proposto por Fujiwara(1991) [Fujiwara et al. 1991], que tem por objetivo gerar sequências (caminhos) capazes de distinguir os estados da MEF por meio de seu comportamento. Para MEFs determinísticas e totalmente especificadas, o método garante a detecção de defeitos de transferência (transição para estado incorreto) e defeitos de saída (saída incorreta em uma transição). Desa forma, o método garante uma alta cobertura de toda a funcionalidade em teste. A partir destas sequências podem ser produzidos artefatos de execução de teste, como por exemplo *drivers* JUnit.

3.6. Geração de *Scripts* de Teste

A partir das sequências geradas pelo método Wp, produziu-se um *parser* para a criação do *driver* de teste, fazendo uso da biblioteca JUnit. Para cada uma das sequências foi gerado um método de teste e para cada item da sequência utilizou-se o comando *assertEquals*. Neste *assert* busca-se verificar se a resposta recebida é a confirmação da entrada enviada.

4. Exemplo de Uso

Para demonstração da aplicação da estratégia proposta foi utilizado um software didático que simula a funcionalidade de um estojo de canetas. Este estojo tem capacidade de armazenar uma única caneta e possui os estados aberto e fechado. O estojo está inicialmente sempre fechado e vazio, sendo que, para efetuar qualquer operação o estojo precisa ser aberto.

Abrir: Consiste na operação de abrir o estojo. Esta operação é pré-requisito para as operações de Adicionar e Remover que serão descritas na sequência. Desta forma só podem ser acionadas as funções de Adicionar e Remover quando o estojo estiver aberto; além disso, uma vez aberto o estojo ele não poderá ser aberto novamente sem antes ter sido fechado.

Fechar: Consiste na operação de fechar o estojo, podendo ser executada apenas quando o estojo está aberto. Estando fechado não é possível adicionar e nem remover uma caneta no estojo.

Adicionar: Adiciona uma caneta no estojo (a capacidade do estojo é de um único item). Após adicionada uma caneta, ela pode ser removido com a operação remover.

Remover: Remove um item no interior do estojo; como efeito, o estojo está vazio, podendo ser adicionada uma caneta.

Vazio: Operação que retorna verdadeiro caso o estojo esteja vazio.

4.1. Cenários Gherkin

O primeiro passo para a aplicação do modelo proposto neste trabalho é a escrita dos requisitos em formato de cenários Gherkin. Para o software porta-canetas, foram criados quatro cenários Gherkin, sendo eles Colocar Caneta (colocandoCaneta), Colocar e Retirar Caneta (colocaRetira), Abrir e Fechar Estojo (abreFecha) e Retira Caneta (retiraCaneta) os quais podem ser visualizados a seguir na *feature* EstojoCaneta:

```
Feature: EstojoCaneta
  Scenario: colocandoCaneta
    Given Abrir
    When Depositar
    And Fechar
    Then NaoVazio
  Scenario: colocaRetira
    Given Abrir
    When Depositar
    When Remover
    When Fechar
    Then Vazio
  Scenario: abreFecha
```

```
  Given Abrir
  Then Fechar
Scenario: retiraCaneta
  Given Vazio
  When Abrir
  And Depositar
  And Remover
  Then Vazio
```

4.2. Geração de Sequências e criação de MEF

A partir dos cenários descritos utilizando Gherkin, são criadas sequências que representam os comportamentos descritos no cenário. Para os cenários listados acima são criadas quatro sequências, sendo elas: *<Abrir, Depositar, Fechar, NaoVazio>*, *<Abrir, Depositar, Remover, Fechar, Vazio>*, *<Abrir, Fechar>*, *<Vazio, Abrir, Depositar, Remover, Vazio>*.

A partir destas sequências criou-se uma MEF, sendo que foi criado um estado inicial para a máquina e a partir dele foram derivados ramos em uma estrutura de árvore. Cada ramo da árvore formada consiste em umas das sequências positivas recebidas, como pode ser visto na Figura 1.

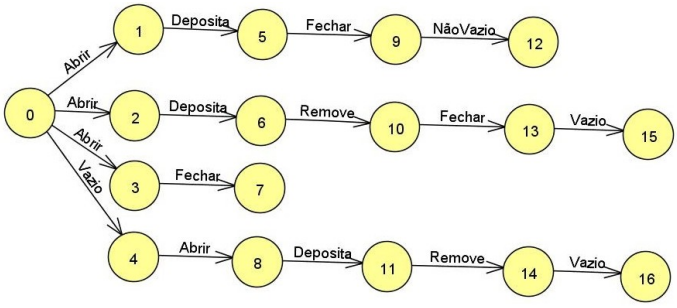


Figura 1. Primeira MEF gerada

4.3. Eliminação de Não Determinismos

A criação da máquina de estados dessa forma pode resultar em casos de não determinismo, ou seja, possuir estados onde um mesmo valor de entrada[1] leva a máquina para diferentes estados. Um exemplo pode ser observado no estado 0, que possui transições para os estados 1, 2 e 3 utilizando a mesma entrada do alfabeto, a entrada *Abrir*.

A fim de solucionar este problema, estas transições são removidas, fazendo a união dos estados que tem a mesma origem e, quando submetidas a mesma entrada, levam para estados diferentes. Caso esta união crie novos casos de não-determinismo o processo é repetido. Em nosso exemplo os estados 1, 2 e 3 são unidos em um único estado. Como regra, em toda união de estados é preservado o valor na hierarquia do menor número. Portanto eles são fundidos no estado 1. Como resultado passou a existir um novo não determinismo do estado 1 para o estado 5 e 6 quando a entrada do alfabeto é *Depositar*. Deste modo, o processo de unir os estados acontece novamente agrupando os estados no estado 5. Como resultado final se tem a máquina de estados que está representada na Figura 2.

[1]Considere entrada o valor que é descrito na transição antes

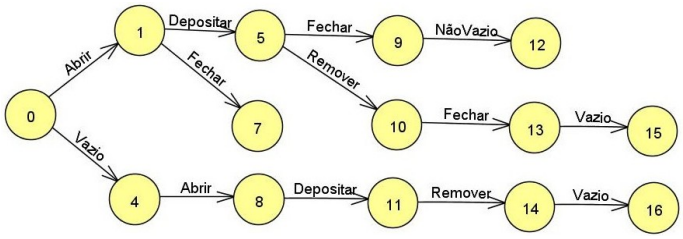


Figura 2. MEF Determinística

4.4. Redução da MEF por meio de induções

Para a MEF da Figura 2, o estado de menor hierarquia é o estado 0 e o de maior hierarquia o estado 16. Por este motivo o processo de indução começa comparando o estado 0 a todos os demais estados da MEF. A fim de verificar se uma união é possível, são geradas sequências contendo o prefixo do estado de menor hierarquia, e todos os sufixos do estado que está sendo comparado com este. Por exemplo: Para a MEF da Figura 2, a tentativa de união do estado 1 com o estado 5 resultaria nas seguintes *Queries*: *<Abrir, Fechar, NaoVazio>, <Abrir, Remover, Fechar, Vazio>*, isto porque o prefixo do estado 1 é ¡abrir¿ e os sufixos do estado 5 são: *<¡fechar, NaoVazio¿>* e *<¡Remover,fechar, vazio¿>*. Após gerar as *Queries* o algoritmo de QSM gera uma pergunta ao usuário sobre a sua validade. Caso o usuário considere positivas todas as *Queris*, ou seja, considere que com a união dos estados da MEF os fluxos resultantes são válidos, os estados são unidos, senão a MEF permanece inalterada.

Ao concluir as induções do estado de menor hierarquia com todos os demais estados da MEF, o algoritmo elege um novo estado como sendo o de menor hierarquia, considerando sempre o imediatamente posterior na ordem hierárquica e todo o processo de indução é repetido. Assim sucessivamente para todos os estados da MEF. Desta forma para a MEF da Figura 2 seria executada a seguinte ordem de tentativas:*<(0,1), (0,2),(0,3)...(0,16),(1,2),(1,3)...(1,16)(2,3)(2,4)...>*.

Após realizadas todas as uniões e gerada a MEF final, para ser possível aplicar métodos de geração de sequências de testes, mais especificamente o método Wp, foi necessário adicionar as informações de saída *output* para a MEF. Conforme supracitado a saída da MEF é uma confirmação da entrada e foi inserida de forma manual. Na Figura 3 é possível visualizar a MEF final que foi resultado de todas as uniões de estados efetuadas contendo as saídas.

Figura 3. Resultado Output

4.5. Aplicação do Método Wp

A aplicação do método Wp [Fujiwara et al. 1991] resultou nas seguintes sequências de testes: **Sequência 1 :** [Vazio, Abrir]; **Sequência 2 :** [Abrir, Vazio, Abrir]; **Sequência 3 :**

[Abrir, Vazio, Vazio]; **Sequência 4 :** [Abrir, Fechar, Abrir]; **Sequência 5 :** [Abrir, Depositar, Remover, Abrir];**Sequência 6 :** [Abrir, Depositar, Remover, Vazio];**Sequência 7 :** [Abrir, Depositar, Fechar, NaoVazio, Abrir] ;**Sequência 8 :** [Abrir, Depositar, Fechar, NaoVazio, Vazio] ;**Sequência 9 :** [Abrir, Depositar, Fechar, NaoVazio, Fechar]. Importante notar que para aplicação do método é necessário a MEF completamente especificada. Isso pode ser alcançado gerando transições vazias para as transições inexistentes.

4.6. Geração de *script* JUnit

Utilizando as sequências geradas e as saídas da MEF criada anteriormente, foi possível criar um *parser* para gerar os *drivers* de teste. Para isto, foi utilizada a biblioteca JUnit. Para cada entrada em uma sequência de testes, foi criado um *assertEquals* comparando se o resultado da ação é igual à saída especificada na MEF. Abaixo apresentamos um exemplo de *Script* para a sequência 5: [Abrir, Depositar, Remover, Abrir]. As demais sequências podem ser acessadas em: https://goo.gl/mDT1Ee.

```
@Test
public static void _Test_Sequence_5(){
assertEquals("Aberto", Abrir());
assertEquals("Depositado", Depositar());
assertEquals("Removido", Remover());
assertEquals("Aberto", Abrir());}
```

5. Considerações finais e Trabalhos Futuros

Neste trabalho apresentamos uma estratégia para a geração de *scripts* de testes a partir de cenários Gherkin. A estratégia utiliza o algoritmo QSM em conjunto com o método de geração de sequências de testes Wp para aplicação da técnica de MBT. Este trabalho está em fase inicial, sendo desenvolvido em parceria com uma empresa e estando em fase de testes com cenários maiores do que o utilizado como exemplo. Desta forma, considera-se como principal ameaça à validade dos resultados o fato do teste ter sido executado com uma *feature* muito simples, o que pode não apresentar os mesmos resultados do que em uma aplicação real. Além disso, também identificamos duas limitações na estratégia, sendo elas a inserção manual das saídas (*outputs*) na MEF e a alta participação do usuário respondendo as perguntas de validação das *Queries*. Ambas as situações pontuadas são consideradas limitações por envolverem trabalho repetitivo do usuário, o que consome tempo e pode prejudicar a aplicabilidade da proposta.

Projeta-se como trabalhos futuros para este estudo, a automação da geração de saídas (*outputs*) para a MEF, bem como a otimização do método para reduzir ou eliminar as perguntas feitas para o usuário na aceitação das *Queries*. Além disso, está em fase de estruturação a realização de um estudo de caso com uma aplicação real.

6. Agradecimentos

Agradecemos a CAPES pelo financiamento dos estudos através da bolsa de doutorado de Aline Zanin e a empresa DBServer pelo financiamento parcial deste trabalho.

Referências

Cucumber (2017). Gherkin. Disponível em: https://cucumber.io/docs/reference Acesso em 08 de setembro de 2017.

Dupont, P., Lambeau, B., Damas, C., and Lamsweerde, A. v. (2008). The qsm algorithm and its application to software behavior model induction. *Applied Artificial Intelligence*, 22(1-2):77–115.

El-Far, I. K. and Whittaker, J. A. (2002). Model-based software testing. *Encyclopedia of Software Engineering*.

Entin, V., Winder, M., Zhang, B., and Claus, A. (2015). A process to increase the model quality in the context of model-based testing. In *Proceedings of the IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops*.

Fujiwara, S., Khendek, F., Amalou, M., Ghedamsi, A., et al. (1991). Test selection based on finite state models. *IEEE Transactions on Software Engineering*, 17(6):591–603.

Li, N., Escalona, A., and Kamal, T. (2016). Skyfire: Model-based testing with cucumber. In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation*, pages 393–400.

Seijas, P. L., Thompson, S., and Francisco, M. Á. (2016). Model extraction and test generation from junit test suites. In *Proceedings of the 11th International Workshop on Automation of Software Test*, pages 8–14. ACM.

Sivanandan, S. and B, Y. C. (2014). Agile development cycle: Approach to design an effective model based testing with behaviour driven automation framework. In *Proceedings of the 20th Annual International Conference on Advanced Computing and Communications*.

Wynne, M., Hellesoy, A., and Tooke, S. (2017). *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*. Pragmatic Programmers. Pragmatic Bookshelf.

Usa-DSL: Um Framework para Avaliação de Usabilidade de Linguagens Específicas de Domínio*

Ildevana Poltronieri Rodrigues[1], Márcia de Borba Campos[1], Avelino F. Zorzo[1], Maicon Bernardino[2], Elder de Macedo Rodrigues[2]

[1]Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Av. Ipiranga, 6681, Partenon – Porto Alegre – RS – Brasil

[2]Universidade Federal do Pampa (UNIPAMPA)
Laboratory of Empirical Studies in Software Engineering (LESSE)
Av. Tiarajú, 810, Ibirapuitã – Alegrete, RS – Brasil

ildevana@gmail.com, marciabcampos@hotmail.com, avelino.zorzo@pucrs.br
bernardino@acm.org, elderrodrigues@unipampa.edu.br

Abstract. *The lack of well-defined methods to assess usability criteria for Domain-Specific Languages (DSL) is a problem for DSL developers. Therefore, software engineers usually evaluate usability in an ad-hoc and subjective manner. In this paper, we present the first version of a DSL usability assessment framework, called Usa-DSL. We also present a preliminary evaluation of this framework.*

Resumo. *Estudos mostraram não haver métodos bem definidos para avaliar critérios de usabilidade em linguagens específicas de domínio (Domain-Specific Languages - DSL). Desta forma, engenheiros de software avaliam de forma experimental, até mesmo subjetivamente, se uma DSL é fácil de ser utilizada e de fácil entendimento, por exemplo. Este trabalho descreve a primeira versão de um framework de avaliação de usabilidade de DSL denominado Usa-DSL. Este artigo também apresenta uma avaliação preliminar do Usa-DSL.*

1. Introdução

Em desenvolvimento de software geralmente Linguagens de Propósito Geral (*General-Purpose Languages - GPL*) são utilizadas, por exemplo, Java, C#, Ruby, Python, dentre outras. Porém, existem algumas aplicações de domínios específicos que se beneficiam de linguagens com características específicas, que podem gerar maior desempenho, representações diferenciadas, abstrações no domínio do negócio, melhor comunicação entre os desenvolvedores e os analistas de negócio, entre outros aspectos. Para isso, os engenheiros de sistemas buscam, por meio do desenvolvimento de diferentes linguagens, facilitar cada vez mais a concepção de novos conceitos e teorias. Neste sentido, se verifica a existência de linguagens de programação que, aliadas à complexidade do desenvolvimento de aplicações, podem apresentar certas dificuldades às empresas no que se refere à modelagem, implementação, avaliação e manutenção de sistemas, e, com isso, causar diferentes problemas e comprometer a qualidade do projeto dos mesmos.

*Agradecimentos: DELL e CAPES pela concessão de bolsas de mestrado e doutorado.

Sendo assim, para minimizar dificuldades inerentes ao desenvolvimento de aplicações, destaca-se a possibilidade de uso do desenvolvimento dirigido por modelos (*Model-Driven Development* - MDD) [Mernik et al. 2005], que, a partir de modelos do sistema, permite a geração de automática de códigos. Em específico para este artigo, a abordagem de MDD está baseada em Linguagens Específicas de Domínio (*Domain-Specific Languages - DSLs*) [Fowler 2010]. Pode-se destacar que tem se investido em diferentes propósitos para o desenvolvimento de DSL, que variam conforme o domínio da aplicação. Por exemplo, existem DSLs que podem ser aplicadas nos domínios de robótica [Conrado 2012], anomalias em arquitetura de software [Albuquerque et al. 2015] e teste de desempenho [Bernardino et al. 2016], entre outros. O diferencial entre as DSLs é definido pela sintaxe e semântica, que são determinadas pelo domínio do problema a ser resolvido. Ainda, um domínio pode utilizar mais de uma DSL para modelar as suas características, sem haver sobreposição.

Desta forma, acredita-se que o atendimento a recomendações de usabilidade em uma DSL possa aumentar seu uso no desenvolvimento de sistemas. Para que os usuários de domínios específicos possam utilizar estas linguagens de forma mais independente e com facilidade, os engenheiros de linguagens precisam preocupar-se com critérios de usabilidade e ainda com a diversidade de contextos e domínios, levando em conta a satisfação no uso durante a interação destes usuários e os diferentes sistemas. Sendo assim, de posse de diferentes conceitos sobre DSL, Teste de Desempenho, Estudos Empíricos em Engenharia de Software e Avaliação de Usabilidade, este artigo visa apresentar o *Framework* Usa-DSL para Avaliação de Usabilidade de Linguagens Específicas de Domínio. Também apresenta a metodologia e resultados de uma avaliação preliminar, que foi realizada utilizando a técnica de Entrevista.

Este artigo está organizado da seguinte forma: na Seção 2 são apresentados trabalhos relacionados referentes a DSLs e os métodos e técnicas aplicadas na avaliação de usabilidade no contexto de DSL. A Seção 3 descreve o *Framework* Usa-DSL, na qual será apresentada sua estrutura que é composta por etapas, fases e atividades. A Seção 4 apresenta uma avaliação preliminar realizada por meio de uma entrevista. Finalmente, a Seção 5 apresenta a conclusão e os trabalhos futuros.

2. Trabalhos Relacionados

Com o propósito de embasar conceitos e definições utilizadas no desenvolvimento do *Framework* Usa-DSL e unificar o entendimento dos tópicos relacionados ao estudo, são apresentados alguns trabalhos identificados na revisão sistemática [Rodrigues et al. 2017]. Estes artigos estão diretamente relacionados ao processo de avaliação de DSL e consideram questões referentes à usabilidade, bem como técnicas e abordagens de usabilidade que foram utilizadas nas avaliações.

O trabalho de Barisic *et al.* [Barisic et al. 2014] considera fundamental avaliar critérios de qualidade de uso no desenvolvimento de uma DSL, para que esta possa melhor atender as necessidades do usuário final. O estudo indica que a avaliação de critérios de usabilidade deve ser aplicada durante todas as etapas do processo de concepção, tais como: projeto, análise, desenvolvimento e avaliação de uma DSL. Para melhor atender o usuário final de uma DSL, Barisic *et al.* [Barisic et al. 2014] sugere que a avaliação de Interação Humano Computador (IHC) inclua métodos quantitativos e qualitativos, tais

como: observação do usuário e entrevistas.

Com relação a importância dada a usabilidade de DSL, o estudo de Albuquerque [Albuquerque et al. 2015] destaca a preocupação com a realização de uma análise quantitativa no que se refere a aspectos de usabilidade. Para isso, realizaram uma instanciação do *Cognitive Dimensions of Notations framework* (CDN), proposto por Blackwelland and Green em 2003. Este *framework* foi desenvolvido para discutir o desenvolvimento e avaliação de ferramentas por *designers*, já a sua instância CDN foi desenvolvida para detectar aspectos de usabilidade em artefatos de manutenção de tarefas em DSL. O artigo ainda relata que uma DSL compreende quatro aspectos importantes: expressividade, concisão, integração e desempenho.

Do mesmo modo, Barisic *et al.* [Barisic et al. 2011] define um modelo cognitivo para a linguagem baseado no contexto de cenários de usuários. As atividades cognitivas envolvidas na linguagem são: aprendizado da sintaxe e semântica, composição da sintaxe necessária para desempenhar uma função, a compreensão da sintaxe desenvolvida por outra pessoa, depuração da sintaxe e a modificação de uma função escrita pelo próprio desenvolvedor ou por outro. Referente a avaliação desta DSL foi realizado um estudo de caso com 6 participantes, que utilizaram a DSL Pheasant (PHysicist's EAsy Analysis Tool) e uma *baseline* GPL em C++. Ainda com relação ao método de avaliação, Teruel *et al.* [Teruel et al. 2014], como os autores Barisic *et al.* [Barisic et al. 2012] e outros apresenta a condução de um experimento para avaliar a usabilidade, no caso de Teruel o domínio da linguagem se refere a modelagem colaborativa de requisitos de sistema.

A avaliação de usabilidade no estudo de Sinha e Smidts [Sinha and Smidts 2006] está baseado em quatro das dez heurísticas propostas por Nielsen, sendo que para cada uma das heurísticas avaliadas foram definidas métricas. Sendo estas: aprendizado, eficiência, índice de erros, satisfação e facilidade. No final, a avaliação quantitativa considerou cada uma das métricas definidas para avaliar a DSL.

Quanto às técnicas e métodos, estudos apresentam a adaptação ou utilização de conjuntos de métricas de usabilidade. Apesar dos esforços, ainda é necessário um grande esforço para conceber uma DSL de maneira fácil, com maior compreensão e expressividade em relação ao domínio que se pretende representar. Além disto, é necessário também desenvolver processos, métodos e técnicas que auxiliem em uma melhor avaliação de usabilidade de DSLs.

3. Usa-DSL: *Framework* para Avaliação de Usabilidade de DSL

Pensando na necessidade de avaliar linguagens para domínio específicos, buscou-se na literatura por meio da execução de uma Revisão Sistemática da Literatura - RSL [Rodrigues et al. 2017] e da consulta de bibliografias consolidadas na área de IHC [Rogers et al. 2013] [Barbosa and da Silva 2010] [Stone et al. 2005] [Nielsen 1993], diferentes métodos e *frameworks* que realizassem avaliação de usabilidade. Na RSL [Rodrigues et al. 2017] foram identificado diferentes estudos, que discutem e apresentam preocupações relacionadas a aspectos de usabilidade e destacam a importância deste tipo de avaliação durante o projeto, análise e desenvolvimento das DSLs, porém nenhum *framework* ou método para realização avaliação de usabilidade foi apresentado. Sendo assim, este trabalho propõe um *framework* para avaliação de usabilidade de DSL denominado Usa-DSL, que é constituído por etapas, fases e atividades, conforme descrito

nas próximas subseções.

3.1. Estrutura do *Framework* Usa-DSL

A estrutura do *framework* Usa-DSL é um legado do processo de ciclo de vida de projeto [Stone et al. 2005]. Está organizada em etapas, fases e atividades, como pode ser visto na Figura 3.1. As etapas estão definidas em: Perfil do Avaliador, Responsabilidades Éticas e Legais, Tipo de Dados, Métodos Empíricos (ES), Método de Avaliação (IHC), Métricas, Instrumentos de Coleta, Instrumentos de Instrução e Treinamento, Condução da Avaliação, Empacotamento dos Dados e Relatório da Avaliação. Já as fases são compostas por: Definição, Execução, Análise e Resultados. As atividades são formadas por um conjunto de 32 conceitos que são distribuídas entre as etapas e fases, e estão dispostas no *link* `http://tiny.cc/ERES2017_Usa-DSL`.

A estrutura do *framework* Usa-DSL foi planejada para ser utilizada a partir da necessidade de cada avaliação. Desta forma, por exemplo, o avaliador pode escolher iniciar o planejamento de sua avaliação a partir do perfil do avaliador, ou pelo tipo de dado (qualitativo ou quantitativo), pelo tipo de método empírico ou de avaliação que deseja realizar. Com isso, o avaliador tem a liberdade de desenvolver a avaliação conforme o tipo de linguagem que deseja avaliar e de uma forma iterativa, na qual ele poderá seguir diferente instâncias/caminhos na avaliação da DSL.

3.2. Etapas do *Framework* Usa-DSL

O *framework* Usa-DSL é constituído por 11 etapas, que podem ser selecionadas e executadas de acordo com o tipo de linguagem que será avaliada. As etapas do *Framework* Usa-DSL são descritas a seguir (ver Figura 3.1):

Etapa 1 *Perfil do Avaliador*: define os perfis de avaliador, que podem ser Experiente em IHC, Analista do Domínio, Desenvolvedor do Domínio ou Testador do Domínio, por exemplo. Estes perfis foram definido a partir do Mapeamento da RSL, no qual os autores [Cuenca et al. 2015], [Ewais and De Troyer 2014], [Gibbs et al. 2015], [Albuquerque et al. 2015], [Barisic et al. 2014] descrevem ter utilizado especialistas no domínio para realização de avaliações.

Etapa 2 *Responsabilidades Éticas e Legais*: toda pesquisa que envolve seres humanos deve atender a questões éticas. Sendo assim, o Usa-DSL está baseado no *framework* DECIDE, que sugere que seja fornecido ao participante um documento, no qual estará explicado o objetivo da pesquisa, e ainda certifica a segurança do participante prevendo o sigilo dos seus dados não associando suas respostas ao seu perfil e nem relatando em qualquer documento associado a pesquisa. Também está baseado no código da Association for Computing Machinery (ACM) [1].

Etapa 3 *Tipo de Dados*: os dados quantitativos são resultados numéricos que prevem a quantidade de respostas atribuídas para determinado item de uma questão. Já os dados qualitativos constituem informações subjetivas referentes a opinião do participante sobre o objeto estudado. Estes dados buscam prever que tipo de informação o avaliador pretende obter, Albuquerque *et al.* [Albuquerque et al. 2015] sugere o uso dos dois tipos de dados, a fim de obter

[1] https://www.acm.org/about-acm/acm-code-of-ethics-andprofessional

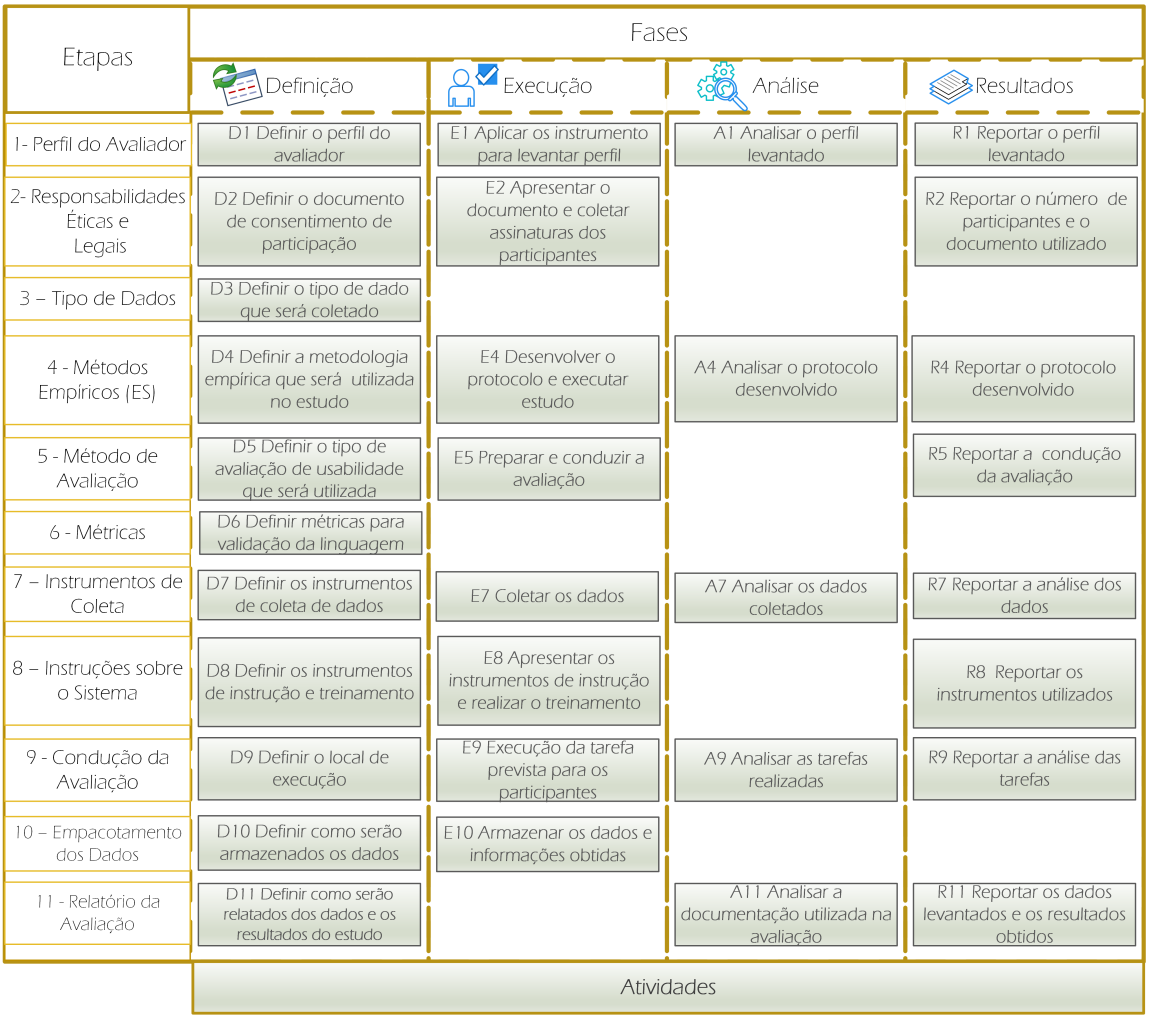


Figura 1. Estrutura da Versão Preliminar do Framework Usa-DSL

uma visão mais abrangente e completa sobre a opinião dos participante. Já Barisic *et al.* [Barisic et al. 2014] utiliza em seus experimentos dados quantitativos e considera que sejam suficientes para o objetivo da sua pesquisa. Desta forma, o pesquisador deve analisar a linguagem a ser avaliada e definir qual tipo de dado melhor atende aos objetivos de sua avaliação.

Etapa 4 *Métodos Empíricos (Engenharia de Software)*: Wohlin *et al.* [Wohlin 2014] sugerem o uso de *survey*, estudo de caso e experimento. Estes métodos podem ser definidos a partir do objeto de estudo, do perfil dos participantes da avaliação e/ou dos dados que se pretende coletar.

Etapa 5 *Método de Avaliação*: podem ser realizados junto a usuários finais (observação de uso, teste de usabilidade, por exemplo) ou a especialistas (avaliação heurística, Percurso cognitivo - *Cognitive Walkthrough*, por exemplo).

Etapa 6 *Métricas*: foram definidas a partir da RSL ([Rodrigues et al. 2017]). São sugeridas as seguintes características: compreensão/aprendizado, facilidade no uso, esforço/tempo de conclusão, complexidade percebida, taxa de conclusão, taxa de erro na tarefa, eficiência e eficácia. Estas métricas devem orientar a construção dos instrumentos de coleta de dados, a serem utilizados durante a avaliação.

Etapa 7 *Instrumentos de Coleta*: estão baseados em [Rodrigues et al. 2017], [Rogers et al. 2013] e [Rouly et al. 2014]. Sugerem-se *checklist* para avaliação heurística, *checklist* ergonômico, questionários, entrevistas, observação de uso e gravação de ação do usuário.

Etapa 8 *Instruções sobre o Sistema*: conforme Wohlin *et al.* [Wohlin 2014], as instruções do sistemas constituem em manual de uso, apresentação dos instrumentos, e tarefa a ser executada. Estes materiais devem ser distribuídos e utilizados ao realizar um experimento, *survey* ou estudo de caso e servem para orientar e esclarecer à dúvidas dos participantes.

Etapa 9 *Condução da Avaliação*: é nessa etapa que se aplica o que foi definido nas etapas anteriores. Sendo assim, é necessário que as demais etapas tenham sido cuidadosamente definidas e testadas antes de envolver os participantes da avaliação. Caso não tenha bem definido o que deve ser feito e como será executado, o pesquisador poderá perder a amostra e invalidar os dados coletados. Por esta razão, os dados que deseja coletar, a tarefa e os materiais que serão aplicados devem ter passado por um teste piloto anteriormente e somente após obter uma versão estável, a avaliação poderá ser executada. Outro aspecto se refere à confirmação dos participantes, de forma que o avaliador deve buscar garantir que o número necessário de participantes compareceram às atividades da fase de execução.

Etapa 10 *Empacotamento dos Dados*: ao finalizar a avaliação aconselha-se que os materiais utilizados no treinamento e na coleta dos dados sejam armazenados em local seguro e de fácil acesso, para que o estudo possa ser replicado quando necessário e assim conduzir uma reavaliação da linguagem que está sendo testada. Ainda, o armazenamento dos dados coletados também são úteis para uma posterior comparação com avaliações que poderão vir a ser conduzidas até a finalização e/ou estabilização da linguagem.

Etapa 11 *Relatório da Avaliação*: deve seguir o método de avaliação escolhido. Cada método gera um tipo de documento para relatar a avaliação, este documento deve conter a descrição da DSL analisada, objetivo da avaliação, descrição do método aplicado, perfil dos participantes/especialistas, instrumentos aplicados, dados coletados, análise dos dados, problemas identificados, soluções propostas, dentre outros.

3.3. Fases do *Framework Usa-DSL*

As fases são compostas pela definição, execução, análise e resultados da avaliação, como podem ser observadas na Figura 3.1. Cada Fase possui um conjunto de atividades, que se relacionam à etapa que está sendo desenvolvida.

Fase 1 *Definição*: esta fase tem como propósito a organização da pesquisa por parte do avaliador, a definição do que será realizado para avaliar a DSL. Nesta fase devem ser definidos e construídos documentos, tomadas decisões sobre os dados que se deseja obter, qual o perfil de usuário que fará parte da avaliação. Ou seja, nesta fase se estrutura e se planeja a avaliação da DSL.

Fase 2 *Execução*: nesta fase se concretiza a avaliação. São aplicados os documentos para coleta dos dados e conduzida a avaliação, conforme protocolo que fora definido.

Fase 3 *Análise*: terá como objetivo realizar a análise dos artefatos gerados tanto na fase de Definição como na fase de Execução. Na fase de definição está análise é feita

para que sejam ajustados os documentos e decisões tomadas quanto ao desenvolvimento da avaliação. Sobre a fase de Execução à análise se concentra nos dados e tarefas geradas durante a realização da avaliação.

Fase 4 *Resultados*: nesta fase, o avaliador realiza o registro do protocolo utilizado, dos artefatos gerados, dados analisados. Estas informações podem ser reportadas por meio de relatório no caso da indústria ou artigo científico no caso de pesquisa acadêmica. O formato do relatório também depende do tipo de avaliação que foi adotada.

3.4. Atividades do *Framework* Usa-DSL

As atividades do *framework* Usa-DSL são formadas por um conjunto de teorias e práticas utilizadas para realização de avaliação de software pelas áreas de Interação Humano-Computador e Engenharia de Software. Os conceitos utilizados foram identificados a partir de uma Revisão Sistemática da Literatura [Rodrigues et al. 2017], e autores das áreas envolvidas. O quadro com a estrutura das atividades pode ser visto na Figura 3.1 e suas descrições no repositório online, por meio do *link* `http://tiny.cc/ERES2017_Usa-DSL`.

4. Avaliação Preliminar

Esta avaliação foi realizada por meio da técnica de entrevista, e teve como objetivo apresentar o *framework* Usa-DSL, e obter a opinião de um desenvolvedor de DSL. Este desenvolvedor criou uma DSL para os domínios de visualização de dados geoespaciais (DSL GMaVis) e abstração do paralelismo em aplicações de *streaming* (DSL SPar). No primeiro momento desta entrevista, foi solicitado ao participante que expressasse a sua concordância em participar, por meio do Termo de Consentimento Livre e Esclarecido (TCLE). Após, foram repassadas a ele, a descrição do *framework* Usa-DSL, um exemplo de uso, e as informações contidas no glossário. Estes e outros documentos usados na avaliação estão disponíveis para consulta no repositório online.

Após os procedimentos iniciais, foi solicitado ao participante que respondesse a um questionário de caracterização, que pode ser acessado no *link* `http://tiny.cc/ERES2017_USA-DSL`. Com base nas respostas do entrevistado, foram destacadas as seguintes informações sobre o perfil do participante: atua profissionalmente como pesquisador e professor por um período de 7 anos; possui 2 anos de experiência em análise, desenvolvimento e teste de DSL; não possui experiência quanto à avaliação de usabilidade; já participou de métodos experimentais empíricos, e *i.e.* realizou experimento controlado e *survey*.

Ao ser questionado sobre quais critérios de usabilidade considera importante no desenvolvimento de uma DSL, o participante citou: efetividade, eficiência, satisfação de uso, prevenção de erros, facilidade de uso, facilidade de aprender como usar, facilidade de lembrar como usar, segurança, e, ainda, acrescentou o critério "desempenho". Como justificativa para importância de tais critérios, afirmou que se o usuário não estiver satisfeito com a DSL que foi projetada, mesmo que possua todas as funcionalidades que necessita, buscará por uma ferramenta que atenda a critérios de usabilidade.

Em seguida, o avaliador apresentou ao participante o exemplo de uso, DSL Canopus, bem como esclareceu dúvidas sobre a composição e funcionamento do *framework*.

Ao final, quando o participante considerou que estava apto para responder ao questionário, o avaliador fez as perguntas, que foram gravadas em áudio e registradas em um questionário escrito. Dentre as repostas desta entrevista, destacam-se: **a) A importância em avaliar critérios de usabilidade em DSLs gráficas:** o participante informou ser importante avaliá-los, e afirmou que o usuário precisa se sentir satisfeito para continuar usando a DSL; caso contrário, irá buscar outra ferramenta, mesmo que não seja para o domínio específico. **b) Durante o processo de desenvolvimento de uma DSL o *framework* Usa-DSL poderia ser usado:** o participante afirmou que usaria o Usa-DSL. E justifica que, por não ter conhecimentos na área de IHC, o *framewok* o ajudaria no processo de avaliação de sua DSL, pois oferece uma metodologia a ser seguida. **c) O entendimento do *framework* é fácil:** o participante confirmou que a descrição é de fácil entendimento, mas não soube precisar se seria fácil aplicá-lo. Assim, o participante considerou o *framewok* Usa-DSL de fácil compreensão, mas não soube, naquele momento, se seria de fácil uso. **d) O uso do *framework* facilita o planejamento, execução e análise da avaliação de DSLs:** o participante afirmou que o *framework* Usa-DSL oferece informações de forma clara. Porém, não saberia avaliar se, de fato, seria fácil de ser utilizado, embora as descrições pareçam facilitar a avaliação. Novamente, percebe-se a preocupação do participante no que se refere à facilidade de uso do *framewok*. **e) O uso do *framework* torna a avaliação de DSL mais produtiva:** embora não tenha utilizado o *framework* Usa-DSL, ainda assim, achou que seria mais produtivo o seu uso na construção e execução da avaliação de uma DSL. E complementa que, ao utilizá-lo, não precisaria aprender sobre uma área diferente de sua pesquisa (no caso, IHC), e que certamente levaria mais de um ano para compreender os conceitos. Por fim, justifica que, esta curva de aprendizagem é o que o afasta de realizar tais avaliações. **f) Informações que adicionaria no *framework*:** informações sobre recrutamento de participantes, pois considerou que é difícil encontrar pessoas adequadas para participar de avaliações de DSL, por necessitar de um perfil mais específico. **g) Informações que retiraria do *framework*:** informou que não excluiria informação, que estão sendo apresentadas no Usa-DSL. **h) Perfis de usuário para avaliar uma DSL gráfica:** primeiramente indica a consulta a alguém com experiência em IHC, para resolver problemas básicos de interface com o usuário. E, após as modificações apontadas pelo avaliador de IHC, aplicaria uma segunda avaliação com usuários do domínio. O participante acredita que ao realizar a avaliação com alguém experiente em IHC, evitaria que o usuário do domínio realizasse várias avaliações. Relatou, que ao utilizar os dois perfis seria mais produtivo, pois é uma tarefa difícil conseguir voluntários para realização de tais avaliações.

5. Considerações Finais e Trabalhos Futuros

Nos últimos anos é possível identificar uma aumento em relação ao desenvolvimento de DSLs. A literatura menciona que, para as DSLs se tornarem mais atrativas e com maior qualidade de uso, é necessário dedicar mais esforço na avaliação de usabilidade destas DSLs [Quiñones and Rusu 2017].

Neste sentido, busca-se com o *framework* Usa-DSL auxiliar os engenheiros de linguagem com a avaliação das DSLs desenvolvidas e minimizar dificuldades inerentes a utilização das aplicações construídas. O *framework* Usa-DSL ainda busca tornar a avaliação de DSLs mais fácil, produtiva e, consequentemente, procura alcançar critérios de usabilidade e de satisfação de uso [Sinha and Smidts 2006]. Com o uso do Usa-DSL,

busca-se facilitar o planejamento, execução e análise da avaliação de DSLs, oferecendo informações de forma clara e objetivas para que a equipe de avaliação (analista, desenvolvedor, testador) possa planejar e avaliar a DSL.

Após a avaliação preliminar apresentada na Seção 4, foi realizado um Grupo Focal envolvendo participantes com experiência em IHC, Engenharia de Software e Teste de Desempenho. Esta avaliação teve como objetivo apresentar o *framework* Usa-DSL, e obter a opinião dos participantes quanto a sua clareza, facilidade no uso e entendimento. Os resultados obtidos e as alterações sugeridas serão apresentadas em outro artigo, que está em processo de submissão. Referente aos trabalhos futuros, pretende-se aplicar o *framework* Usa-DSL na avaliação de diferentes DSLs, a partir de diferentes métodos de avaliação de usabilidade; refinar os materiais informativos do Usa-DSL, incluindo exemplos de uso; utilizar o *framework* Usa-DSL com diferentes perfis de avaliadores de forma a refinar o *framework* e sua descrição.

Referências

Albuquerque, D., Cafeo, B., Garcia, A., Barbosa, S., Abrahão, S., and Ribeiro, A. (2015). Quantifying usability of domain-specific languages: An empirical study on software maintenance. *Journal of Systems and Software*, 101:245 – 259.

Barbosa, S. and da Silva, B. (2010). *Interação humano-computador*. Campus.

Barisic, A., Amaral, V., Goulão, M., and Aguiar, A. (2014). Introducing usability concerns early in the dsl development cycle: Flowsl experience report. In *Proceedings of the 17th International Conference on Model Driven Engineering Languages and Systems*.

Barisic, A., Amaral, V., Goulão, M., and Barroca, B. (2011). Quality in use of domain-specific languages: A case study. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools*, pages 65–72, New York, NY, USA. ACM.

Barisic, A., Amaral, V., Goulãoo, M., and Barroca, B. (2012). Evaluating the usability of domain-specific languages. *Formal and practical aspects of domain-specific languages: Recent developments*.

Bernardino, M., Zorzo, A., and Rodrigues, E. (2016). Canopus: A Domain-Specific Language for Modeling Performance Testing. In *Proceedings of the 9th International Conference on Software Testing, Verification and Validation (ICST)*, pages 157–167. IEEE.

Conrado, D. B. F. (2012). Abordagem pata criação de linguagem específica de domínio para robótica móvel. Master's thesis, Universidade Federal de São Carlos, São Paulo.

Cuenca, F., Bergh, J. V. d., Luyten, K., and Coninx, K. (2015). A user study for comparing the programming efficiency of modifying executable multimodal interaction descriptions: A domain-specific language versus equivalent event-callback code. In *Proceedings of the 6th Workshop on Evaluation and Usability of Programming Languages and Tools*, pages 31–38, New York, NY, USA. ACM.

Ewais, A. B. and De Troyer, O. (2014). A usability evaluation of graphical modelling languages for authoring adaptive 3d virtual learning environments. In *Proceedings of the 6th International Conference on Computer Supported Education*, pages 459–466.

Fowler, M. (2010). *Domain Specific Languages*. Addison-Wesley Professional, 1 edition.

Gibbs, I., Dascalu, S., Harris, F. C., and Jr. (2015). A separation-based ui architecture with a dsl for role specialization. *Journal of Systems and Software*, 101:69 – 85.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4):316–344.

Nielsen, J. (1993). *Usability Engineering*. Morgan Kaufmann, San Francisco, USA.

Quiñones, D. and Rusu, C. (2017). How to develop usability heuristics: A systematic literature review. *Computer Standards & Interfaces*, 53:89–122.

Rodrigues, I., Campos, M., and Zorzo, A. (2017). Usability evaluation of domain-specific languages: a systematic literature review. pages 522–534.

Rogers, Y., Sharp, H., and Preece, J. (2013). *Design de interação: além da interação humano-computador*. Bookman.

Rouly, J. M., Orbeck, J. D., and Syriani, E. (2014). Usability and suitability survey of features in visual ides for non-programmers. In *Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools*, pages 31–42, New York, NY, USA. ACM.

Sinha, A. c. and Smidts, C. d. (2006). An experimental evaluation of a higher-ordered-typed-functional specification-based test-generation technique. *Empirical Software Engineering*, 11(2):173–202.

Stone, D., Jarrett, C., Woodroffe, M., and Minocha, S. (2005). *User Interface Design and Evaluation*. Interactive Technologies. Elsevier Science.

Teruel, M. A., Navarro, E., López-Jaquero, V., Montero, F., and González, P. (2014). A cscw requirements engineering case tool: Development and usability evaluation. *Information and Software Technology*, 56(8):922 – 949.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 38. ACM.

Mapeamento Sistemático de Gamificação em Engenharia de Software

Gabriel Belinazo[1], Lisandra Manzoni Fontoura[1]

[1]Laboratório de Computação Aplicada – Universidade Federal de Santa Maria (UFSM)
Santa Maria – RS – Brazil

{gbelinazo,lisandra}@inf.ufsm.br,

Abstract. *Gamification aims to improve engagement, motivation and performance by adding gaming mechanisms to other domains. The objective of this work is to map the state of the art of gamification in software engineering. Systematic mapping was conducted to find studies conducted in the area, and to classify them in terms of software process, research method, publication vehicle and gamification technique. Finally, 45 studies were selected to form the basis of this systematic mapping. Finally, although many studies do not present validation, those who do show an improvement in engagement, motivation and performance in the processes in which they were implemented.*

Resumo. *A gamificação objetiva a melhoria do engajamento, da motivação e do desempenho da equipe, adicionando mecanismos de jogos em outros domínios. O objetivo desse trabalho é mapear o estado da arte da gamificação em engenharia de software. O mapeamento sistemático foi conduzido objetivando principalmente encontrar estudos desenvolvidos na área, e classificá-los em termos de processo de software, método de pesquisa, veículo de publicação e técnica de gamificação. Por fim foram selecionados 45 estudos para compor a base deste mapeamento sistemático. Ao fim, apesar de muitos estudos não apresentarem validação, os que o fazem, mostram uma melhoria do engajamento, motivação e performance nos processos em que foram implementados.*

1. Introdução

Existe um alto percentual de insucesso entre os projetos de software devido a uma coleção complexa de motivos. Sabe-se que a falta de aderência da equipe ao processo de desenvolvimento de software é um desses. (Beecham *et al.,*2008) Por diversas vezes, equipes de desenvolvimento de software creem que os processos de software são burocráticos, que duplicam as informações e que limitam a criatividade, seja em um contexto planejado ou ágil. Além disso, muitos desenvolvedores têm dificuldades de seguir um processo de software, haja vistas os desafios da atividade de desenvolvimento. (FOWLER, 2001)

Segundo MEDEIROS (2015), a utilização de elementos de gamificação "não só pode ajudar a envolver as equipes a fazer o seu trabalho, mas também pode ajudar a monitorar, controlar e melhorar o desenvolvimento". Gamificação trata-se do uso de elementos de *game design* em contextos diferentes dos de jogos, com o objetivo de alterar o comportamento, aumentar o engajamento e a motivação de pessoas na realização de atividades. Para DUBOIS (2013), gamificação foi inicialmente adotada para aumentar o engajamento de consumidores que recebem pontos e medalhas no uso de serviços. Além

dos pontos e medalhas, outra técnica de gamificação amplamente utilizada é o indicador de progresso. Nos últimos anos, gamificação tem sido adotada sistematicamente como uma possível solução ao desinteresse de alunos no desenvolvimento de cursos e atividades educacionais, assim tem aparecido como tendência em diversos estudos. Há muitas razões pelas quais gamificação tornou-se rapidamente uma *buzzword*, entre elas KUMAR (2013) cita: a natureza mutável da informação, a entrada na força de trabalho de nativos da era digital – uma nova geração que cresceu jogando videogames, a ampla adoção de mídias sociais, a adoção de tecnologias móveis. Desta forma, temos um *design* de produtos de software que crescentemente é centrado no jogador.

Um mapeamento sistemático da área conduzido em 2015, por PEDREIRA *et al*., constatou que a gamificação em engenharia de software encontra-se em um estágio inicial, e que um grande número de estudos apresenta apenas a utilização de mecanismos básicos da gamificação, por exemplo um sistema de pontuação simples. Alguns trabalhos, como o de MEDEIROS (2015), citam gamificação na área de processos de desenvolvimento de software. Este estudo apresentou uma abordagem para aumentar a aderência na implantação de processos de software nas empresas. Justifica-se a necessidade da condução dessa pesquisa para que se verifique se as condições descritas por PEDREIRA *et al*. (2015) se mantêm. O estudo de PEDREIRA *et al.* (2015) mapeou estudos publicados até junho de 2014, e posteriormente, analisou-os a partir dos seguintes critérios: a área de processo de software abordada, os elementos de gamificação utilizados, o tipo de método de pesquisa seguido e o veículo em que os estudos foram publicados. Também foi conduzida uma avaliação subjetiva desses estudos a fim de avaliá-los em termos de metodologia, evidências empíricas, integração com a organização e replicação. Dessa forma, o apresentou como resultado a seleção de 29 artigos, publicados entre janeiro de 2011 e junho de 2014. Em sua maioria, o ponto focal apresentado era o desenvolvimento de software. Bem como, verificou-se que, em sua maioria, a aplicação de técnicas de gamificação eram rudimentares, e se limitavam, muitas vezes, a utilização de pontos e emblemas. E poucos, segundo os autores, fornecem evidências do impacto da gamificação.

2. Método de pesquisa

O objetivo deste estudo é identificar o estado da arte de gamificação em engenharia de software, mais especificamente, sua aplicação em processos de software, revendo os resultados apontados pelo estudo de PEDREIRA *et al*.(2015). Para que assim seja possível conhecer as lacunas e as oportunidades para futuras pesquisas com o tema. Nessa seção, apresenta-se a metodologia, bem como o planejamento de cada etapa do estudo conduzido.

2.1 Questões de pesquisa

Para este estudo, seguiu-se a metodologia proposta por PETERSEN *et al.* (2008) e KITCHENHAM *et al.* (2011), de forma a mapear e classificar estudos relevantes em relação as categorias de classificação definidas. Como guia para a organização deste estudo tomou-se a seguinte questão: *qual é o estado da arte de gamificação em engenharia de software?* Para que fosse possível responder a essa questão, utilizou-se um conjunto de questões mais específicas que podem ser vistas na TABELA 1 – Questões de Pesquisa.

Tabela 1 - Questões de pesquisa

#	Questão
1	Quais áreas da engenharia de software são gamificadas?
2	Quais métodos de pesquisa têm sido aplicados ao contexto?
3	Quais veículos tem abordado essa temática?
4	Quais técnicas de gamificação são mais utilizadas?

2.2 Fontes de dados e estratégia de busca

As fontes de dados consultadas são as indexadas pelo portal de periódicos da CAPES, sendo que a efetivação da pesquisa se deu através da ferramenta e busca desse portal, que considera publicações tanto da academia como em veículos profissionais estabelecidos, bem como as contribuições científicas e industriais apresentadas em conferências, fóruns, *workshops* e outros.

Para a composição da *string* de busca, utilizou-se os termos definidos pela ISO/IEC 12207 para buscar áreas de processo. A *string* final de busca é mostrada na TABELA 2 – *String* de busca, em que se utiliza a notação booleana de AND para indicar a existência simultânea dos termos e OR para indicar a alternância dos termos.

Tabela 2 - *String* de busca

Termos globais	Termos Específicos
Gamification/Gamificação	(*gamification* ***OR*** *gamifying* ***OR*** *gamify*)/(gamificação ***OR*** gamificar) ***AND***
Software Engineering/ Engenharia de software	(*'software engineering'* ***OR*** *'software process'* ***OR*** *'software management'* ***OR*** *'software testing'* ***OR*** *'software implementation'*)/('engenharia de software' ***OR*** 'gerência de software' ***OR*** 'teste de software' ***OR*** 'implementação de software')

Optou-se por incluir apenas os estudos produzidos até maio de 2017, tendo em vista o tempo de execução da pesquisa. A *string* de busca foi utilizada no texto completo. Ainda, optou-se por realizar-se a pesquisa em português e inglês, de forma que fosse possível abranger o maior número de estudos.

Antes da classificação, foram aplicadas técnicas de inclusão, para diminuir o número de estudos que comporiam o *corpus* de trabalhos a serem analisados. O primeiro critério de inclusão foi a disponibilidade do texto completo do trabalho; estar publicado em português e inglês. Foram excluídos os trabalhos que não enfoquem engenharia de software e gamificação, bem como relatórios técnicos e resumos de conferências.

2.3 Classificação

Para classificar os estudos, foram estabelecidas categorias relacionadas a cada uma das perguntas de pesquisa. Dessa forma, foram estabelecidas as seguintes categorias: processo de software, método de pesquisa, veículo de publicação e técnica de

gamificação. Por processo de software entende-se as áreas de um processo de software estabelecido, para tanto consideramos o processo descrito pela ISO/IEC 12207. No entanto, durante a análise foi necessária a identificação de outras disciplinas, práticas, atividades e processos que não estavam descritos por tal norma, tendo em vista sua aplicação nos estudos, pode-se citar como exemplo a *colaboração*.

Por método de pesquisa buscamos identificar quais métodos foram aplicados, observando se apresentavam validação dos resultados, se propunham soluções, ou se apresentavam experiências ou opiniões pessoais. Ao categorizar-se veículo de publicação, objetiva-se identificar quais eventos, periódicos, conferências e *workshops* têm apresentado interesse pelo tema. Por técnica de gamificação se entende quais elementos de jogos formaram o objeto do estudo, podendo ser pontos, emblemas, líderes, quadros, rankings, etc.

3. Avaliação

Ao realizar-se a busca, junto ao portal de periódicos da CAPES, foram identificados 490 artigos que continham a *string* de busca. No Gráfico 1 podemos ver que o número de estudos tem crescido no decorrer dos anos.

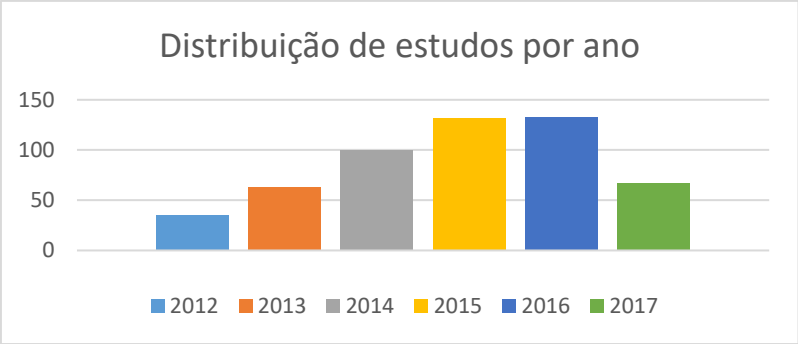


Gráfico 1 - Distribuição de estudos por ano

Desses, após leitura de seu título, resumo e palavras-chave, selecionou-se, de acordo com os critérios discutidos na seção 3.2, 45 artigos para a leitura. Foram numerados os artigos de A1 a A45. Ao considerarmos a primeira categoria, processo de software, foram analisadas as áreas de processos de software que cada um dos estudos tratou, com isso, obtivemos a distribuição apresentada no Gráfico 2.

No Gráfico 2, vimos uma forte tendência à implementação do sistema como temática recorrente, isto é, é uma forte preocupação na área, conjuntamente com as áreas que lidam com requisitos e testes de software. Desta forma, várias propostas apresentam empirismos e estudos de caso sobre estes temas. Há propostas de gamificação de um processo completo, como temos em A45. Se considerarmos requisitos, desenvolvimento, teste, gerenciamento e processos de suporte, como áreas de processos de software, temos uma distribuição de 13% de estudos focando em requisitos, 28% em desenvolvimento, 10% em teste, 23% em gerenciamento e 26% nos processos de suporte.

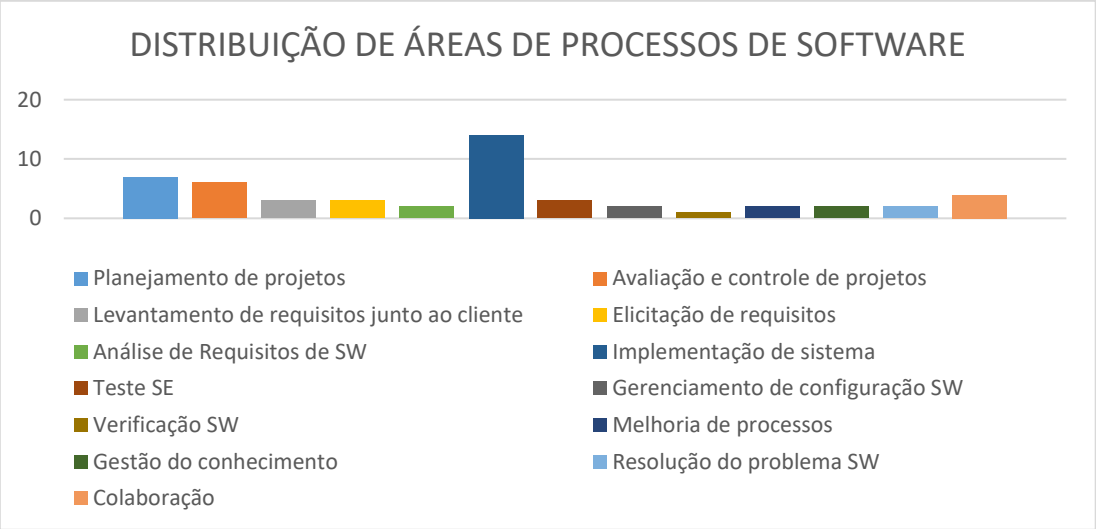


Gráfico 2 - Distribuição de áreas de processos de software

Quando consideramos a segunda categoria, isto é, o método da pesquisa, temos o cenário descrito no Gráfico 3.

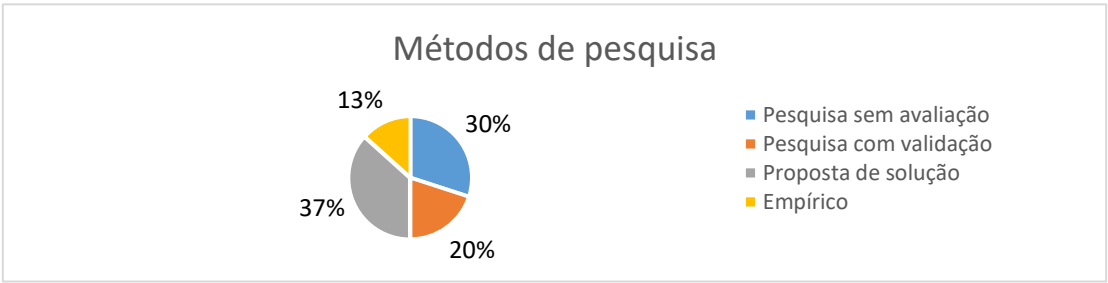


Gráfico 3 - Métodos de pesquisa

No Gráfico 3 é possível perceber que os estudos parecem propor soluções que não são diretamente avaliadas ou validadas com a utilização de métodos formais para tanto, ou não os descrevem em profundidade. Percebe-se que os estudos propõem algum tipo de proposta de aplicação, mas não o fazem em um cenário real de forma a poder validar tal proposta. Para respondermos à pergunta número 3, observamos os veículos de publicação das pesquisas, nota-se que tem crescido progressivamente o número de trabalhos veiculados em *journals*, porém este percentual é inferior ainda a 20%, o que demonstra fortemente que os estudos são bastante preliminares.

E por fim foi avaliado quais técnicas de gamificação são propostas pelo estudo, considerando que não há uma organização de técnicas amplamente aceita, e a maior parte da literatura define como '*mecânicas de jogo'*, listou-se a partir do *corpus* de análise os seguintes elementos: Prêmios: um prêmio especial é dado ao jogador na conclusão de um comportamento; Sistema de recompensa baseado em pontos: os jogadores obtêm uma recompensa na forma de pontos na conclusão de um determinado comportamento; Distintivos: representam certas realizações do usuário; Níveis: relacionados às recompensas baseadas em pontos, os usuários têm um nível que aumenta à medida que atingem um certo número de pontos; Missões: as tarefas que o jogador precisa completar são apresentadas como uma missão, com elementos de jogo adicionais que a torna mais atraente; Votação: os jogadores podem votar no comportamento de outro jogador. Os próprios votos representam as recompensas obtidas por cada jogador; Ranking: um

ranking com os melhores jogadores é apresentado a todos os jogadores para aumentar a competitividade. A posição no ranking pode ser definida por pontos, níveis ou número de votos, por exemplo; Apostas: os usuários apostam em um determinado evento, como uma estimativa, por exemplo. O vencedor da aposta recebe alguma recompensa em troca.

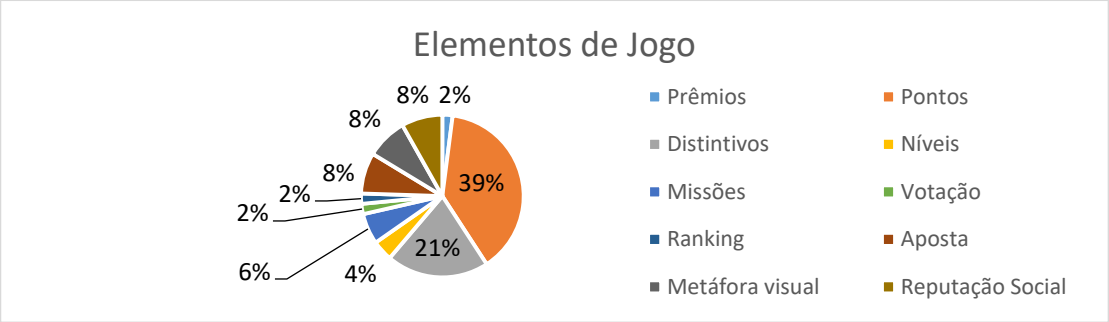


Gráfico 4 - Elementos de jogo

Como pode ser visto no Gráfico 4 - Elementos de jogo, a grande maioria dos estudos aplica o conceito de pontuação, seguido por distintivos. Esses também são vistos com maior frequência na gamificação de outras áreas, como aplicações orientadas a clientes em que pontos são acumulados.

4. Discussão e conclusões

A última etapa da pesquisa foi avaliar os dados obtidos, para tanto se alçou mão de alguns questionamentos, o primeiro deles foi 'a engenharia de software foi utilizada como meio para a gamificação de outra área, ou foi o objeto da gamificação?' Ao ler-se os artigos, tomando por base essa indagação, percebemos que há vários estudos que gamificam outras áreas, por exemplo o ensino da própria engenharia de software. É importante notar que há muitas evidências que o estudo da gamificação da engenharia de software está em uma fase bastante inicial quando comparado com a gamificação na educação, por exemplo. Pensou-se acerca da aplicabilidade da metodologia de pesquisa aplicada no contexto. Tendo em vista os dados apresentados no Gráfico 3, nota-se que aproximadamente metade dos estudos não apresenta uma validação ou uma avaliação, e, dessa forma, sua aplicação em outros contextos é bastante questionável. Aqui há um indício claro de que os estudos de gamificação precisam ainda ser aprofundados de forma a poderem ser generalizados.

Quando consideramos as análises de impactos obtidos nas três frentes analisadas da gamificação, o desempenho, a aderência e o desenvolvimento do domínio, retoma-se a falta de avaliação e validação dos estudos. Embora, nos estudos em que houve algum tipo de avaliação, validação ou estudo de caso, são perceptíveis os impactos positivos da gamificação no desempenho, na aderência e no desenvolvimento do domínio. Também se toma o forte interesse em gamificar áreas de processos em que o cliente está envolvido, como a elicitação e análise de requisitos.

De forma geral, os resultados de PEDREIRA *et al*. (2015) foram reforçados por esse estudo, nota-se um aumento no número de estudos, e um aprofundamento desses, no entanto há uma grande oportunidade de pesquisa para gamificar áreas de processo. Bem como notou-se um crescimento dos estudos que tratam a gamificação como parte central do projeto, bem como um crescimento da preocupação com validação.

O presente estudo objetivou mapear o estado da arte da aplicação de gamificação em engenharia de software, mais especificamente sua aplicação em processos de software. Após a localização dos estudos que compuseram os dados de análise, esses estudos foram classificados segundo as características selecionadas. Além disto, foi realizada uma avaliação subjetiva da metodologia adotada pelos estudos.

Os resultados obtidos durante a análise dos estudos selecionados mostraram que a maioria das pesquisas existentes sobre gamificação aplicada a engenharia de software é bastante preliminar, já que a maioria dos estudos tem sido publicado em workshops ou conferências, e poucos desses oferecem evidências empíricas sólidas do impacto de suas propostas sobre o engajamento e desempenho dos usuários. Além disso, a maioria dos estudos ainda parece não abordar a gamificação como parte fundamental e determinante do projeto; ao contrário, vê-se em geral a incorporação de elementos de gamificação em projetos já prontos. Dessa forma, os resultados corroboram aqueles de PEDREIRA *et al*. (2015)

Esse mapeamento sistemático foi conduzido como parte de um projeto maior que apresenta como principal objetivo o desnvolvimento de uma proposta de abordagem gamificada para execução de processos de software, a partir de uma adaptação de um Role Playing Game (RPG), Dungeons & Dragons, ao contexto de processos de software, utilizando representações dos elementos do jogo para os elementos de processos de software.

6. Referências

Chow, I; Huang, L. (2017) An expert gamification system for virtual and cross-cultural software teams. in Computer Science & Information Technology, Vol.7(1), pp.1-11

Hernández, L; Muñoz, M; Mejía, J; Peña, A; Rangel, N; Torres, C. (2017) Una Revisión Sistemática de la Literatura Enfocada en el uso de Gamificación en Equipos de Trabajo en la Ingeniería de Software in Revista Ibérica de Sistemas e Tecnologias de Informação, Issue 21, pp.33-50

Morschheuser, B; Hamari, J; Koivisto, J; Maedche, A. (2017) Gamified crowdsourcing: Conceptualization, literature review, and future agenda. In International Journal of Human - Computer Studies

Mcconville, J..; Rauch, S; Helgegren, I ; Kain, J. (2017) Using role-playing games to broaden engineering education. In International Journal of Sustainability in Higher Education, Vol.18 (4), pp.594-607

Baptista, G; Oliveira, T. (2017) Why so serious? Gamification impact in the acceptance of mobile banking services. In Internet Research, Vol.27(1), pp.118-139

Gañán, D; Caballé, S; Clarisó, R; Conesa, J; Bañeres, D. (2017) ICT-FLAG: a web-based e-assessment platform featuring learning analytics and gamification. In International Journal of Web Information Systems, Vol.13 (1), pp.25-54

Eysenbach, G; Exeter, C; Yang, S; Willig, J; Almarshedi, A; Wills, G; Ranchhod, A. (2017) Guidelines for the Gamification of Self-Management of Chronic Illnesses: Multimethod Study. In JMIR Serious Games, 2017, Vol.5(2)

Rodrigues, L; Costa, C; Oliveira, A. (2016) Gamification: A framework for designing software in e-banking. In Computers in humam behaviour

Pedreira, O; García, F; Brisaboa, N Mario Piattini. (2015) Gamification in software engineering – A systematic mapping. In Information and software technology

Aseriskis, D; Damasevicius, R. (2014) Gamification pattern for gamification applications. In 6th International conference, Intelligent Human Computer Interaction

Stikkolorum, D; Chaudron, M; Bruin, O. (2014) The Art of Software Design, a Video Game for Learning Software Design Principles. In ACM/IEEE 15th International Conference on Model Driven Engineering Languages and Systems

Sangkyun, K. (2014) Decision Support Model for Introduction of Gamification Solution Using AHP. In The Scientific World Journal

Harbert, T. (2013) Giving gamification a go. In Computerworld

Chung-Ho, S. (2016) The effects of students' motivation, cognitive load and learning anxiety in gamification software engineering education: a structural equation modeling study. In Multimedia Tools and Applications

Blohm, I; Leimeister, J. (2015) Design of IT-Based Enhancing Services for Motivational Support and Behavioral Change. In Business & Information Systems Engineering

Ašeriškis, D; Damaševicˇius, R. (2014) Gamification of a Project Management System. 7th International Conference on Advances in Computer-Human Interactions.

Bacon, D; Parkes, D; Chen, Y, Rao, M; Kash, I; Sridharan, M. (2012) Predicting your own effort, in Proceedings of the 11th International Conference on Autonomous Agentsand Multiagent Systems, pp. 695–702.

Bell, J; Sheth, S; Kaiser. G; (2011) Secret ninja testing with HALO software engineering, in 4th international workshop on Social software engineering(SSE'11), pp. 43–47.

Melo, A; Hinz, M; Scheibel, G; Berkenbrock, C; Gasparini, I; Baldo, F; (2014) Version Control System Gamification: A Proposal to Encourage the Engagement of Developers to Collaborate in Software Projects, presented at the Proceedings of the 6th Int. Conf. on Social Computing and Social Media (SCSM'2014).

Dencheva, S., Prause, C. R., and Prinz, W. Dynamic Selfmoderation in a Corporate Wiki to Improve Participation and Contribution Quality, presented at the Proceedings of the 12th European Conference on Computer Supported Cooperative Work (ECSCW'11), 2013.

Dorling, A., McCaffery, F., The gamification of SPICE, Communications in Computer and Information Science, vol. 290, pp. 295–301, 2012.

Duarte, D., Farinha, C., Silva, M. M. da, and Silva, A. R. da, Collaborative Requirements Elicitation with Visualization Techniques, presented at the Proceedings of the IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'12), 2012.

Dubois, D. J. and Tamburrelli, G. Understanding gamification mechanisms for software development, in Proceedings of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'13), 2013, pp. 659–662.

Fernandes, J., Duarte, D., Ribeiro, C., Farinha, Madeiras Pereira, C., J., and Silva, M. M. da, iThink: A Game-Based Approach Toward Improving Collaboration and

Participation in Requirement Elicitation, Procedia Computer Science, vol. 15, pp. 66–77, 2012 2012.

S. Grant and B. Betts, Encouraging user behavior with achievements: an empirical study, in Proceedings of the 10th Working Conference on Mining Software Repositories (MSR'13), 2013, pp. 65–68.

K. Januszevski. (2012, Visual Studio Achievements Programs Brings Gamification to Development.

Nikkila, S., Byrne, D., Sundaram, H., Kelliher, A., and Linn, S. Taskville: visualizing tasks and raising awareness in the workplace, presented at the Proceedings of Conference on Human Factors in Computing Systems (CHI'2013), 2013.

Nikkila, S., Linn, S., Sundaram, H., and Kelliher, A., Playing in Taskville: Designing a Social Game for the Workplace,presented at the Proceedings of Int. Conf. on Computer Human Interaction (CHI'2011) – Workshop on Gamification: Using Game Design Elements in Non-Game Contexts, 2011.

Passos, E. B., Medeiros, D. B., Neto, P. A. S. and Clua, E. W. G., Turning Real-World Software Development into a Game, in Proceedings of the Brazilian Symposium on Games and Digital Entertainment (SBGAMES'11), 2011, pp. 260–269.

Prause, C. R. and Eisenhauer, M., First Results from an Investigation into the Validity of Developer Reputation Derived from Wiki Articles and Source Code, presented at the Proceedings of the 5th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE'2012), 2012.

Prause, C. R., Nonnen, J., and Vinkovits, M., A Field Experiment on Gamification of Code Quality in Agile Development, presented at the Proceedings of the 24th Annual Workshop of the Psychology of Programming Interest Group (PPIG'2012), 2012.

Prause, C. R., Reiners, R., Dencheva, S., and Zimmermann, A. Incentives for Maintaining High-Quality Source Code, presented at the Proceedings of the Psychology of Programming Interest Group Work-in-Progress Workshop 2010 (PPIG'2010), 2010.

Sheth, S. K., Bell, J. S., and Kaiser, G. E. Increasing Student Engagement in Software Engineering with Gamification, CUCS-018-12, Technical Report, Columbia University, Department of Computer Science, 2012.

Singer L., Figueira Filho, F., Cleary, B., Treude, C., Storey, M.-A., and Schneider, K., Mutual assessment in the social programmer ecosystem: an empirical investigation of developer profile aggregators, in Proceedings of the Conference on Computer supported cooperative work (CSCW'13), 2013, pp. 103–116.

Singer, L. and Schneider, K., Influencing the adoption of software engineering methods using social software, presented at the Proceedings of the International Conference on Software Engineering (ICSE'12), 2012.

Singer, L. and Schneider, K., It was a bit of a race: Gamification of version control, in Proceedings of the 2nd International Workshop on Games and Software Engineering (GAS'12), 2012, pp. 5–8.

Snipes, W., Augustine, V., Nair, A. R., and E. Murphy-Hill, Toward recognizing and rewarding efficient developer work patterns, in Proceedings of the International Conference on Software Engineering (ICSE'13), 2013, pp. 1277–1280.

Snipes W., Nair A. R., and Murphy-Hill, E. Experiences gamifying developer adoption of practices and tools, presented at the Proceedings of the 36th International Conference on Software Engineering (ICSE'2014), 2014.

Sukale, R. and M. Pfaff, S. QuoDocs: improving developer engagement in software documentation through gamification, presented at the Proceedings of the 32nd ACM Conference on Human Factors in Computing Systems(ACHI'2014), 2014.

Thom, J., Millen, D., and DiMicco, J. Removing gamification from an enterprise SNS, in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'12), 2012, pp. 1067–1070.

R. Troughton, Gamification in Software Development and Agile, Software Developer's Journal, 2012.

Vasilescu, B. Human aspects, gamification, and social media in collaborative software engineering, presented at the Proceedings of the 36th International Conference on Software Engineering (ICSE'2014).

Vasilescu, B., Serebrenik, A., Devanbu, P. and Filkov, V.,How social Q&A sites are changing knowledge sharing in open source software communities, presented at the Proceedings of the 17th ACM conference on Computer supported cooperative work & social computing (CSCW'2014), 2014.

Medeiros, D. B., Neto, P. A. S., Passos, E. B., Araújo, W. S. . Working and Playing With Scrum. In International Journal of Software Engineering and Knowledge Engineering. 2015.

Beecham, S. et al. Motivation in software engineering: A systematic literature review. Information and software technology, Elsevier, v. 50, n. 9, p. 860–878, 2008. Citado 2 vezes nas páginas 6 e 15.

Boehm, B. W. Improving software productivity. In: CITESEER. Computer. [S.l.], 1987.

Dubois, D. J.; Tamburrelli, G. Understanding gamification mechanisms for software development. In: acm. Proceedings of the 2013 9th joint meeting on foundations of software engineering. [s.l.], 2013. P. 659–662.

Fowler, M.; Highsmith, J. The agile manifesto, in software development, issue on agile methodologies, http://www.sdmagazine.com, aug. 2001.

Humphrey, W. S. Managing the software process. Massachusetts: addison wesley, 1990.

Kitchenham, B. A., Budgen, D., Brereton, O.P. Using mapping studies as the basis for further research – a participant–observer case study, inf. Softw.technol. 53 (2011) 638–651 (06/01/2011).

Kumar, J. (2013) gamification at work: designing engaging business software. In: marcus a. (eds) design, user experience, and usability. Health, learning, playing, cultural, and cross-cultural user experience. Duxu 2013. Lecture notes in computer science, vol 8013. Springer, berlin, heidelberg

Marczewski, A. Gamification: a simple introduction. [s.l.]: andrzej marczewski, 2012.

Petersen K., Feld, R. Mujtaba, S. Mattsson, M.. Systematic mapping studies in software engineering, in: proceedings of the 12th international conference on evaluation and assessment in software engineering (ease'12), 2008, pp. 68–77.

Learning by Doing em Fábrica de Software: Relato de uma Experiência no Mestrado Profissional em Computação Aplicada

Tainara Lucateli Bernardi, Alexandre Lazaretti Zanatta, Jucélia Giacomelli Beux, Daiana Biduski, Éricles Andrei Bellei

Programa de Pós-Graduação em Computação Aplicada (PPGCA)
Universidade de Passo Fundo – Caixa Postal 611 – 99.001-970 – Passo Fundo – RS

{tainarab, zanatta, 68428, 130011, 168729}@upf.br

Abstract. *Practical teaching approaches, such as Learning by Doing, are fundamentals in the academic training offered by professional master's programs, which seek to apply the obtained knowledge to a field of specific professional action. This paper reports the experience, within a Software Factory, executed in a discipline of the professional masters in Applied Computing, promoting academics integration with real work processes and problems. Formulated reflections and indications show increase of synergy among groups, creativity development, ability to plan among others. The experience was valid as an application in teaching practice and student training.*

Resumo. *Abordagens práticas de ensino, como Learning by Doing, são fundamentais na formação acadêmica proposta por programas de mestrado profissional, os quais buscam a aplicação orientada para um campo de atuação profissional específico. O presente artigo relata a experiência, no âmbito de uma Fábrica de Software, realizada em uma disciplina do mestrado profissional em Computação Aplicada, promovendo a integração dos acadêmicos com processos de trabalho e problemas reais. As reflexões e indícios levantados apontam para o aumento da sinergia entre os grupos, o desenvolvimento da criatividade, capacidade de planejar entre outros. Foi válida e proveitosa a experiência como aplicação na prática docente e na formação discente.*

1. Introdução

A transformação na forma de desenvolver software faz com que várias áreas sejam conectadas e integradas para atender às demandas do mercado. Nesse contexto, a busca por software que atende às necessidades e desejos dos clientes aumentam, transformando as maneiras de desenvolvimento e entrega. A demanda e complexidade no cenário de desenvolvimento de software exigem maior produtividade, associada a um elevado grau de qualidade. Para tanto, é importante observar e enriquecer os processos de aprendizagem utilizados para difundir esse novo contexto de desenvolvimento. Tais questões podem ser aperfeiçoadas em um ambiente educacional com a integração de abordagens práticas de aprendizagem e a aplicação de determinadas técnicas e modelos de funcionamento de Fábricas de Software [Ahmad *et al.* 2014].

Nesse contexto, o conceito da metodologia *Learning by Doing* (LBD) exemplifica uma forma de aprendizagem experiencial, em que os estudantes ganham experiência prática na resolução de problemas do mundo real. Em sua teoria, Dewey (2011) argumenta que para a educação ser mais efetiva, os indivíduos devem receber oportunidades de aprendizagem que lhes permitam vincular o conteúdo presente a experiências e conhecimentos anteriores. A educação também deve preparar os alunos para assumirem uma parte plena e ativa na formação de sua sociedade futura.

A metodologia LBD baseia-se na abordagem construcionista e promove um ambiente de aprendizagem orientado para o aluno. Ela promove o desenvolvimento de habilidades e a aprendizagem de informações por meio da implementação prática de conceitos ou conteúdos [Shank *et al.* 2013]. Nessa metodologia, mentores, que podem ser os professores, auxiliam os alunos no curso de seu trabalho ao fornecer sugestões e não soluções prontas. Ao contrário do modo de ensino convencional de sala de aula, a dependência entre estudante e professor é baixa, o que permite a orientação e o autocrescimento do aluno de maneira prática e consistente [Karri and Kode 2011].

Diante do exposto, este estudo tem como objetivo apresentar a experiência dos discentes do curso de mestrado profissional em computação aplicada da Universidade de Passo Fundo. Aplicaram-se os conceitos de fábrica de software associados à metodologia *Learning by Doing* com o desafio de implementar um software de agendamento online para um departamento dentro da mesma universidade.

O texto está organizado da seguinte maneira: na seção 2 são apresentadas algumas definições conceituais e estudos relacionados. A seção 3 descreve o projeto e detalha a organização do trabalho. Na seção 4 são expostas discussões sobre o esforço e as escolhas feitas. Na seção 5 são apresentadas algumas dificuldades e lições aprendidas com a execução do projeto. A seção 6 encerra com algumas considerações sobre a experiência obtida, limitações e trabalhos futuros.

2. Fundamentação Teórica e Trabalhos Relacionados

Uma fábrica de software é um ambiente definido a partir de processos para realizar as atividades relacionadas ao ciclo de vida de software, abrangendo determinadas práticas de manufatura industrial, com processos, uso de normas e modelos de qualidade, cumprimento de cronogramas e entregas. Os conceitos e as propostas de fábrica de software estão relacionados às linhas de produtos de software, às linguagens para domínios específicos, ao uso de ferramentas computacionais específicas e ao uso de processos [Borsoi 2008].

[Fernandes and Teixeira 2007] afirmam que uma fábrica de software é um processo estruturado, controlado e melhorado de forma contínua, orientado para o atendimento de demandas de natureza e escopo distintas, visando a geração de produtos de software que atendam às necessidades dos usuários da forma mais produtiva e econômica possível. No ambiente acadêmico, pode haver a inserção dos conceitos de fábrica de software com o objetivo de integração entre teoria e prática, onde o discente tem oportunidade de vivenciar problemas reais. [Ahmad *et al.* 2014] explicam que as fábricas de software podem servir como suporte de testes para ideias de engenharia de software e uma fonte para pesquisa básica de desenvolvimento de software científico. Explicam também que essa prática pode ser um veículo educacional para universidades,

onde os artefatos produzidos na fábrica servem para melhorar a aprendizagem e fornecer materiais didáticos em estreita colaboração com a indústria.

O estudo de [Karri and Kode 2011] apresenta um programa de treinamento desenvolvido pelos autores a fim de melhorar a qualidade da educação em Tecnologia da Informação nas faculdades indianas de engenharia no ano de 2008. A metodologia adotada para esse processo foi a LBD, com o objetivo de fornecer aos membros da faculdade conhecimento sobre habilidades e ferramentas necessárias para integrar a tecnologia no ensino e na aprendizagem. A abordagem LBD foi customizada de acordo com o programa elaborado. Em seus resultados, observaram que alunos que passaram pelo treinamento customizado aprenderam de forma mais rápida, demonstraram melhorias em seus níveis de habilidade e expressaram maior satisfação com a experiência de aprendizagem do que aqueles que expostos ao aprendizado sem personalização. Dessa forma, a metodologia LBD mostrou-se satisfatória, não substituindo o método utilizado pela faculdade, mas sim complementando-o de forma refinada.

[Mansker *et al.* 2016] investigaram como a incorporação da experiência prática com aprendizado de serviço – *service learning* – poderia influenciar a aprendizagem e o interesse por serviço comunitário nos alunos integrantes de uma turma de gerenciamento de reuniões e eventos do programa Hospitality and Restaurant Management em uma universidade nos Estados Unidos. Os resultados indicaram que a aprendizagem de serviços juntamente com a atividade prática aumenta a compreensão dos participantes sobre os conceitos abordados no curso, além de facilitar a compreensão de tempo e esforço para planejar um evento, aumentando o desempenho acadêmico dos participantes.

As implicações para os alunos incluem a melhoria da realização acadêmica resultante de um aumento do nível de interesse no curso e maior nível de aprendizagem, levando a uma experiência acadêmica mais positiva. Os estudantes acreditam que a abordagem aumenta sua capacidade de aplicar conceitos aprendidos a novas situações, sendo capaz de direcionar sua escolha profissional para um mercado onde eles serão inseridos já possuindo conhecimentos prévios, baseados na experiência de campo.

A LBD pode ser utilizada em diversas áreas do conhecimento, permitindo que os alunos se envolvam no processo e busquem soluções próprias para tarefa práticas. A experiência por meio dessa aprendizagem acontece de maneira diferente para cada indivíduo que, guiado por suas experiências passadas, conhecimento pessoal e características individuais, pode interagir com a informação de diferentes maneiras e assim, alcançar resultados distintos, tornando o ambiente de sala de aula uma verdadeira representação do mundo real.

O trabalho de [Ahmad *et al.* 2014] relata as impressões e lições aprendidas com a criação de um laboratório de fábrica de software em uma universidade da Finlândia, concebida com o objetivo de facilitar a aprendizagem dos estudantes e atender às necessidades da indústria. No estudo, é examinado como os fatores no ambiente de aprendizado da fábrica de software podem afetar os alunos ao longo de um curso relacionado ao assunto, tendo em vista a avaliação do projeto sob a perspectiva dos estudantes. Dentre as técnicas empregadas nesse processo, destaca-se o uso da aprendizagem colaborativa por meio da LBD, que juntamente com outras técnicas, proporcionou resultados significativos na aprendizagem dos estudantes, em suas realizações acadêmicas e no ganho de habilidades profissionais.

3. O Projeto Desenvolvido

O projeto atual foi desenvolvido na disciplina de Fábrica de Software com carga horária de 30 horas, divididas em 15 encontros semanais, ofertada uma vez ao ano em um Mestrado Profissional em Computação Aplicada. O objetivo principal da disciplina envolveu a construção do conhecimento e o entendimento dos conceitos e atividades realizadas no processo de trabalho de fábricas de software. A adoção da metodologia LBD visou transformar a abordagem da aula em uma experiência de aprendizagem ativa, em que os alunos participam ativamente na busca pelo conhecimento, deixando de serem apenas ouvintes de conteúdo para serem partícipes do processo. Nesse modelo, os alunos, com autonomia, se engajam na realização de projetos que os proporcionam aprendizados com compreensão conceitual, experiência e desenvolvimento de suas habilidades [Revans 2011].

No início dos trabalhos do projeto, em março de 2017, os participantes tomaram diversas decisões, tais como: os papéis dentro do projeto e da estrutura da fábrica de software, o processo a ser seguido, as ferramentas a ser utilizadas, as formas de comunicação, entre outras. Para organizar os trabalhos, o primeiro passo foi definir um plano de projeto com todas as informações necessárias para que o trabalho fosse conduzido da melhor forma possível. Esse planejamento continha uma visão geral do projeto: objetivos, escopo, papéis e responsabilidades, riscos identificados, um plano de comunicação entre as equipes de trabalho e o cliente e, finalmente, o cronograma com as entregas. O prazo final do projeto foi junho de 2017.

O processo avaliativo da disciplina teve três momentos. O primeiro, ao final da disciplina, os discentes realizaram uma descrição (relatório) com suas percepções sobre a disciplina em si e a metodologia utilizada. O segundo, no último encontro, fez-se uma discussão em classe com todos participantes, tendo o professor da disciplina com moderador. O cliente foi chamado para avaliar – conforme as suas experiências – o software entregue. Vale lembrar que uma avaliação institucional foi realizada. Esses dados foram recolhidos e analisados utilizando algumas práticas da Análise de Conteúdo [Bardin 2005].

A fim de organizar e distribuir as atividades, fez-se a análise do perfil de cada integrante do projeto com o intuito de identificar as habilidades e possibilitar a distribuição de tarefas condizentes aos interesses particulares. Os discentes foram organizados em equipes de trabalho, em que cada um assumiu algum papel dentro do contexto de fábrica e, assim, formaram-se três equipes, com as finalidades de análise de requisitos, desenvolvimento, testes e qualidade de software. Para a distribuição de papéis, levou-se em consideração o conhecimento pessoal, as experiências anteriores e as características individuais de cada aluno, seguindo o que preconiza a LBD. Complementando os papéis, um aluno assumiu a responsabilidade de gerente de projeto, com o objetivo de planejar, organizar o cronograma, acompanhar o andamento dos trabalhos e gerenciar o plano de comunicação. Cabe ressaltar que nenhum dos integrantes havia trabalhado anteriormente com outro membro do projeto.

Participaram dez alunos com graduação na área da tecnologia da informação e faixa etária entre 20 e 40 anos, com experiência entre 1 e 5 anos em projetos de desenvolvimento de software. Embora sendo um projeto acadêmico de pós-graduação, buscou-se uma experiência próxima dos desafios de um contexto real de fábrica de software, com a demanda de um software a ser desenvolvido para um cliente real e com

prazos determinados para as entregas de cada tarefa. Após a compreensão geral do escopo do software a ser desenvolvido, os alunos dividiram-se em equipes de trabalho para iniciar as atividades do projeto.

As atividades foram desenvolvidas conforme a estrutura de trabalho utilizada na fábrica de software envolvida no projeto (Figura 1), a qual utiliza o modelo de processo de desenvolvimento iterativo e incremental, acrescido de algumas práticas ágeis. Na figura, podem ser observadas as atividades definidas no processo de desenvolvimento utilizado pela fábrica de software. As equipes do projeto foram definidas e organizadas conforme as áreas de processos abordadas na engenharia de software, as quais são identificadas no fluxo apresentado. Inicialmente, as atividades referem-se ao processo de engenharia de requisitos, na continuidade ao desenvolvimento (projeto e implementação) e, posteriormente, às atividades de verificação e validação.

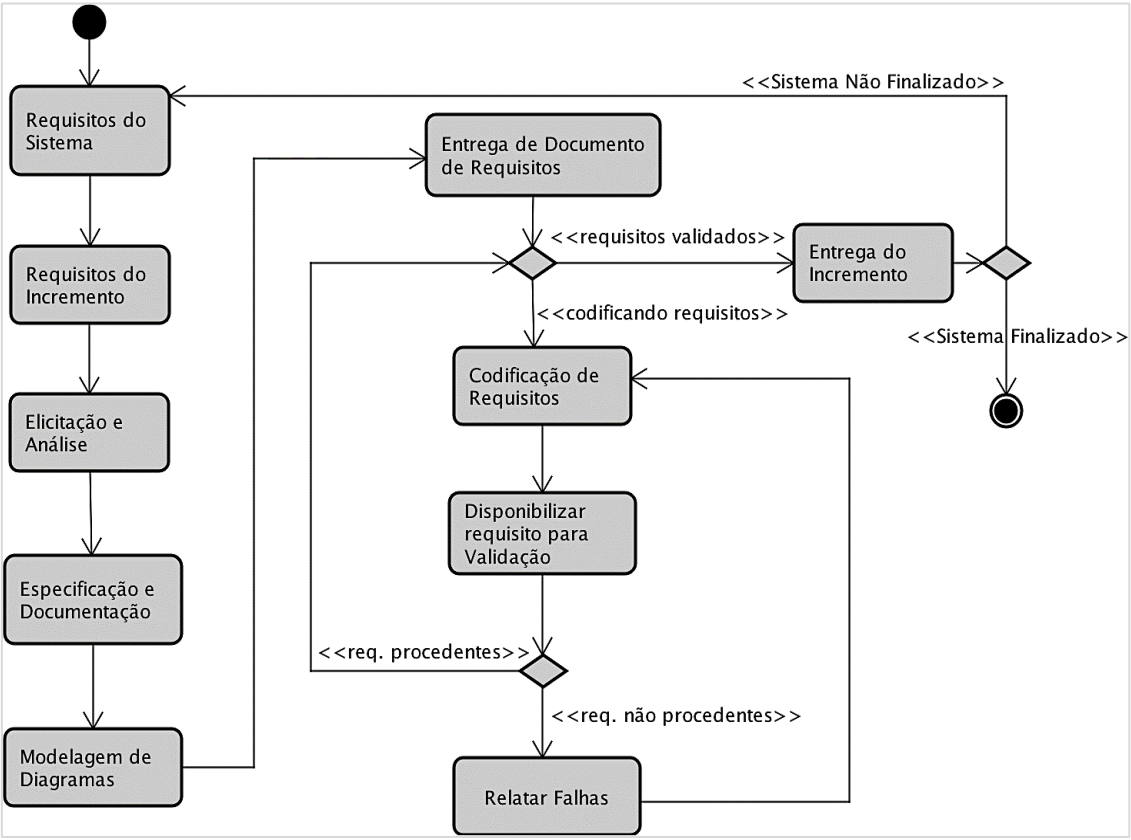


Figura 1. Fluxo de atividades da Fábrica de Software

A equipe responsável pelo processo de análise de requisitos iniciou seu trabalho fazendo a elicitação e análise dos requisitos do software. O processo de especificação e documentação dos requisitos foi construído gradativamente conforme o andamento dos ciclos de trabalho (*sprints*) definidos no projeto. Inicialmente, um formato robusto de documentação foi utilizado, com descrições detalhadas acompanhadas de alguns diagramas da UML (casos de uso, atividades e diagrama de classes). Além disso, *mockups* com a representação das informações de cada funcionalidade foram utilizados para documentar os requisitos funcionais do software. Em um segundo momento, a equipe substituiu o modelo de documentação, passando a utilizar o formato de *User Stories,*

juntamente com diagramas de UML e *mockups*. Desse modo, a equipe aprofundou seus conhecimentos em duas técnicas diferentes de especificação e documentação de requisitos, podendo visualizar as diferenças existentes entre elas. Todos o processo organizacional, como a definição das atividades, a escolha das técnicas utilizadas e a distribuição dos papéis, foram autogerenciadas pelos integrantes da própria equipe, possibilitando que cada integrante realizasse atividades condizentes com suas habilidades e disponibilidade de tempo. Em consequência, todos demonstraram empenho e dedicação para que as atividades fossem realizadas da melhor forma possível.

A equipe de desenvolvimento iniciou o projeto com a estruturação do ambiente de desenvolvimento, bem como a definição da gestão de configuração. Neste projeto, o banco de dados utilizado foi Oracle, o *backend* foi desenvolvido em Java com o framework Spring[1] e o *frontend* com AngularJS[2]. A cada ciclo de desenvolvimento, a equipe recebia os requisitos da equipe de análise, codificava e aplicava os testes de desenvolvimento. Depois disso, encaminhava a funcionalidade para uma nova equipe no projeto. A gestão de configuração foi organizada utilizando a ferramenta GitLab[3], a qual contribui na qualidade do trabalho colaborativo realizado pela equipe. Toda a organização e gestão das atividades realizadas na equipe de desenvolvimento foram gerenciadas pelos próprios integrantes da equipe. Os mesmos consideraram os conhecimentos técnicos e as habilidades de cada membro para a definição dos papéis e responsabilidades na equipe. Esta análise e organização contribuiu significativamente para o bom andamento das atividades do projeto.

O processo de teste e qualidade de software ficou sob responsabilidade de uma terceira equipe, a qual foi analisou, documentou, implementou e executou os testes de cada incremento do software. Para tanto, foram efetuados diferentes tipos de teste que contribuíram com a qualidade do produto de software final. Os problemas identificados eram reportados ao gerente do projeto que, imediatamente, via ferramenta de gerenciamento de atividades, repassava aos desenvolvedores as demandas de ajustes.

O tempo reservado para as aulas semanais da disciplina foi utilizado para reuniões das equipes de trabalho e alinhamento das atividades gerais do projeto. Acompanhados pelo professor, os membros decidiram sobre o andamento de suas tarefas, verificação de dependências, entregas de artefatos e outras necessidades. As equipes tiveram autonomia, como preconiza a LBD, para fazer a divisão de tarefas e seu autogerenciamento, acompanhadas pelo gerente de projeto. O período disponível para o projeto foi de quatro meses e, a partir desse prazo, foram estipuladas datas para entregas parciais e a entrega final do software. As reuniões posteriores com o cliente foram feitas pelo gerente de projeto e pela equipe de análise de requisitos. Durante as reuniões, foram apresentados protótipos da interface gráfica do software para detalhamento das funcionalidades ao cliente, que verificavam a conformidade à sua demanda e opinavam sobre necessidades de ajustes. Os protótipos da interface do software, assim como os *mockups* utilizados na documentação de requisitos, foram elaborados no software Balsamiq[4].

[1] https://spring.io
[2] https://angularjs.org
[3] https://about.gitlab.com
[4] https://balsamiq.com

Assim como acontece em um ambiente de fábrica de software [Fernandes 2009], os requisitos do software desenvolvido na disciplina foram divididos por ordem de importância e urgência, separados em duas modalidades: requisitos do usuário e requisitos do administrador do software. Os requisitos essenciais para o funcionamento do software foram movidos para o início da lista de prioridades, enquanto as sugestões de melhorias e os ajustes necessários foram analisados e organizados para um segundo momento. Para os requisitos com baixa complexidade, reservou-se uma semana para o desenvolvimento e teste, enquanto outros, como requisitos fundamentais e requisitos com alta complexidade, tinham 20 dias para o desenvolvimento e entrega parcial.

No contexto, muitas das atividades foram desenvolvidas em ambiente distribuído, possibilitando aos discentes uma nova experiência, pois nos dias atuais, muitas empresas e fábricas de software adotam metodologias de trabalho que independem da presença física dos colaboradores no ambiente corporativo [Fernandes 2009]. Desse modo, para facilitar a organização, colaboração e comunicação entre os participantes, foram utilizadas algumas ferramentas de trabalho, as quais fazem parte do portfólio de ferramentas da fábrica de software da instituição. O repositório e o controle de versionamento de artefatos da documentação do projeto e dos códigos fonte foram gerenciados com o GitLab. A organização do fluxo de realização das tarefas aconteceu na ferramenta Trello[5]. Além disso, a comunicação no formato de listas de discussão, fórum e chats entre os participantes, aconteceu na ferramenta Slack[6] e nos grupos criados no WhatsApp[7].

4. Discussão

Além da consolidação dos conteúdos da disciplina, os alunos que participaram desse projeto aproveitaram a vivência para conhecer os processos de trabalho de uma fábrica de software e conviver com a resolução de problemas reais. A forma de trabalho utilizada na disciplina materializa o objetivo do mestrado profissional de estreitar as relações entre as universidades e o setor produtivo, visando a capacitação de profissionais nas diversas áreas do conhecimento, mediante o estudo de técnicas, processos ou temáticas que atendem a alguma demanda do mercado de trabalho [MEC 2009]. Essa modalidade de pós-graduação *stricto sensu* apresenta forte relação didática com as abordagens práticas de ensino, como *Learning by Doing*, pois ambos buscam a aplicação orientada para um campo de atuação profissional, em que a aprendizagem experiencial é utilizada para que os estudantes adquiram experiências práticas na resolução de problemas do mundo real.

A partir da experiência, várias reflexões e discussões foram realizadas com os alunos com o objetivo de analisar o aprendizado construído por meio da vivência real do ambiente de trabalho. O trabalho distribuído e a colaboração foram as características mais preocupantes do projeto, considerando o agravante de que nenhum dos participantes, nem mesmo o gerente de projetos, tinham experiência em trabalhar com esse perfil de equipe. Nesse cenário, as ferramentas de trabalho auxiliaram o processo de comunicação e de gerenciamento das atividades a desenvolvidas em cada equipe dentro do contexto de fábrica de software. A comunicação com o cliente, predominantemente feita pelo gerente de projetos e pela equipe de análise de requisitos, colocou construtivamente os alunos no

[5] https://trello.com/home
[6] https://slack.com
[7] https://web.whatsapp.com

ambiente real de trabalho, onde puderam perceber que a comunicação e o processo de negociação são importantes e decisivos para o sucesso de um projeto.

O *Learning by Doing* traz uma nova perspectiva de aprendizagem pelo fato de promover o autodesenvolvimento, o qual foi enfaticamente relatado pelos discentes participantes da disciplina. Alguns dos alunos não tinham experiência profissional e nunca haviam trabalhado em um ambiente real de produção de software. Os mesmos relataram que os desafios inicialmente pareciam assustadores, mas que a vivência das atividades ampliou significativamente o conhecimento e o desenvolvimento particular. Um fator a ser considerado da experiência vivenciada refere-se aos diferentes perfis de profissionais integrantes do projeto. Ressalta-se que ninguém conhecia a forma de trabalho utilizada por cada um de seus colegas, o que poderia ter dificultado o andamento do trabalho. Porém, os relatos demonstram que isso agregou muito crescimento pessoal e profissional no que se refere ao trabalho colaborativo e distribuído.

5. Desafios e Lições Aprendidas

Durante o andamento do projeto, a equipe enfrentou alguns desafios relacionados a cronograma, planejamento de atividades, comprometimento, domínio de tecnologias, heterogeneidade da equipe e dinâmica de trabalho. Estes desafios levaram a algumas lições aprendidas consideradas importantes para a execução de novos projetos.

Em relação ao cronograma e o planejamento, o cumprimento dos prazos foi um dos maiores desafios enfrentados no projeto. Inicialmente, fez-se o planejamento considerando quatro horas semanais de trabalho para cada integrante do projeto. Porém, nem todos os discentes conseguiram disponibilizar o tempo planejado para a realização das atividades. Por outro lado, alguns relataram maior dedicação do que o tempo planejado, dedicando aproximadamente seis horas semanais. Foi necessário ajustar o cronograma e adiar as entregas ao cliente, pois a tarefa de um integrante acabava comprometendo o cronograma de realização das atividades subsequentes. Houve incremento na demanda de horas de trabalho, ocasionado pela dificuldade de integração do software desenvolvido com a estrutura da Fábrica, sendo esse o principal problema de cunho técnico do projeto.

Oficialmente, a disciplina finalizou em junho de 2017. Todavia, como o projeto não foi concluído, os alunos se comprometeram em terminar o projeto após o encerramento da disciplina, mesmo estando em período de férias. Isso implicou trabalho extra de motivação para a sua conclusão. Após 15 dias do encerramento da disciplina, foi possível disponibilizar uma versão funcional do software para a análise do usuário. A versão final do software foi entregue ao cliente 30 dias após a data planejada e estipulada durante a estruturação da disciplina. Deste modo, foi possível confirmar a existência de readequações no planejamento das atividades da disciplina.

Particularmente nesse projeto, pelo fato de trabalhar com discentes de uma disciplina de mestrado, o comprometimento estava fortemente relacionado com o interesse particular de cada aluno. Foi difícil granjear o comprometimento de todos os integrantes numa mesma proporção, considerando que cada indivíduo tem uma percepção individual e objetivos diferentes sobre o comprometimento na realização das tarefas. Logo no início do projeto, um dos integrantes da equipe de testes e qualidade de software solicitou desligamento do programa de mestrado. Esse fato impactou diretamente no andamento do projeto, pois houveram dificuldades no repasse das atividades que até então

estavam sob sua responsabilidade. Dessa forma, foi necessário readequar os integrantes da equipe e aperfeiçoar a comunicação para que não se tornasse uma vulnerabilidade no projeto. Ao se aproximar da conclusão das atividades, novas dificuldades foram enfrentadas nessa mesma equipe de trabalho, todas elas decorrentes do comprometimento e conhecimento técnico dos integrantes. As demais equipes não apresentaram problemas. Nessa perspectiva, acredita-se que o interesse individual de cada integrante foi fator preponderante para o sucesso na realização das atividades.

A falta de domínio das tecnologias escolhidas e a pouca experiência em desenvolvimento de software, por parte de alguns componentes da equipe, foram fatores desafiadores. Parte do trabalho aconteceu remotamente e os integrantes não estavam habituados a trabalhar juntos e em um projeto colaborativo com a equipe distribuída. Nesse fator, a comunicação da equipe foi um dos elementos mais proeminentes para compartilhamento de soluções. Soma-se o fato de que a ocasião foi a primeira experiência com um projeto de trabalho diferenciado em uma disciplina do mestrado e, por isso, suscitava dúvidas e inseguranças quanto ao resultado final.

Dentre as várias lições aprendidas, destaca-se a importância de estabelecer canais de comunicação eficientes entre todos os integrantes e utilizar ferramentas para gerenciar as atividades de cada equipe do projeto, como destacam [Fuks and Pimentel 2011]. Essa organização facilitou o trabalho dos participantes e levou ao sucesso das atividades. Outro destaque deve-se ao trabalho com grupo heterogêneo de pessoas, as quais não se conheciam e apresentavam perfis profissional diferentes. Os participantes, além de desenvolver um software com prazo estabelecido para a sua conclusão, precisaram resolver conflitos de opiniões e dinâmica de trabalho diferentes do comumente utilizado.

6. Considerações Finais

A partir da experiência relatada, é conciso afirmar que as pessoas precisam receber oportunidades de aprendizagem que possibilitem o vínculo dos conteúdos da atualidade com as experiências e conhecimento já vivenciados. Dessa forma, os envolvidos estarão se preparando para assumir e atuar ativamente em projetos futuros. Considerando os indícios levantados que apontaram para o aumento da sinergia entre os grupos, o desenvolvimento da criatividade, capacidade de planejamento, entre outras melhorias, conclui-se que foi válida a experiência na formação discente. Além disso, foi considerada válida a experiência como aplicação na prática docente e na formação de discente.

Cabe ressaltar que a experiência confirmou as expectativas de sucesso e de ganhos didáticos ao integrar os conhecimentos teóricos sobre fábrica de software e a vivência prática em um ambiente de trabalho profissional promovido pela abordagem LBD. Isso mostra que uma das formas mais assertivas de aprendizado é a prática. Acredita-se que essa abordagem de ensino poderá ser aplicada em novas experiências e contribuir significativamente como técnica de ensino em ambientes de mestrados profissionais.

Por outro lado, mais pesquisas envolvendo a abordagem LBD precisam ser executadas, investigando todo o conjunto de características, possibilidades e aplicações. Dessa forma, pode-se confirmar que os alunos realmente adquirem e fazem uso do conhecimento e das habilidades transmitidas por meio do ensino, corroborando que a metodologia LBD é uma excelente maneira de alcançar os resultados almejados [Karri and Kode 2011].

Durante a realização do projeto, foram percebidas algumas limitações. A experiência relatada refere-se a uma primeira aplicação da abordagem LBD no trabalho com fábrica de software no mestrado profissional da instituição. Consequentemente, os resultados atingidos limitam-se a um projeto e a um modelo de processo de trabalho específico. Além do mais, até o momento ainda não foi possível validar o produto de software com o cliente, pois encontra-se em processo de implantação. Da mesma forma, a validação do conhecimento construído na disciplina foi baseada apenas nos indícios observados e no relato dos participantes do projeto. Como trabalhos futuros, espera-se realizar novo projeto com o mesmo modelo utilizado e validar o conhecimento adquirido pelos participantes.

Agradecimentos

Agradecemos a todos alunos que participaram dessa atividade, ao gestor da Fábrica de software e à Divisão de Tecnologia da Informação da Universidade de Passo Fundo pela oportunidade na realização do presente trabalho.

Referências

Ahmad, M. O., Liukkunen, K. and Markkula, J. (abr 2014). Student perceptions and attitudes towards the software factory as a learning environment. In *2014 Global Engineering Education Conference (EDUCON)*. IEEE.

Bardin. L. (2005), Análise de conteúdo, Lisboa: Edições 70.

Borsoi, B. T. (1 jul 2008). Arquitetura de processo aplicada na integração de fábricas de software. Biblioteca Digital de Teses e Dissertações da Universidade de São Paulo.

Dewey, J. (2012), Democracy and education, Milton Keynes: Simon & Brown.

Fernandes, A. A., Teixeira, D. S. (2009), Fábrica de Software, São Paulo: Atlas.

Fuks, H., Pimentel, M. (2011), Sistemas colaborativos, Elsevier Brasil.

Karri, S. K. R. e Kode, S. (2011). Effectiveness of *Learning by Doing* Methodology in Training Programs. In *2011 11th International Conference on Advanced Learning Technologies*. IEEE, p. 227-231.

Mansker, V., Fulks, R., Peters, B., Curtner, R. e Ogbeide, G.-C. (2016). *Learning by Doing*: A Case Study of Hospitality Students' Learning Experience via Service Learning/Hands-on Experience. *Tourism Travel and Research Association: Advancing Tourism Research Globally*.

MEC, Ministério da Educação (2009). "Portaria Normativa nº 17, de 28 de dezembro de 2009". http://bit.ly/2sI53we.

Revans, R. W. (2011), ABC of action learning, New York: Gower.

Shank, R. C., Berman, T. R. e Macpherson, K. A. (2013). *Learning by Doing*. In: Reigeluth, C. M. *Instructional-design Theories and Models: A New Paradigm of Instructional Theory*. II ed. Evanston: Taylor & Francis. p. 161–181.

On the Characterization of the SDET Role in Agile Projects: Expected Skills, Responsibilities, and Practices

Carolina Toscani, Júlia Couto, Josiane Kroll,
Alessandra Dutra, Sabrina Marczak, Rafael Prikladnicki

[1] MunDDos Research Group - PUCRS
Porto Alegre, RS - Brazil

{carolina.toscani, julia.couto, josiane.kroll}@acad.pucrs.br

{alessandra.dutra,sabrina.marczak,rafaelp}@pucrs.br

Abstract. *Software Development Engineer in Test (SDET) is a formal title for a software tester in software development context. In this paper, we report on a systematic literature review on the SDET role in agile projects. From the 27 selected papers, we extracted SDET role' skills, practices and responsibilities. For instance, we found that motivation, flexibility, and work experience in software engineering are important skills for the SDET be able to perform her work. We also found that developing the system, performing automated unit test, and performing exploratory tests are among the main responsibilities of a SDET. Test automation, A/B test, and Behavior-Driven Design are some of the most cited practices of this role in agile projects.*

1. Introduction

The SDET - Software Development Engineer in Test, is a term originated in 2005 at Microsoft. Developers at Microsoft have the common title of Software Development Engineer, or SDE [Page and Johnston 2008]. Analogously, the develop features by writing code. Analogously, the formal title for a software tester at Microsoft is Software Development Engineer in Test, or SDET. The similarity in the names of the two disciplines is by design, because testers at Microsoft are developers.

As initially proposed by Microsoft and later adopted by other companies, a SDET is part of the development team, and it participates in the full development cycle process. This role must be able to create high quality, maintainable, and performance code. Among other things, testers design tests, influence product design, conduct root cause analysis, participate in code reviews, and write automation tests. Occasionally they check bug fixes or work on small features. Sometimes testers check bug fixes or work on small features. Testing is a heavy workload, so writing features is not very common for testers. In this context, there are many open questions about the SDET concept and its implementation in agile projects: What are the skills necessary to perform the SDET role? What are the responsibilities of the SDET role? What are the practices associated with the SDET role? To answer these questions we investigated the SDET concept in the literature. We searched for theoretical and empirical work, given the novelty of the topic.

Our results yielded 14 skills for a SDET person to perform the role. These skills are organized into two categories, namely: personal and technical. We also found 16 key responsibilities and 34 practices associated with the SDET role. The reminder of this paper presents our literature review and details our results.

Figure 1. Phases of our literature review

2. Research Method

Based on the perceived need for conducting this study, our research questions (RQ) are as follows: What are the responsibilities of the SDET role? What are the skills necessary to perform the SDET role? What are the practices associated with SDET role? To answer the posed research questions, we searched studies using the Scopus digital library. We adopted the Start tool [Start 2017] for data extraction and to categorize the necessary information about study characteristics and findings from the included studies. The search was conducted using the boolean search expression as follows:

(ALL(”software development”) OR ALL(”software project”) OR ALL(”software engineering”) OR ALL(”software”) OR ALL(”system development”) OR ALL(”information system”)) AND TITLE-ABS-KEY(agile) AND ALL(automat*) AND (ALL(testing) OR ALL(test))) AND (LIMIT-TO(SUBJAREA,”COMP”)) AND (LIMIT-TO(LANGUAGE,”English”))

Figure 1 presents the phases of our literature review. After this extensive data search, 705 papers were identified, published between the years 1995 and 2016 - year in which the research was made. Upon a first round of reviews considering the paper title, abstract, and keywords, a total of 130 papers were selected for full reading. Out of these 130 papers, 27 were selected after the application of inclusion and exclusion criteria (second round of review). The inclusion and exclusion criteria for the study are as follows:

- (I) Papers of qualitative or quantitative nature of the subject on SDET.
- (I) Papers available online for download at no charge.
- (I) Papers with the full text of the study in electronic format.
- (I) Conference, workshop, and journals studies.
- (I) Empirical, theoretical papers.
- (E) Extended abstracts and editorial papers.
- (E) Papers unavailable online for download at no charge.
- (E) Papers not related to SDET.
- (E) Duplicated papers.
- (E) Non-English written papers.

3. Results

Our literature review identified 14 skills, 16 responsibilities, and 34 practices related to the SDET role in agile projects. To better relate these results to agile development, this section first describes the context in which the reported projects took place and next introduce each of the results per category: skills, responsibilities, and practices.

Context is the set of circumstances or facts that surround the software development where the SDET role acts. Most studies did not specify the companies' name in which this role was observed (74.1%), but Microsoft, Ericsson, and John Deere are cited. Most of the companies were in the domain of software development (33,3%), 7% are university studies, and 33,3% are undefined. Companies are located in Germany, Norway, India, Italy, China, South Korea, Brazil, and Finland.

The analysis show that in 75% of the papers developers are responsible for the tests. About 94% of selected papers (25 out of 27) presented testers as part of the team, while a minority presented testers working outside the team. About the adopted agile methods, 40,7% of papers cited Scrum, 25,9% cited XP, and 3,8% cited both Scrum and XP. It also shows that 44,4% of the papers described projects developed in a collocated environment, while only 14,8% are executed using distributed software development. More than 40% do not have information about how the teams interact physically, if they are or not in the same location.

Most of the projects (70%) do not inform how many teams are working in the projects, and less than 30% have between 1 and 3 teams working together. 75% papers do not give information about the number of people in the same team, neither. In the other ones, the number varies from 3 to 19 people working together in the same team.

3.1. SDET Skills

A total of 14 skills were identified. SDET skills can be organized into two groups of characteristics: personal and technical. Personal skills are the individual characteristics of a person, mostly related to attitude and personality. Technical skills are abilities acquired during training or experience, such as a specific programming language, test techniques, a technique used to get systems requirement, etc. It is important to mention that a SDET may have all personal and technical skill or at least some of them.

According to the literature, people who are SDETs must be *highly motivated*. The responsibility sense and the positive feedback can contribute to high levels of motivation. Motivated people think about their tasks as challenges, and do not care if they have to redo the tasks some time, to make it better. They think about new ideas, so the time they spend at work is not waste just in routine stuff. On the other hand, when someone is not motivated to do some task, the tasks can be annoying, and upsetting [Prechelt et al. 2016].

Other personal skill related to SDET role is *responsibility*. It is one of the consequences of empowerment. Every team is responsible for what they release. It is most related to the responsibility that a developer have to have on the code he develops, so one cannot write a code knowing that it will fail and rely that other person (the tester) will find and fix it [Prechelt et al. 2016]. Additionally, SDETs also should be *determined, flexible,* and *talented*.

As Agile teams usually work with the premise that everyone on the team have a broad skill set, the following technical skill are emphasized:

- Knowledge of code development: know how to use the programming language.
- Knowledge of database administration: knowledge on managing, installing, update and monitoring a database.
- An understanding of system analysis: process mapping, data modeling, and requirements gathering in a system development project.

- Knowledge of database architecture: data modelling in a database.
- Ability to work with interface design: design of the system interaction and user experience.
- Work experience in software engineering: processes related to software specification, development, and maintenance.
- Knowledge of project management: application of knowledge, skills and techniques to the execution of projects in an effective and effective way.
- Knowledge of systems engineering: development and organization of complex artificial systems.

3.2. SDET Responsibilities

Responsibilities are tasks or set of tasks attributed to the SDET role, that are executed individually or in collaboration with other roles. Some responsibilities such as performing automated unit test, developing the system [Agarwal and Deep 2014], performing integration tests [Sultanía 2015], and writing test cases [Little et al. 2012] are only cited in the papers. Others are described in details as presented next.

- Refactoring code: this responsibility is based on rewriting the code to match with acceptable standards [Kim and Ryoo 2012].
- Performing automated test: it is made in order to perform various testing activities, such as: test case management, test monitor and control, test data generation, test case generation, and test case execution [Rahman et al. 2015]. It can be used as a real-time indicator of the development progress and as a live documentation on the software [Park and Maurer 2008].
- Performing acceptance testing: it is an agile testing approach, also known as automated acceptance testing (AAT). In this approach, the customer or her representative is given the role of expressing requirements as input to the software team paired, with an acceptance test as a result. Acceptance testing allows to test business logic. It is a way of communicating precise requirements from the customer to the developers' point of view [Haugset and Hanssen 2008]. Acceptance tests are written in early stages, in order to validate user needs and it helps to direct the codification [Abrahamsson et al. 2009].
- Performing exploratory testing: it is a type of manual test, in which the responsible for the test utilize their domain expertise to explore and challenge the system in a way more likely to be used by one of the real users [Little et al. 2012].
- Performing fixtures test: it is strictly related to the software platform, and it can be used as formal specifications. The developer writes test to legacy systems, then migrates it to the new system, modifying or rewriting the code according to the target platform. The fixture tests should be written to test both: the legacy and the target systems [Abbattista et al. 2009].
- Writing positive and negative tests: negative tests are tests written to provoke an error, and it is used mainly to ensure that the system handles errors as intended. Positive tests are the common type of tests [Haugset and Hanssen 2008].
- Participating in requirements definition: the developers substantially participated in the requirements definition, apparently based on (1) their increased understanding of the domain as produced by more realistic feedback from the field (as discussed below), (2) their generally high motivation (also discussed below), and (3)

their feeling of responsibility. Their participation reinforced the motivation and the feeling of responsibility [Prechelt et al. 2016].

- Performing tables test: once story test tables have been defined, developers write test fixtures – the code which is used to take the inputs from the story test tables – exercise the system under test (SUT) and compare the observed and expected results [Abbattista et al. 2009].
- Writing test specification: developers will start from scenarios produced in one iterative decomposition process. Scenarios will be translated to tests, which will drive the implementation. A scenario is composed of several steps. A step is an abstraction that represents one of the elements in a scenario, which are: contexts, events, and actions. [Solis and Wang 2011].
- Performing unit test: once user stories are ready, customers or users together with testers or developers define story tests aimed at validating requirements described by user stories. Having the tests in place gives a clearer perspective on what one wants to achieve as well as a confidence in performing the migration. Differently to test fixtures, which are strictly related to the software platform, story tests are written just once and they should be completely or, at least, partially reused on the target system after they are written on the legacy system [Abbattista et al. 2009].
- Writing user stories: it aims building a descriptive form of the requirements of the system to be migrated. User stories might also consider issues related to the specific platform for which the migration is executed [Abbattista et al. 2009].
- Performing white-box test: in Agile methodologies, developers are in charge of executing white-box test using unit testing as a part of the validation process. As developers are working with the source code, they know: the software structure to define the appropriate test cases and, when a test case fail which lines of code are are involved in the revealed a problem [Abrahamsson et al. 2009].

3.3. SDET Practices

In the software engineering area, a practice is a software development practice that, through experience and research, has proven to reliably lead to a desired result and is considered prudent and advisable to do in a variety of contexts. We found 34 practices related to SDET role, presented in detailed next. However, three practices were only cited in the papers, with no associated description, as follows: exploratory testing [Anand and Mani 2015], refactoring [Tripp et al. 2016], and user stories [Almog and Tsubery 2015].

- Automated testing: it is the practice of automated techniques in order to perform various testing activities, such as test case management, test monitor and control, test data generation, test case generation, and test case execution [Rahman et al. 2015]. Test automation is a software framework that helps to test software product. It allows the team to accumulate test cases over the life of the application, so that both existing and new features can always be tested [Sultanía 2015].Verify definition in the papers [Sultanía 2015, Rahman et al. 2015].
- Collaboration with stakeholders: it is about practices of collaborations between groups of stakeholders, such as product managers, senior managers, user manual authors, application specialists. One example is participation in trade shows, where the test engineers have the opportunity to interact with customers and vendors [Anand and Mani 2015].

- A/B testing: also known as Alpha Beta test, it is a type of test in which a specific version of the software is deployed to a limited number of users to get feedback [Rahman et al. 2015].
- Automated Acceptance Testing (AAT): is used to test if the software developed comply with the contractual requirements of the end user [Rahman et al. 2015]. It relies on automation, and provide specification details[Kim and Ryoo 2012]. It is a complementary test approach in agile development.
- Agile specification driven development: it is an extension of TDD (Test-Driven Development), combined with Design-by-Contract (DbC) that allows a language to explicitly declare pre and post-conditions for invariant and methods for a class [Hammond and Umphress 2012].
- Reproduce concepts and theories: it is concerned to applying learnings from conferences, trade shows, courses, etc. in the tester engineer daily routine [Anand and Mani 2015].
- Automated builds: to rebuild the software, the team use a code script. Using this practice, they can assure that all the files that are necessary to successfully build the software are being added to the code repository [Tripp et al. 2016].
- Behavior-Driven Design (BDD): aims to change the mindset of the developers to focus on the behavior of the code [Hammond and Umphress 2012]. Its objective is to get executable specifications of a system [Solis and Wang 2011].
- Bring-up tests: test cases made to verify the most basic functionality of the application. The build is tested just after the bring-up test run successfully [Anand and Mani 2015].
- Competency development: as test engineers has to become generalizing specialists, competency development plans has to be defined. For example, a training needs list is defined, as well as a list of people who have the competency to provide training, and it results in training schedules. It can also be auto developed, by listening/watching audio and video records [Anand and Mani 2015].
- Continuous integration: using this process, the code is systematically and regularly built and deployed to a test server [Tripp et al. 2016, Williams et al. 2011]. In continuous integration practice, the work created by the members of a team is frequently integrated, normally each person integrates at least daily, and it leads to multiple integration per day [Kim and Ryoo 2012].
- Customer-centric configuration information: are identical replicas of the client systems configuration and it is formed by identical hardware in the customer site, operating system, third-party software, patches, and appropriate database [Anand and Mani 2015].
- Adoption of EATDD or ATDD or STDD: in Executable Acceptance Test Driven Development (EATDD), software requirements are written in form of executable acceptance tests, which are developed by the customer in collaboration with business analysts and/or quality assurance engineers [Park and Maurer 2008]. Acceptance Test-Driven Development (ATDD) is one approach in which the customer writes acceptance tests that the developers can use to see if their software is providing the correct functionality [Hammond and Umphress 2012] [Smith et al. 2009].
- Exposure to testers: to expose the test engineer to the end user work-flow, training can be provided in application workflows, customer site visits may be arranged, and weekly meeting with customer care personnel can be scheduled. Others forms

of exposures are related to awareness of competitor's solution and establish or improve communication with Product Management [Anand and Mani 2015].

- Focus on high impact errors: every test engineer attempt to create the highest impact, by testing only as much as necessary, in order to make testing effective. The code changes that have the highest impact on the entire functionality are prioritized [Anand and Mani 2015].
- Functional test: in this type of test, functional requirements of the software are tested with a specific set of input, ignoring the internals of the software [Rahman et al. 2015].
- Global test planning: it is derived from the test concept, that specifies the test approach, the major test assets required and their availability and the strategy for adhere to regulatory requirements and acceptance with customers. The definition of the test concept requires a deep understanding of the domain, technology expertise, and awareness of the regulatory requirements [Anand and Mani 2015].
- Increasing quality focus: to reduce unproductive effort spent on testing, adherence to quality, regulatory and organization quality standards is a very important step way to achieve this goal is providing testers with the same access to team activities, documentation and portals, just like the others team members [Anand and Mani 2015].
- Integration test: refers to gradual testing among different components of the software to test the functionality of the complete software [Rahman et al. 2015].
- Involving the customers: keeping they well informed. For example, when done in moderation, it was possible for one team in Europe to interact and align with the customers due to their proximity, on a regular basis seeking feedback through product demo and informal verification of fixes at customer sites. Controlled customer involvement is important an agile contribution, but attention must be paid to all stakeholders [Anand and Mani 2015].
- Training plan in applications: it typically takes a long time to learn the complex domain of the applications. The complexity is in the terminology, conceptual understanding and a wide range of interfacing systems. A training plan involving all the basics was imparted through instructor-led training's, supplemented with online training material [Anand and Mani 2015].
- Pair programming: this practice recommends putting two developers working in pairs to develop a portion of code [Tripp et al. 2016].
- Planning meetings: involves to plan a meeting with the entire global team for test team coordination and stand-up meetings covering different locations [Anand and Mani 2015].
- Regression suite/test: includes functional and non-functional tests. Some regression test are automated. However, several regression tests are executed manually, such as simulating customer-centric configurations and testing for interoperability, using the approach of "warm body testing'. To ensure the efficiency, it was scheduled only when the application was sufficiently stable [Anand and Mani 2015]. Regression testing is performed to check if a software change has adversely affected the functionality of the whole software [Smith et al. 2009].
- Repository Driven Test Automation (RDTA): proposing reuse of testing artifacts as a fundamental principle for the creation test automation is what makes RDTA a unique approach to the buildup process of test automation infrastructure. The

term software repository here refers to a storage location from which software packages or artifacts may be retrieved to be reused on another system or other software products [Almog and Tsubery 2015].

- Achieving Sprint/iteration level planning: some user stories were large and spanned multiple sprints. Scrum teams broke these down to smaller user stories and defined the done criteria for these smaller user stories. This enabled assessing the degree of completion of planned tasks for the sprint and in coordinating activities [Anand and Mani 2015].
- Test Driven Development (TDD): involves the implementation of a system starting from the unit test cases of an object. After writing automated test cases that generally will not even compile, the programmers write implementation code to pass these test cases. The work is kept within programmer's intellectual control; as the programmer is continuously making small implementation decisions and increasing functionality at a relatively consistent rate. All of the test cases that exist for the entire program must successfully pass before new code is considered fully implemented [George and Williams 2004].
- Testing only on Sprints: for the 'test-only' sprints, the test case selection is based on availability of fixes, impacted features, and new test cases created since the last testing cycle, new environment, and test data. In these sprints, complete regression testing at system level is done, as derived from the guidance from the regulatory agencies. In this phase all members of team are involved in testing [Anand and Mani 2015].
- Testing early: before moving to agile, the number of integration tests was lower than the number of system tests. With the aim or executing more test cases early in the life cycle, they tried to identify tests that could equivalently test the feature at an earlier phase. If it was possible for the objective of a system test case to be realized at the integration level, the system test case was moved to the integration test phase [Anand and Mani 2015].
- Training testers on test case design: Senior Test Team members trained for Test Engineers on test case design. This practice suggest the following activities:
  - Use of flowcharts and tables was encouraged during test case design.
  - A set of "do's don'ts" have helped in a uniform understanding of the test case design methodology.
  - Test objective was stated precisely matching the requirements being tested and not as generic statements, which don't imply anything specific.
  - Risk mitigation test cases were treated as a separate set of test cases to give better focus.
  - The data-flow test cases identified and recorded all application attributes and their combinations that would be passed through, on test execution.
- Training related regulatory requirements: a rigorous enforcement of requirement tracing and collecting evidence of testing through screen shots of results and logs during test execution, clear document versioning, keeping a record of document reviews help in complying with regulatory requirements [Anand and Mani 2015].

4. Final Remarks

The Software Development Engineer in Test (SDET) is a formal title for a software tester in the context of software development. In this paper, we conducted a Literature Review

for the period between 1995 and 2016. As a result from the search, we select 130 papers for further analysis and 27 papers that selected. Our results report the expected skills, responsibilities and practices related to SDET role.

SDET is highly skilled resource with development and testing skills. However a tester is involved in preparing and executing the test cases either manually or automatically. Testers are resources with limited programming skills and are focused on functional tests, while SDETs are skilled resources with good programming skills and do the job of tester as well as a developer in the automation of test. Testers are not expected to develop test automation tools, they can use this test automation tool to automate the test cases required for your software application or project. A SDET is proficient in software development, and them can participate in the development of test automation tools and can make it for generic use.

We learned that some of the Expected personal skills of a SDET are highly motivated and responsibility and the technical skills are knowledge of database administration and understanding of system analysis between 14 skills. There are 34 practices associated with the SDT role, but the practice is depend on the context of project is in, but Automated Test and collaboration with stakeholders showed as the most discussed.

It is important to highlight that our results are limited in the sense that are searched only for papers available in databases we have access to. It might be we left on to behind.

5. Acknowledgments

This work is part of a Research Project, sponsored by the PDTI Program, financed by Dell Computers of Brazil Ltd. (Law 8.248/91) under the Dell/PUCRS Research Agreement.

References

Abbattista, F., Bianchi, A., and Lanubile, F. (2009). A storytest-driven approach to the migration of legacy systems. In *Proceedings of the International Conference on Agile Software Development*, pages 149–154, Pula, Italy. Springer.

Abrahamsson, P., Marchesi, M., and Maurer, F. (2009). *Agile Processes in Software Engineering and Extreme Programming*. Springer, Pula, Italy.

Agarwal, N. and Deep, P. (2014). Obtaining better software product by using test first programming technique. In *Proceedings of International Conference-Confluence The Next Generation Information Technology Summit*, pages 742–747, Noida, India. IEEE.

Almog, D. and Tsubery, Y. (2015). How the repository driven test automation (rdta) will make test automation more efficient, easier & maintainable. In *Proceedings of India Software Engineering Conference*, pages 196–197, Bangalore, India. ACM.

Anand, T. and Mani, V. (2015). Practices to make agile test teams effective: challenges and solutions. In *Proceedings of International Conference on Global Software Engineering Workshops*, pages 7–11, Ciudad Real, Spain. IEEE.

George, B. and Williams, L. (2004). A structured experiment of test-driven development. *Information and software Technology*, 46(5):337–342.

Hammond, S. and Umphress, D. (2012). Test driven development: the state of the practice. In *Proceedings of the Annual Southeast Regional Conference*, pages 158–163, Tuscaloosa, USA. ACM.

Haugset, B. and Hanssen, G. K. (2008). Automated acceptance testing: A literature review and an industrial case study. In *Agile Conference*, pages 27–38, Toronto, Canada. IEEE.

Kim, E. and Ryoo, S. (2012). Agile adoption story from Nhn. In *Proceedings of the Annual Computer Software and Applications Conference*, pages 476–481, Izmir, Turkey. IEEE.

Little, T., Elliott, S., Hughes, J., and Simion, F. (2012). Leveraging global talent for effective test agility. In *Agile Conference*, pages 165–171, Dallas, USA. IEEE.

Page, A. and Johnston, K. (2008). *How We Test Software at Microsoft*. Microsoft Press.

Park, S. S. and Maurer, F. (2008). The benefits and challenges of executable acceptance testing. In *Proceedings of the International Workshop on Scrutinizing Agile Practices or shoot-out at the Agile corral*, pages 19–22, Leipzig, Germany. ACM.

Prechelt, L., Schmeisky, H., and Zieris, F. (2016). Quality experience: a grounded theory of successful agile projects without dedicated testers. In *Proceedings of International Conference on Software Engineering*, pages 1017–1027, Austin, USA. ACM.

Rahman, A. A. U., Helms, E., Williams, L., and Parnin, C. (2015). Synthesizing continuous deployment practices used in software development. In *Agile Conference*, pages 1–10, Washington, USA. IEEE.

Smith, M., Miller, J., and Daeninck, S. (2009). A test-oriented embedded system production methodology. *Journal of Signal Processing Systems*, 56(1):69–89.

Solis, C. and Wang, X. (2011). A study of the characteristics of behaviour driven development. In *Proceedings of Conference on Software Engineering and Advanced Applications*, pages 383–387, Oulu, Finland. Austrian Computer Society.

Start (2017). Start — lapes - laboratório de pesquisa em engenharia de software. "`http://lapes.dc.ufscar.br/tools/start_tool`" -Accessed in September 2017.

Sultanía, A. K. (2015). Developing software product and test automation software using agile methodology. In *Proceedings of the International Conference on Computer, Communication, Control and Information Technology*, pages 1–4, Hooghly, India. IEEE.

Tripp, J. F., Riemenschneider, C., and Thatcher, J. B. (2016). Job satisfaction in agile development teams: Agile development as work redesign. *Journal of the Association for Information Systems*, 17(4):267.

Williams, L., Brown, G., Meltzer, A., and Nagappan, N. (2011). Scrum + engineering practices: Experiences of three microsoft teams. In *Proceedings of the International Symposium on Empirical Software Engineering and Measurement*, pages 463–471, Washington, USA. IEEE.

Lista de Autores

Amorim, N. e Guedes, G. T., 127
Anschau, L. e Melo, A., 25

Belinazo, G. e Fontoura, L., 191
Borgens, C., *et. al.*, 151
Bortoli, L. A., 111

Carbonell, J., *et. al.*, 49

de Lima, F., *et. al.*, 119
dos Santos, D. M., *et. al.*, 81
Dutra, A., *et. al.*, 105

Ecar, M., 9
Ecar, M. e da Silva, J. P. S., 41

Ferrão, I. e Kreutz, D., 135

Iung, A., *et. al.*, 57

Junior, M. V. e Schepke, C., 73

Krug, T., *et. al.*, 143

Lima, Y., *et. al.*, 65
Lopes, J., *et. al.*, 97

Marchezan, L., *et. al.*, 1

Pereira, J. B. e Melo, A., 17

Rodrigues, I., *et. al.*, 181
Rodrigues, P., *et. al.*, 89

Torres, R., *et. al.*, 33
Toscani, C., *et. al.*, 211

Zanatta, A., *et. al.*, 201
Zanin, A. e Zorzo, A., 161
Zanin, A., *et. al.*, 171