X CBSOFT

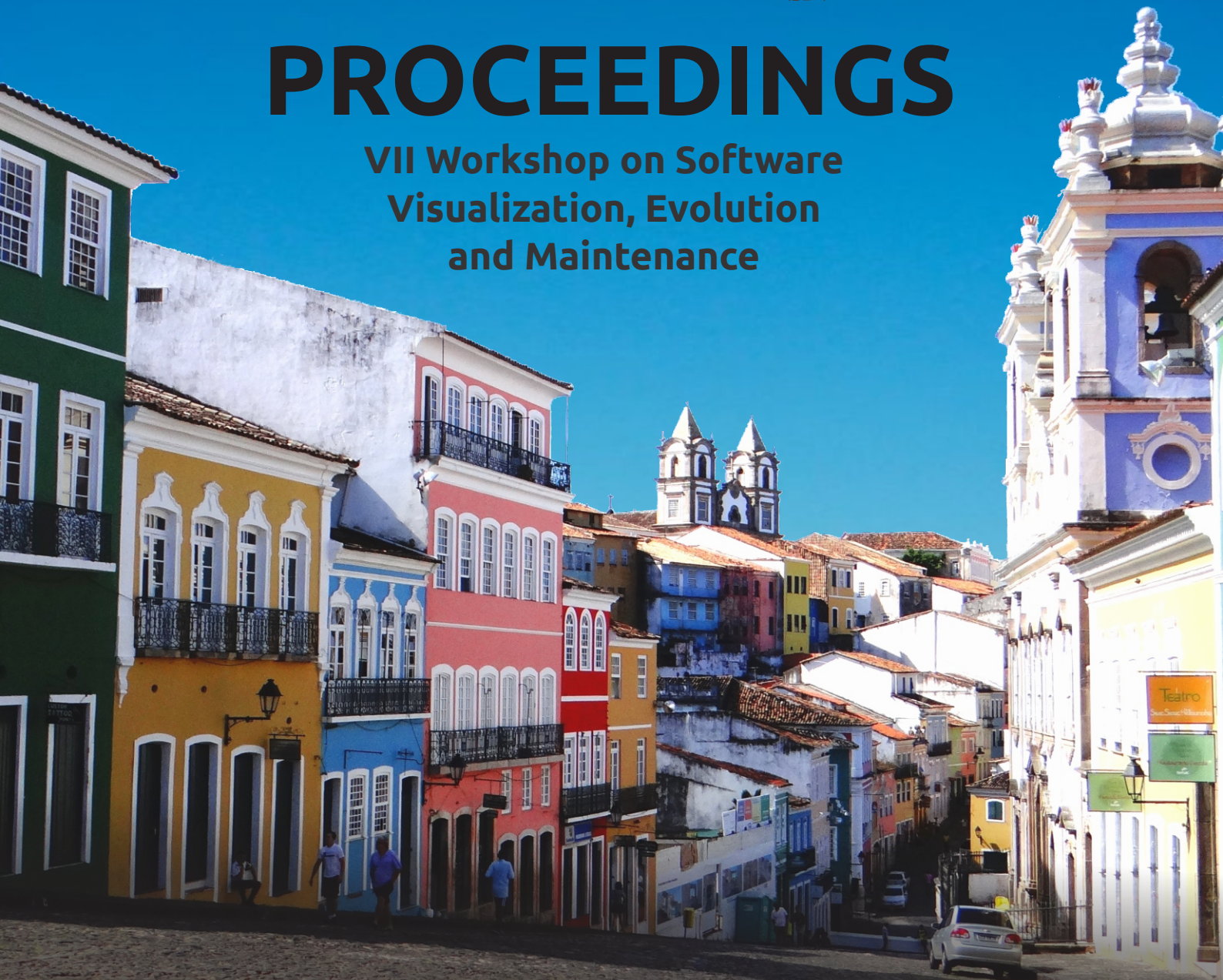
SALVADOR

BRAZILIAN CONFERENCE ON SOFTWARE

23RD - 27TH SEPTEMBER, 2019

PROCEEDINGS

VII Workshop on Software Visualization, Evolution and Maintenance



Promotion

Organization

Supporter

Sponsor

Proceedings

VII Workshop on Software Visualization, Evolution and Maintenance

X Brazilian Conference on Software: Theory and Practice (CBSoft 2019)

VII Workshop on Software Visualization, Evolution and Maintenance

September 23–27, 2019
Salvador, BA, Brazil

PROCEEDINGS Sociedade Brasileira de Computação (SBC)

COVER ART

Railana Santana (UFBA)
Nildo Junior (UFBA)

STUDENT VOLUNTEERS CHAIR

Larissa Rocha (UFBA)

PROMOTION

Sociedade Brasileira de Computação (SBC)

ORGANIZATION

Departamento de Ciência da Computação,
Instituto de Matemática e Estatística —
Universidade Federal da Bahia (DCC/UFBA)

SUPPORT

Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq)
Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES)

SPONSORS

TecnoTRENDS Tecnologia Educacional
Google

Foreword

The VII Workshop on Software Visualization, Evolution and Maintenance (VEM 2019) is part of the X Brazilian Congress on Software: Theory and Practice (CBSoft 2019), held in Salvador – Bahia, from September 23 to 27, 2019. Its main goal is to foster the integration of the software visualization, evolution and maintenance communities, providing a Brazilian forum where researchers, students and professionals can present their work and exchange ideas on the principles, practices and innovations related to their respective areas of interest. The VEM 2019 Program Committee (PC) is composed of 49 active researchers in the areas of software visualization, evolution and maintenance, who come from several regions of Brazil. The PC members selected 11 interesting and promising papers to be presented at VEM 2019, from a total of 18 submissions. Each submission was evaluated by at least three PC members, based on their originality, technical quality and adequacy to the event's scope.

In addition, the VEM 2019 technical program also includes two invited keynote talks where the event participants could discuss the main problems and solutions related to software visualization, evolution and maintenance.

Finally, we would like to express our deepest gratitude to all authors who submitted their work to VEM 2019, for their interest, to the PC members, for their effort and invaluable collaboration during the paper selection process, and to the CBSoft 2019 organizers and sponsors, for their support and contribution.

Fernando Castor (UFPE) and Igor Scaliante Wiese (UTFPR)
Program Co-Chairs - VEM 2019

Program Chairs

Fernando Castor (UFPE)
Igor Scaliante Wiese (UTFPR)

Program Committee

Andre Hora - Federal University of Minas Gerais
Auri Vincenzi - Federal University of São Carlos
Baldoino Fonseca - Federal University of Alagoas
Bruno Cafeo - Federal University of Mato Grosso do Sul
Bruno da Silva - California Polytechnic State University
Christina Chavez - Federal University of Bahia
Claudio Sant'Anna - Federal University of Bahia
Eduardo Figueiredo - Federal University of Minas Gerais
Eduardo Guerra - National Institute of Space Research
Eiji Adachi Barbosa - Federal University of Rio Grande do Norte
Elder Cirilo - Federal University of São João del-Rei
Fabio Petrillo - University of Quebec at Chicoutimi
Felipe Ebert - Federal University of Pernambuco
Fernanda Madeiral Delfim - Federal University of Uberlândia
Fernando Castor - Federal University of Pernambuco
Glauco de Figueiredo Carneiro - University of Salvador
Guilherme Avelino - Federal University of Piauí
Gustavo Pinto - Federal University of Pará
Heitor Costa - Federal University of Lavras
Henrique Rocha - INRIA
Humberto Marques-Neto - Pontifical Catholic University of Minas Gerais
Igor Steinmacher - Federal University of Technology - Paraná
Igor Wiese - Federal University of Technology - Paraná
Ingrid Nunes - Federal University of Rio Grande do Sul
Ivan Machado - Federal University of Bahia
Kecia Ferreira - CEFET-MG
Leopoldo Teixeira - Federal University of Pernambuco
Lincoln Rocha - Federal University of Ceará
Luciana Silva - Federal Institute of Minas Gerais
Marcelo Maia - Federal University of Uberlandia
Marcelo Schots - Rio de Janeiro State University
Marco Tulio Valente - Federal University of Minas Gerais
Maria Istela Cagnin - Federal University of Mato Grosso do Sul
Marx Viana - Pontifical Catholic University of Rio de Janeiro
Nabor Mendonca - University of Fortaleza
Patrick Brito - Federal University of Alagoas
Rafael Durelli - Federal University of Lavras
Regina Braga - Federal University of Juiz de Fora
Renato Novais - Federal Institute of Bahia
Ricardo Terra - Federal University of Lavras
Roberta Coelho - Federal University of Rio Grande do Norte
Rodrigo Bonifacio - University of Brasilia
Rodrigo Souza - Federal University of Bahia
Rogerio de Carvalho - Fluminense Federal Institute
Rogerio Garcia - São Paulo State University
Rosana Braga - São Paulo University
Uirá Kulesza - Federal University of Rio Grande do Norte
Valter Camargo - Federal University of São Carlos
Vinicius Durelli - Federal University of São João del-Rei

Invited Speakers

Christoph Treude

University of Adelaide, Australia

Christoph Treude is an ARC DECRA Fellow and a Senior Lecturer in the School of Computer Science at the University of Adelaide, Australia. He completed his PhD in Computer Science at the University of Victoria, Canada, and spent two years in Brazil as a postdoctoral researcher at DIMAp/UFRN and IME/USP. The goal of his research is to advance collaborative software engineering through empirical studies and the innovation of processes and tools that explicitly take the wide variety of artefacts available in a software repository into account.

Uirá Kulesza

Federal University of Rio Grande do Norte, Brazil

Uirá Kulesza is an associate professor in the Department of Informatics and Applied Mathematics (DIMAp), Federal University of Rio Grande do Norte (UFRN). He received his PhD in computer science from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio) in Brazil, in cooperation with University of Waterloo (Canada) in 2007. He has a PQ-2 research grant from CNPq since 2010. He currently heads the Collaborative & Automated Software Engineering (CASE) Lab and Research Group at Digital Metropolis Institute (IMD), UFRN. His main research interests include software evolution, software architecture, and software analytics.

Table of Contents

Keynotes

Papers

Bite-sized software documentation

Christoph Treude
University of Adelaide, Australia

Keynote

Abstract

Software documentation is like farofa in Brazilian cuisine: it is an essential ingredient, it can be a bit dry, and it comes in many different forms. Finding the best parts (i.e., the bacon) can be difficult, and we need tools to uncover them. To help software developers uncover the best parts of software documentation, we are developing approaches to get information to them when and where they need it. This talk will highlight several approaches, including a task-based search interface for software documentation, an approach for improving compiler error messages with content from Stack Overflow, and a badge-generator for sections of GitHub README files. While much of the information needed by software developers is already available somewhere, we still have a long way to go to give developers the best tools for transforming documentation into bite-sized pieces.

Understanding the Impact of Continuous Integration in Open Source Projects

Uirá Kulesza
Federal University of Rio Grande do Norte, Brazil

Keynote

Abstract

Continuous Integration (CI) is the practice of automating and improving the frequency of code integration. Software projects have adopted CI to ship new releases more frequently and to improve code integration. Recent research has analyzed the benefits that CI can bring to open-source projects by using mining software repositories techniques. In this talk, I will present and discuss some results of empirical studies conducted by our research group that analyzed the impact of adopting CI practices on the delivery time of pull requests and in the evolution of automated test artifacts.

Análise de Sentimentos em Discussões de Issues Reabertas do Github

Gláucya Boechat, Joselito Mota Jr, Ivan Machado, Manoel Mendonça

[1]Universidade Federal da Bahia (UFBA)
Campus Ondina – 40.170-110 – Salvador – BA – Brazil

{glaucya.boechat, ivan.machado, manoel.mendonca}@ufba.br,

joselitomota@dcc.ufba.br

Abstract. *The behavior of reopened issues is a perception to be studied to analyze the impact of discussions on the continuity of software project maintenance. Sentiment analysis is presented as a powerful technique to assist such analysis. In this study, we analyzed 12,996 reopened issues, which contained discussions, from 80 Github projects. Based on the analysis of such historical data, we seek to analyze whether a closed issue tends to be reopened from the sentiment analysis of this issue's discussions. The analyzes are performed through the degree of sentiment of the texts of the comments of the issues. The SentiStrength tool, supported by Software Engineering lexicons, were used to classify the degree of polarity of the texts found. The study identified that the polarity of feelings in discussions can directly affect the issue's life cycle, including support for the prediction about reopening issues.*

Resumo. *O comportamento de issues reabertas é uma percepção a ser estudada para analisar o impacto das discussões na continuidade da manutenção de projetos de software. A análise de sentimentos apresenta-se como uma poderosa técnica para auxiliar tal análise. Neste estudo, analisamos 12.996 issues reabertas, contendo discussões, de 80 projetos do Github. Com base na análise dessa massa de dados históricos, buscamos analisar se uma issue fechada tende a ser reaberta a partir da análise de sentimentos das discussões dessa issue. As análises são realizadas através do grau de sentimento dos textos dos comentários das issues. A ferramenta SentiStrength, com suporte aos léxicos da área de Engenharia de Software, foi utilizada para classificar o grau de polaridade dos textos encontrados. O estudo identificou que a polaridade dos sentimentos nas discussões pode afetar diretamente o ciclo de vida da issue, inclusive com suporte à predição sobre a reabertura das issues.*

1. Introdução

Uma atividade fundamental da fase de manutenção do software é compreender porque *issues* fechadas são reabertas [Caglayan et al. 2012]. As *issues* podem ser reabertas depois de serem fechadas devido ao fechamento incorreto, descoberta do problema real da *issue*, etc. As *issues* reabertas podem aumentar o custo de manutenção, degradar a qualidade geral do produto de software, reduzir a confiança dos usuários e trazer trabalho desnecessário para os desenvolvedores [Pan and Mao 2014].

Nesse trabalho, optamos por investigar o comportamento das *issues* e *pull requests* que foram reabertas no GitHub. O GitHub é uma plataforma de hospedagem de código-fonte do sistema de controle de versão git, que contém mais de 36 milhões de usuários e 100 milhões de repositórios [1].

No GitHub, os colaboradores dos repositórios, opcionalmente, podem criar e participar de discussões de *issues* para coletar feedback dos usuários nas discussões sobre o projeto, adição de novas features, bugs e outras tarefas de manutenção. Eles podem ainda participar de discussões de *pull request*, que é um tipo de *issue*, para discutir e revisar alterações realizadas no código-fonte antes de serem incorporadas na *branch* principal [Ortu et al. 2016]. Os comentários dessas discussões das *issues* postados pelos colaboradores não contêm apenas informações técnicas, mas também informações valiosas sobre sentimentos ou emoções [Ortu et al. 2015]. Os sentimentos podem ser classificados como positivo, negativo ou neutro, que estão associados as emoções como felicidade, tristeza, alegria, raiva, dentre outras [Liu 2015]. *A investigação sobre os sentimentos contidos nas issues pode trazer informações ou indicativos que auxiliam no gerenciamento desses repositórios, por exemplo ...* .

O objetivo do trabalho consiste em investigar os sentimentos dos colaboradores durante as discussões das *issues* que ajudam a prever se uma *issue* fechada tem propensão de ser reaberta. As seguintes questões de pesquisa (QP) direcionam esta investigação:

QP1 Existe algum indicativo de que uma *issue* não será reaberta se ela for fechada com um sentimento positivo?

QP2 É possível predizer se uma *issue* será reaberta quando o número de sentimentos negativos adicionados ao número de sentimentos neutros for maior que o número de sentimentos positivos presentes nas suas discussões?

QP3 Uma *issue* com discussões com sentimentos neutros após o fechamento indica que ela será reaberta?

2. Metodologia

Inicialmente, foram selecionados os 90 repositórios listados no desafio *MSR Challenge Dataset* da conferênca *Mining Software Repositories* (MSR)[2] do ano de 2014. Ao validalos, observamos que oito repositórios não possuiam *issues* reabertas com discussões, e dois repositórios não mais encontravam-se disponíveis. Assim, a lista final incluiu os 80 repositórios disponíveis, que englobavam 506.227 *issues* abertas e fechadas, com uma média de aproximadamente 379 mil linhas de código (*LOC*), e média de 2.769 arquivos por repositório. As *issues* foram extraídas entre 4 de Junho à 24 Julho de 2019. Selecionamos apenas as *issues* que foram reabertas e que possuíam discussões, resultando em 12.996 issues no *dataset*. A Figura 1 apresenta um *workflow* com as etapas seguidas neste trabalho. Cada uma das etapas será discutida a seguir.

Os repositórios selecionados foram implementados nas seguintes linguagens: 8 em C, 8 em C++, 8 em C#, 1 em CSS, 3 em HTML, 7 em Java, 8 em JavaScript, 3 em Markdown, 5 em PHP, 10 em Python, 4 em R, 8 em Ruby e 7 em Scala.

[1] Dados de https://github.com/about

[2] http://ghtorrent.org/msr14.html

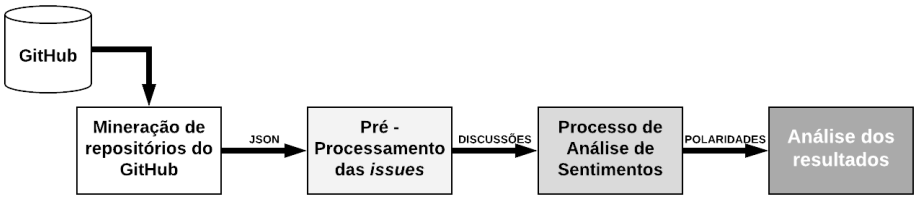


Figura 1. *Workflow* de captura e análise de dados.

Todos os dados dos repositórios utilizados neste projeto encontram-se disponíveis em [Boechat et al. 2019].

2.1. Etapa 1 - Mineração de Repositórios do Github

Os dados foram obtidos a partir de um script de mineração desenvolvido utilizando a biblioteca PyGitHub[3] para extração dos dados. Este script é reponsável por recuperar todas as informações referentes as *issues*: comentários, horário de criação, usuários, reações e outras informações. Processamos todas as respostas das requisições ao GitHub no formato *JavaScript Object Notation* (JSON) e em seguida armazenamos todas as informações em um banco de dados não relacional, MongoDB[4]. O script encontra-se disponível no GitHub[5]. A mineração das *issues* foi realizada em uma estação de trabalho com um processador Core i7-7500U, 8 GB RAM, SSD 240 GB, Sistema Operacional Win 10 x64.

2.2. Etapa 2 - Pré-processamento das *issues*

A etapa de pré-processamento dos dados foi realizada através da limpeza dos textos do título, descrição e comentários das issues. A limpeza dos textos foi feita através do módulo RE[6] do Python, com operações com expressões regulares para excluir trechos indesejados no texto, tais como URLs, código-fonte, trechos de código, erros de compilação, classes, interfaces, imagens, quebra de linhas, excesso de espaços em branco, respostas de comentário, frases de alertas sobre *warning* e exceções.

O pré-processamento considerou o truncamento de palavras, por exemplo *joy** para todas as palavras que começam com *joy*, ao invés de utilizar os processos de lematização e stemização. Não utilizamos a remoção de *stopwords*, pois pode ocorrer de remover palavras que distorcem o verdadeiro sentimento da frase, por exemplo os advérbios de negação ou intensidade podem alterar o sentido da frase [Thelwall et al. 2010].

2.3. Etapa 3 - Processo de Análise de Sentimentos

Durante o processo de análise de sentimentos das discussões das *issues* reabertas, e que tenham pelo menos dois comentários, foi utilizada a versão Java da ferramenta SentiStrength [Thelwall et al. 2010] para classificar as polaridades dos textos. A polaridade pode ser *negativa*, *neutra* ou *positiva*. A SentiStrength é uma ferramenta de análise de sentimentos baseada em um dicionário léxico; esse dicionário é um arquivo léxico de

[3]https://pygithub.readthedocs.io
[4]https://www.mongodb.com/
[5]https://github.com/joselitojunior94/gfetcher
[6]https://docs.python.org/3/library/re.html

emoções, onde palavras com sentimentos negativos estão previamente classificadas com pesos entre -5 e -1, e palavras com sentimentos positivos possuem pesos entre +1 e +5. Utilizamos o dicionário léxico da ferramenta SentiStrength-SE[7] [Islam and Zibran 2017], uma versão desenvolvida para aplicar análise de sentimentos no domínio de Engenharia de Software. A ferramenta SentiStrength divide a sentença em *tokens* e para cada palavra (*token*) que transmite uma emoção é atribuída uma pontuação. Após pontuar todas as palavras, a ferramenta retorna a pontuação máxima dos sentimentos negativos e a pontuação máxima dos sentimentos positivos. O sentimento final do texto é obtido através da soma da pontuação positiva com a pontuação negativa. Para valores menores que zero, o texto é classificado como negativo; para valores iguais a zero, o texto é classificado como neutro; e para valores maiores que zero, o texto é classificado como positivo.

2.4. Etapa 4 - Análise dos Resultados

A ordem cronológica dos eventos de uma *issue* foram consideradas na análise dos resultados. Os eventos são apresentados no diagrama de estado da Figura 2, que exibe os possíveis status da *issue*: "ABERTA" ("OPEN") e "FECHADA" ("CLOSED"), e as transições responsáveis pelas mudanças de status ("FECHAR" e "REABRIR").

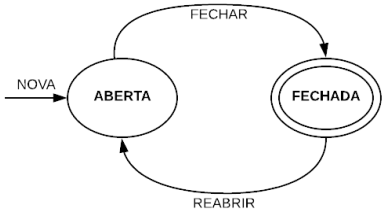


Figura 2. Ciclo de Vida da *issue*.

Durante o ciclo de vida de uma *issue*, os colaboradores do repositório e/ou usuários do GitHub podem colabarorar com o repositório através de comentários nas *issues*. Nos comentários, os colaboradores podem expressar seus sentimentos por meio de textos, emoticons e emojis. Neste estudo, foram analisadas as polaridades dos sentimentos encontradas nos comentários no período de tempo entre os status possíveis da *issue*. A Figura 3 representa o ciclo de vida de uma *issue* reaberta com discussões. Ela apresenta o início da *issue*, no status "Aberta", as discussões dos colaboradores entre a abertura e o fechamento, a mudança de status para "Fechada", as discussões dos colaboradores entre o fechamento e a reabertura da *issue*, a mudança de status para "Reaberta", as discussões dos colaboradores entre a reabertura e o fechamento da *issue*, e por fim a mudança de status para "Fechada".

3. Discussão dos Resultados

Foram analisadas 12.996 *issues* reabertas, que continham discussões, de 80 repositórios do GitHub. A seguir, discutiremos os resultados à luz das QP propostas para este estudo.

QP1 - Existe algum indicativo de que uma *issue* não será reaberta se ela for fechada com um sentimento positivo?

Para calcular os resultados, verificou-se o último sentimento antes do fechamento da *issue*. Assim, caso a *issue* possua pelo menos um fechamento, então é contabilizado.

[7]https://laser.cs.uno.edu/Projects/Projects.html

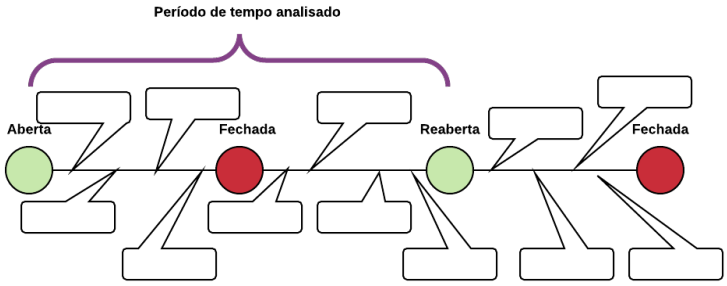


Figura 3. Linha do tempo dos eventos da *issue*.

O ponto central da análise de dados foi encontrar pelo menos um caso durante o tempo de vida de uma *issue* onde existiu fechamento com sentimento positivo, e reabertura logo em seguida. Verificamos que 2.925 (22,51%) *issues* reabertas foram fechadas com sentimento positivo e em seguida foram reabertas. A Figura 4 apresenta a distribuição de issues fechadas com sentimentos positivos com mediana de 14 *issues*. Isso indica que se a *issue* fechar com sentimento positivo, não há garantia de que ela continuará fechada. Entretanto, observa-se uma menor tendência de reaberta dessas *issues*. 10.071 (77,49%) *issues* foram fechadas com sentimento negativo ou neutro, e em seguida foram reabertas.

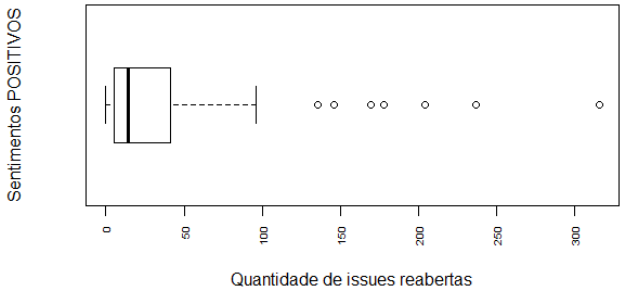


Figura 4. Distribuição de *issues* fechadas com sentimentos positivos.

Dentre os repositórios analisados, apenas o `Impress.js` teve 13 de 18 (72.22%) *issues* reabertas que foram fechadas com sentimento positivo e em seguida foram reabertas. Em contraponto, nenhum caso foi observado nos repositórios `ActionBarSherlock`, `Kestrel` e `Storm` [Boechat et al. 2019].

QP2 - É possível predizer se uma *issue* será reaberta quando o número de sentimentos negativos adicionados ao número de sentimentos neutros for maior que o número de sentimentos positivos presentes nas suas discussões?

Verificamos que 11.722 (90.20%) *issues* foram reabertas com a quantidade de sentimentos negativos ou neutros maior que a quantidade de sentimentos positivos. Encontramos 810 (6,23%) *issues* reabertas com quantidade de sentimentos negativos ou neutros igual a quantidade de sentimentos positivos. Em 464 (3,57%) *issues*, a quantidade de sentimentos positivos foi maior que a quantidade de sentimentos negativos ou neutros. A Figura 5 apresenta a distribuição de *issues* reabertas com mediana de de 57 *issues* para (Neutros+Negativos) > Positivos, em função da polaridade de sentimentos. Nos repositórios `ActionBarSherlock`, `Storm`, `Beanstalkd`, `Kestrel`, `Flockdb`, `Ravendb` e `CCV`, 100% das *issues* foram reabertas com sentimentos negativos ou neutros

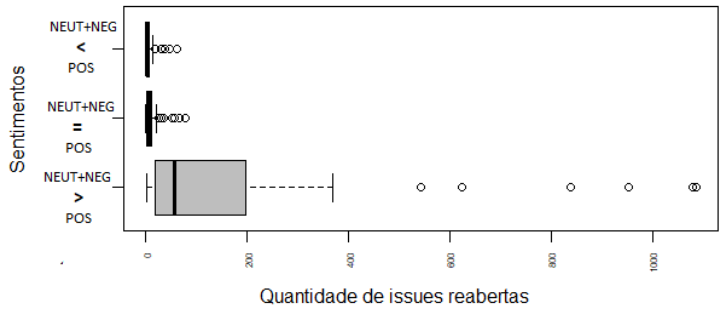


Figura 5. Distribuição de *issues* reabertas em função da polaridade de sentimentos.

maior que a quantidade de sentimentos positivos. Os dados permitiram observar que, se houve uma tendência de discussão negativa ou neutra durante o período de atividade da *issue*, então a propensão de reabertura é maior.

QP3 - Uma *issue* com discussões com sentimentos neutros após o fechamento indica que ela será reaberta?

Encontramos 5.547 (42,68%) *issues* reabertas que possuem comentários com sentimentos neutros entre o fechamento da *issue* e a sua reabertura. A Figura 6 apresenta a distribuição de *issues* que possuem sentimentos neutros entre fechamento e reabertura. 2.079 (16,00%) *issues* reabertas possuem comentários com sentimentos negativos entre o fechamento da *issue* e a sua reabertura. A mediana da Figura 6 foi de 24 *issues*. A *issue* pode ser reaberta quando há sentimentos neutros relacionados, mas existe uma propensão maior de reabertura de *issues* com sentimentos neutros do que com sentimentos negativos. Algumas *issues* observadas foram reabertas sem discussões, portanto não foram contabilizadas.

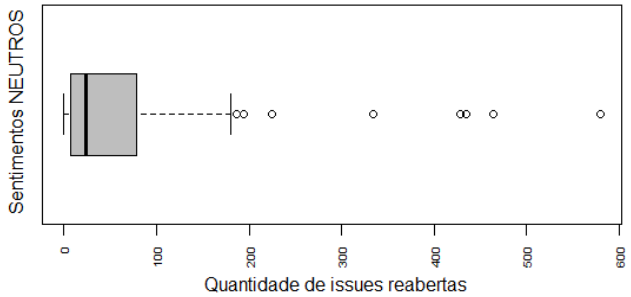


Figura 6. Distribuição de *issues* que possuem sentimentos neutros entre fechamento e reabertura.

4. Ameaças à validade

Validade Interna. A mineração dos dados foi realizada em um período de tempo que começou em 4 de Junho à 24 Julho de 2019. Assim, repositórios ativos podem receber atualizações com novas *issues*, comentários ou mudança de estado (aberta/fechada), alterando a quantidade de dados da nossa base. Como a lista escolhida foi do desafio *MSR Challenge Dataset*, ocorrido em 2014, dentre os repositórios minerados também encontramos alguns que estão arquivados ou são inativos. Entretanto, o cenário utilizado no estudo possui uma quantidade significativa de *issues*.

Validade Externa. Alguns repositórios possuem características que não favorecem o estudo, possuindo poucas *issues*. Resta, portanto, um número menor de *issues* reabertas para serem analisadas. Entretanto, uma menor quantidade propicia uma análise mais criteriosa e assertiva, por serem repositórios verificados e utilizados como referência nos estudos presentes na literatura.

Validade de Construto. Ao passo que havendo uma base de dados com múltiplos domínios distintos, os repositórios possuem tamanhos diferentes entre si. Há uma tendência de alguns repositórios possuírem maior quantidade de dados que os demais. Dessa forma, certos domínios impactam significativamente no resultado. Assim, para encontrar resultados mais homogêneos é preciso expandir a base de dados minerados do GitHub para integrar mais elementos relevantes ao estudo.

5. Trabalhos Relacionados

A pesquisa de [Caglayan et al. 2012] caracterizou os possíveis fatores que podem causar a reabertura de *issues*. Eles identificaram que a rede de proximidade das *issues* e a atividades dos desenvolvedores foram os fatores mais importantes para reabertura das *issues*.

A investigação de [Shihab et al. 2013] identificou que os principais fatores que causam a reabertura de *issues* relacionadas a bugs no Eclipse foram a descrição da *issue*, os textos dos comentários da *issue*, o tempo para solucionar o bug e o componente onde o bug foi encontrado.

Os autores [Zimmermann et al. 2012] categorizaram as principais razões que levam as *issues* relacionadas a defeitos (bugs) serem reabertas com base em um survey realizado com 358 colaboradores da Microsoft. Eles usaram fatores relacionados a processos e organizacionais, incluindo a localização, hábitos de trabalho, além de fatores extraídos dos relatório de *issues* (*issue report*).

O trabalho de [Souza et al. 2015] analisou o projeto Firefox para compreender as relações entre as diferentes formas de rejeição de *patch*. Foi identificado que cerca de 5,7% de todas as *issues* resolvidas do Firefox foram reabertas, o que induziu uma sobrecarga de discussões entre os colaboradores sobre a reabertura de uma *issue*. Também foi identificado que 70% das *issues* fechadas foram reabertas permaturamente devido a equívocos de interpretação dos colaboradores.

6. Considerações Finais

O presente estudo analisou o sentimento de cerca de 13 mil *issues* reabertas (que incluíram cerca de 153 mil comentários) em 80 repositórios de projetos hospedados no GitHub, buscando respostas sobre a previsão de reabertura de *issues* através da análise dos sentimentos presentes nas discussões dos colaboradores. A ferramenta SentiStrength foi fundamental para a classificação do grau de polaridade dos textos encontrados.

O impacto dos sentimentos nas discussões em alguns casos pode afetar ou não diretamente o ciclo de vida da *issue*. Identificado na primeira questão de pesquisa, a reabertura de uma *issue* com sentimento positivo existe, mas ela é menos recorrente do que com outros sentimentos. Também são questionados sobre a tendência da discussão durante a reabertura, que na segunda questão de pesquisa é observado que discussões negativas e neutras podem influenciar diretamente na reabertura de uma *issue*. Durante o

período de fechamento e reabertura de uma *issue*, existe um indicativo maior de reabertura se as discussões forem realizadas com sentimento neutro do que se forem realizadas com sentimento negativo. O estudo mostra a importância da análise de sentimentos para o gerenciamento de repositórios de software. As informações extraídas são indicativos que podem ajudar no gerenciamento do projeto, uma vez que *issues* fechadas podem ser identificados para posterior reabetura.

Como trabalhos futuros, planejamos correlacionar linguagens de programação, tipo de domínio e tipos de colaboradores com a análise de sentimentos. Também existe a necessidade de expandir a base de dados com uma maior amostra de repositórios.

Agradecimentos: O presente trabalho foi realizado com apoio do CNPq e da FAPESB.

Referências

Boechat, G., Júnior, J. M., Machado, I., and Mendonça, M. (2019). Análise de sentimentos em discussões de issues reabertas do github (material suplementar). Zenodo. `http://doi.org/10.5281/zenodo.3376175`.

Caglayan, B., Misirli, A. T., Miranskyy, A., Turhan, B., and Bener, A. (2012). Factors characterizing reopened issues: A case study. In *Proceedings of the 8th Int. Conf. on Predictive Models in Soft. Engineering*, pages 1–10, New York, USA. ACM.

Islam, M. R. and Zibran, M. F. (2017). Leveraging automated sentiment analysis in software engineering. In *14th Int. Conf. on Min. Soft. Repositories(MSR)*, pages 203–214.

Liu, B. (2015). *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. C.U.P.

Ortu, M., Destefanis, G., Adams, B., Murgia, A., Marchesi, M., and Tonelli, R. (2015). The jira repository dataset: Understanding social aspects of software development. In *Proceedings of the 11th Int. Conf. on Predictive Models and Data Analytics in Software Engineering (PROMISE)*, pages 1:1–1:4, New York, NY, USA. ACM.

Ortu, M., Murgia, A., Destefanis, G., Tourani, P., Tonelli, R., Marchesi, M., and Adams, B. (2016). The emotional side of software developers in jira. In *Proceedings of the 13th Int. Conf. on Mining Soft. Repositories(MSR)*, pages 480–483, NY, USA. ACM.

Pan, J. and Mao, X. (2014). An empirical study on interaction factors influencing bug reopenings. In *21st Asia-Pacific Soft. Engineering Conf.*, volume 2, pages 39–42.

Shihab, E., Ihara, A., Kamei, Y., Ibrahim, W. M., Ohira, M., Adams, B., Hassan, A. E., and Matsumoto, K.-i. (2013). Studying re-opened bugs in open source software. *Empirical Software Engineering*, 18(5):1005–1042.

Souza, R. R., Chavez, C. F., and Bittencourt, R. A. (2015). Patch rejection in Firefox: negative reviews, backouts, and issue reopening. *J. of Soft. Eng. Res. and Dev.*, 3(1).

Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., and Kappas, A. (2010). Sentiment strength detection in short informal text. *J. Am. Soc. Inf. Sci. Technol.*, 61(12):2544–2558.

Zimmermann, T., Nagappan, N., Guo, P. J., and Murphy, B. (2012). Characterizing and predicting which bugs get reopened. In *34th Int. Conf. on Software Engineering (ICSE)*, pages 1074–1083.

BULNER: BUg Localization with word embeddings and NEtwork Regularization

Jacson Rodrigues Barbosa[1,5], Ricardo Marcondes Marcacini[3], Ricardo Britto[4], Frederico Soares[6], Solange Rezende[1], Auri M. R. Vincenzi[2], Márcio E. Delamaro[1]

[1]ICMC-Universidade de São Paulo (USP)

[2]DC-Universidade Federal de São Carlos (UFSCar)

[3]Universidade Federal de Mato Grosso do Sul (UFMS)

[4]Ericsson AB / Blekinge Institute of Technology

[5]INF-Universidade Federal de Goiás (UFG)

[6]Tribunal de Justiça do Estado de Goiás

jacsonrb@usp.br, ricardo.marcacini@ufms.br, ricardo.britto@[ericsson.com,bth.se],

fasoares@tjgo.jus.br, {solange,delamaro}@icmc.usp.br, auri@dufscar.br

Abstract. *Bug localization (BL) from the bug report is the strategic activity of the software maintaining process. Because BL is a costly and tedious activity, BL techniques information retrieval-based and machine learning-based could aid software engineers. We propose a method for BUg Localization with word embeddings and Network Regularization (BULNER). The preliminary results suggest that BULNER has better performance than two state-of-the-art methods.*

1. Introduction

Bug localization (BL) from bug reports is an expensive step in the software life cycle because of the manual process of localization. For example, the Mozilla project receives almost 300 bug reports per day, and each one needs a manual triage. Also, often, a bug report (84-93% of bugs) impacts one or a few files [Thung et al. 2014]. There is a family of bug localization techniques that uses Information Retrieval (IR). IR suggests defective parts of a software system by automatically relating a bug report's vocabulary and associated source code metrics. IR often uses Vector Space Model (VSM) but, due to VSM limitations, recent studies apply distributional semantics of words [Rahman and Roy 2018].

In this paper, we propose BULNER, an IR-based bug localization method, which stands for BUg Localization with word embeddings and Network Regularization. BULNER considers both word embedding features of bug reports and features extracted from project source file metrics. We combined these features in an information network proposed in BULNER. We present a network regularization-based machine learning method that obtains a more appropriate representation model for identifying potential buggy files from bug report texts. Our research answers the following research questions: **RQ1** - How effective is BULNER? **RQ2** - What is the contribution of each model? We carried out an experimental evaluation using three well-known real-world datasets.

BULNER is competitive with two other state-of-the-art methods, and the experimental results indicate that combining different representation models, such as information networks and the vector-space model, is a promising method.

2. Bug Localization Data Model Representation

Bug localization has been modeled as an information retrieval task, where the bug report is treated as a query, and sources code files that conform to the system are documents. The goal is to select the files that better match the query based on a defined similarity measure. The effectiveness of the similarity measure depends on the text representation model of bug reports, often based on Bag-of-Words (BoW) and Word Embeddings (WEmb).

BoW representation uses terms (e.g., keywords) extracted from texts as features in a vector space model. BoW is a document-term matrix, where each row (vector) represents a document, each column represents a term (word) present in the document collection, and each cell contains a measure (term frequency and inverse document frequency) [Tan et al. 2005]. BoW representation has as main characteristics of the high dimensionality and the high sparsity. However, these characteristics affect the performance of machine learning algorithms negatively. Furthermore, machine learning algorithms are not able to infer relations between terms or relations between documents as they are not established by BoW.

In bug localization context, each source code is defined as weights' vector in BoW, and they use cosine similarity function to identify closely related vector. Zhou *et al.* define BugLocator, an IR-based bug localization method based on revised Vector Space Model. From the initial bug report, BugLocator applies textual similarity using similar bugs' information that was fixed before. Then, it ranks all suspicious sources files [Zhou et al. 2012].

WEmb are a mapping table from words to continuous vectors (e.g., vec("dog") = [0.8, 0.3, 0.1], vec("cat") = [0.7, 0.5, 0.1] , vec("pasta") = [0.2, 0.1, 0.7]). In this example, the first parameter of each word represents some kind of animal. We could calculate the semantic similarity between words by cosine similarity and consequently calculate the similarity between sentences or entire documents. To obtain each above vector, we use Skip-gram model proposed in the word2vec method for language modeling [Mikolov et al. 2013]. It is a unsupervised method that define meaning each word in its context (e.g., context("dog") = ["Pet," "tail," "smell,"], context("cat") = ["pet," "tail," "home"]). Two sets of context words also have common conceptually. According to the distributional hypothesis, we can estimate how close these two words are to each other by comparing with others in same context [Mikolov et al. 2013].

Ye *et al.* use word embedding to train on software documents (API documents, reference documents, and tutorials). They adapt the Skip-gram model and aggregated software documents to estimate semantic similarities between them. Then, from an initial bug report (query document), the bug localization model computes the ranking score for all source code [Ye et al. 2016].

3. Proposed Method

In this section, we introduce a new method for Bug Localization, called BULNER. Our method innovates by considering both the semantic content of bug reports through

language models and source code content through code metrics. Figure 1 shows an overview of the BULNER method. The method has two stages: (1) language modeling and (2) network regularization. Given a new bug report, BULNER identifies the most similar bug reports, and the source code files that probably contain the related bug. While the BULNER's first stage enables more accurate computation of the similarity between bug reports, the second performs fine-tuning of the language model by considering relationships between bug reports, source code files, and code metrics.

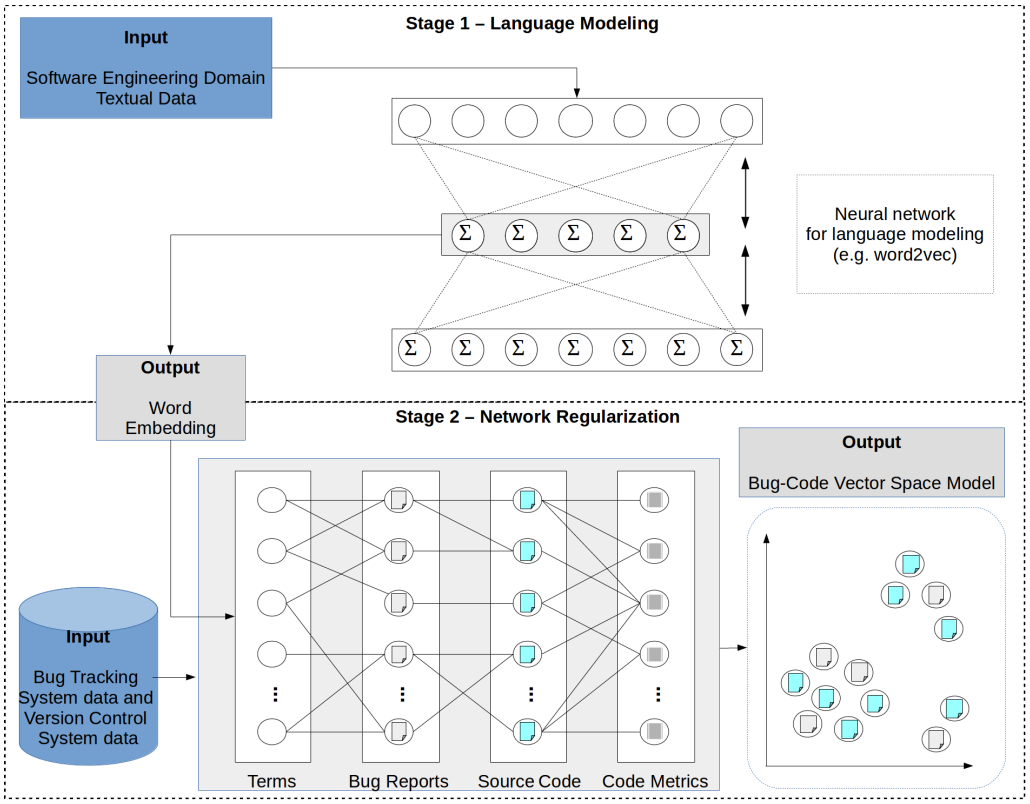


Figure 1. An overview of the BULNER.

Unlike existing methods that calculate the similarity between bug reports through keywords or language models, BULNER learns a new vector space model to directly compares bug reports and source code files. We call this new vector space of Bug-Code Vector Space Model. Moreover, our method allows the inclusion of domain information, such as code metrics, during the learning process of this vector space model.

The first stage of the BULNER method uses a neural network. The primary purpose of this stage is to learn word vector representations from a sizeable textual dataset of the software engineering domain. For example, the term 'abort' has the following correlated terms in the language model used in BULNER: 'interrupted,' 'terminate,' 'halt,' 'timed-out,' and 'exit.' The BULNER method uses the language model based on the skip-gram model trained over 15GB of textual data from Stack Overflow posts, as proposed in [Efstathiou et al. 2018]. The output of the skip-gram model is a representation called word embedding, where each term t contains a representation $\mathbf{Y}(t)$ in the d-dimensional vector space, i.e., $\mathbf{Y}(t) \in \mathbb{R}^d$.

The second stage of the BULNER method uses network regularization to perform the fine-tuning of the model obtained in the first stage. Given a bug reports dataset, we propose a heterogeneous network-based representation $N = (O, R, W)$, where O represents a set of objects o_i of the network, R represents a set of relations r_{o_i,o_j} between objects, and W represents a set of weights $w_{r_{o_i,o_j}}$ of the relations. We organize the set of network objects into four different types $O = \{O_B, O_T, O_S, O_M\}$, where O_B are objects that identify each bug report, O_T are terms t extracted from textual data of the bug reports, O_S are source files related to bug reports, and O_M are code metrics (discretized into intervals) computed from the source code files.

The general idea of network regularization is to obtain a new representation $\mathbf{F} \in \mathbb{R}^d$ in the d-dimensional vector space model, which satisfies two assumptions: (1) two objects o_i and o_j that share neighbors in the network must have similar vector representation, i.e., $\mathbf{F}(o_i) \sim \mathbf{F}(o_j)$, and (2) term-type objects in the network must have vector representation similar to the word embedding representation, i.e., $\mathbf{F}(t) \sim \mathbf{Y}(t)$.

It is inspired by the theoretical regularization framework of Ji et al. [Ji et al. 2010]. In Equation 1 we propose a regularization function for the BULNER method, where the goal is to minimize the function according to a representation model $\mathbf{F}$ for all objects of the network, given a word embedding $\mathbf{Y}$.

$$Q(\mathbf{F}) = \sum_{O_i,O_j}^{\{O_B,O_T,O_S,O_M\}} \frac{1}{2} \sum_{o_i \in O_i} \sum_{o_j \in O_j} w_{r_{o_i,o_j}} \left|\mathbf{F}(o_i) - \mathbf{F}(o_j)\right|^2 + \lim_{\mu \to \infty} \mu \sum_{t \in O_T} \left|\mathbf{F}(t) - \mathbf{Y}(t)\right|^2 \quad (1)$$

The first term of the regularization function is responsible for the first assumption, in which related objects must have similar representations to minimize the distance $w_{r_{o_i,o_j}} \left|\mathbf{F}(o_i) - \mathbf{F}(o_j)\right|^2$. Regarding the second assumption, the proposed regularization function ensures that the d-dimensional representation of a term will remain the same as the word embedding representation, i.e., $\lim_{\mu \to \infty} \mu \sum_{t \in O_T} \left|\mathbf{F}(t) - \mathbf{Y}(t)\right|^2$.

In practical terms, we can minimize the regularization function of the Equation 1 by using label (information) propagation techniques. In this case, BULNER initializes the representation of the term type objects according to word embedding $\mathbf{Y}$, whereas the representation of the remaining objects of the network can be randomly initialized. In each iteration, BULNER propagates the information of the term objects (i.e., WEmb) to the objects of the bug report type. The information is then propagated from bug reports to source code files objects and then to objects representing code metrics. The information is propagated back and each object $o_i \in O$ adjusts its $\mathbf{F}(o_i)$ representation. This process continues until there are no more significant changes in the $\mathbf{F}$ representation or until it reaches a maximum number of iterations, i.e., until the convergence of the BULNER method in which $\mathbf{F}$ is the learned representation for bug-code vector space model.

After the regularization process, we can directly compute the similarity between a bug report object $o_b \in O_B$ and a source code type object $o_s \in O_S$ as defined in the cosine similarity of Equation 2.

$$\cos(o_b, o_s) = \frac{\mathbf{F}(o_b) \cdot \mathbf{F}(o_s)}{||\mathbf{F}(o_b)||\,||\mathbf{F}(o_s)||} \quad (2)$$

$$sim(r_{new}, r_{train}) = (1 - \alpha)BOW(r_{new}, r_{train}) + \alpha BULNER(r_{new}, r_{train}) \quad (3)$$

A new bug report can be represented in the bug-code space vector through the word embedding of its terms and thus obtain a representation $\mathbf{F}(o_b)$. Equation 3 defines a new similarity function for bug localization in BULNER, which is a linear combination of similarity between bug reports in BoW model and similarity in Bug-Code Vector Space model since new bug reports contain only textual information. While we can define the BoW function as the cosine similarity between bug report keywords, the BULNER function represents the cosine similarity between the bug report representation $\mathbf{F}(o_b)$ and the source code file $\mathbf{F}(o_s)$ (Equation 2). The α parameter is a combination factor that allows defining the weight of the bug-code vector space model in the new similarity function, which can be estimated empirically. We use this new similarity function to calculate the scores for source files potentially related to the new bug reports.

4. Experimental Evaluation

4.1. Dataset

To evaluate our approach, we obtained data from three open source projects: AspectJ, Birt, and Tomcat. We extracted bug report data associated with each project from the Bugzilla repository provided by Ye *et al.* [Ye et al. 2014], while we mined each project's repository (located in GitHub) to obtain code-related metrics. For each bug report, we checked out a before-fix version of the source code from Github. Then, we used Understand™[1] to calculate different code metrics (object-oriented, volume and complexity metrics).

4.2. Baselines

We consider two methods in the literature for experimental evaluation. The first uses only the BoW model and cosine similarity to retrieve similar bug reports and related source files [Zhou et al. 2012]. The second combines the BoW model and WEmb models [Ye et al. 2016].

4.3. Evaluation Metrics

We use Mean Average Precision (MAP) as an evaluation criterion. Equation 4 calculates the precision in identifying NP buggy files, given a maximum value of k recommendations. Equation 5 calculates the precision average, where NPI is the total number of positive instances. Equation 6 calculates the MAP, where M is the total of bug reports. We use MAP with $k = \{1, 5, 10\}$, presented as MAP@1, MAP@5 and MAP@10.

$$P(k) = \frac{NB}{k} \quad (4) \qquad AP = \sum_{i=1}^{N} \frac{P(i)}{NPI} \quad (5) \qquad MAP = \frac{1}{M} \sum_{j=1}^{M} AP(j) \quad (6)$$

5. Results and Discussion

5.1. RQ1: How effective is BULNER?

Table 1 shows the best method's performance (max MAP performance independent of combination factor) for the three datasets. The results suggest that BULNER is the best

[1] Understand™: `https://scitools.com/`.

method for the three datasets. BULNER achieves this result by combining BoW with WEmb and Network Regularization. It receives as input a heterogeneous network (N), and it treats each type of objects and links separately. Moreover, BULNER minimizes classification error when preserving consistency for each relation graph (R) by applying graph regularization [Ji et al. 2010]. A statistical analysis of the results (Student's t-Tests with 95% confidence) does not allow us to state that the BULNER method is significantly superior to other methods, mainly due to the few data sets used in the experimental evaluation.

Table 1. MAP Performance Comparison with the State-of-the-art Methods

	AspectJ			Tomcat			Birt		
Methods	MAP@1	MAP@5	MAP@10	MAP@1	MAP@5	MAP@10	MAP@1	MAP@5	MAP@10
BoW+Cosine	0.1185	0.1738	0.1879	0.2121	0.2908	0.3017	0.0900	0.1372	0.1477
Embedding	0.1185	0.1738	0.1811	0.2134	0.2908	0.3001	0.0928	0.1402	0.1504
Bulner	0.1390	0.1913	0.2059	0.2201	0.2952	0.3078	0.0968	0.1420	0.1525

5.2. RQ2: What is the contribution of each method?

We evaluate the contributions of each method by the combination factor (α). In Figure 2, when α=0, all methods have the performance equal to baseline (BoW+Cosine), but when we increment α, each method has different behavior. In general, BULNER has better performance for: AspectJ when $0.15 < \alpha < 0.3$; Birt when $\alpha = 0.1$ and Tomcat when $0.05 < \alpha < 0.1$. These variations between software project occur because each one has its context.

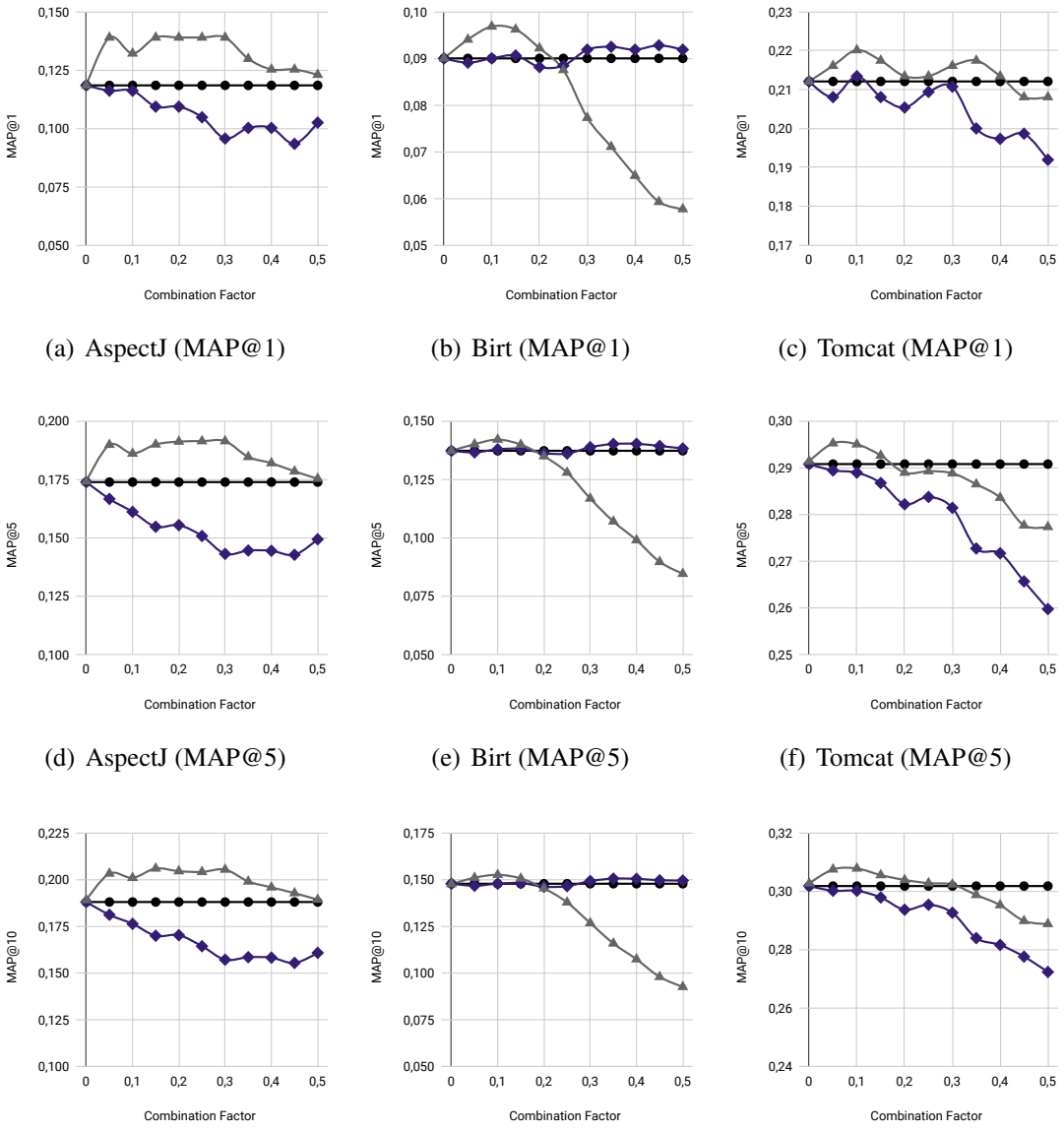
5.3. Threats to Validity

Regarding **Internal validity**, as only bug reports with "resolved" status were selected (because they represent bug reports that span the entire life cycle), the proposed model did not consider an immature bug report, such as those newly created by stakeholders. This restriction may have an impact on the performance of the proposed model. Regarding **External validity**, the results of this study can not be generalized to proprietary software since only Open Source software projects were analyzed. Finally, concerning **Conclusion validity**, we choose dataset provided by Ye *et al.* that is largely used to maximize the quality of the data collected.

6. Related Work

Zhou *et al*. propose an IR-based bug localization tool that using a revised Vector Space Model (rVSM) to represent software documents (bug report and source code) [Zhou et al. 2012]. In our study, we define a baseline using the Vector Space Model to representing data sources.

Wen *et al.* define an IR-based bug localization model that combining three models: the natural language model, code entity names model, and Boosting model [Wen et al. 2016]. In our work, we also evaluate the impact of combining models. Firstly, we only used the embedding model in BULNER, and then we incremental added value of regularization and combined it with the embedding model.



(a) AspectJ (MAP@1) (b) Birt (MAP@1) (c) Tomcat (MAP@1)

(d) AspectJ (MAP@5) (e) Birt (MAP@5) (f) Tomcat (MAP@5)

(g) AspectJ (MAP@10) (h) Birt (MAP@10) (i) Tomcat (MAP@10)

Figure 2. Methods' performance. ▲ BULNER; ● BoW+Cosine; ◆ Embedding.

7. Conclusion

We proposed a method BULNER, which locates bugs in terms of source files from bug reports and source code data. Our method is competitive with two other state-of-the-art methods. BULNER is very promising. In one case, it can recommend the 30% suspicious file within top 5 for one bug report. For future works, we intend to compare our work with different types of network embedding methods, for example, network embedding with side information or advanced information preserving network embedding. Additionally, we plan to extend our dataset with source code change genealogy and evaluate the impact on the performance of the methods. Our BULNER source code, as well as the datasets used, are publicly available at https://github.com/jacsonrbinf/bulner.

References

[Efstathiou et al. 2018] Efstathiou, V., Chatzilenas, C., and Spinellis, D. (2018). Word embeddings for the software engineering domain. In *Proceedings of the 15th International Conference on Mining Software Repositories*, MSR '18, pages 38–41, New York, NY, USA. ACM.

[Ji et al. 2010] Ji, M., Sun, Y., Danilevsky, M., Han, J., and Gao, J. (2010). Graph regularized transductive classification on heterogeneous information networks. In Balcázar, J. L., Bonchi, F., Gionis, A., and Sebag, M., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 570–586, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Mikolov et al. 2013] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

[Rahman and Roy 2018] Rahman, M. M. and Roy, C. K. (2018). Improving ir-based bug localization with context-aware query reformulation. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pages 621–632, New York, NY, USA. ACM.

[Tan et al. 2005] Tan, P.-N., Steinbach, M., and Kumar, V. (2005). *Introduction to Data Mining*. Addison-Wesley.

[Thung et al. 2014] Thung, F., Le, T.-D. B., Kochhar, P. S., and Lo, D. (2014). Buglocalizer: Integrated tool support for bug localization. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 767–770, New York, NY, USA. ACM.

[Wen et al. 2016] Wen, M., Wu, R., and Cheung, S. (2016). Locus: Locating bugs from software changes. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 262–273.

[Ye et al. 2014] Ye, X., Bunescu, R., and Liu, C. (2014). Learning to rank relevant files for bug reports using domain knowledge. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 689–699, New York, NY, USA. ACM.

[Ye et al. 2016] Ye, X., Shen, H., Ma, X., Bunescu, R., and Liu, C. (2016). From word embeddings to document similarities for improved information retrieval in software engineering. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 404–415.

[Zhou et al. 2012] Zhou, J., Zhang, H., and Lo, D. (2012). Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 14–24.

Um relato sobre a migração de uma plataforma de *offloading* para microsserviços

Adriano L. Cândido[1,4], Fernando A. M. Trinta[1], Paulo A. L. Rego [1], Lincoln S Rocha[1], Nabor C. Mendonça [2], Vinicius C. Garcia[3]

[1]Departamento de Ciências da Computação da Universidade Federal do Ceará (UFC)
Campus do Pici, s/n, CEP 60451-970 Fortaleza, Ceará, Brasil

[2]Centro de Ciências Tecnológicas - Universidade de Fortaleza (UNIFOR)
Av. Washington Soares, 1321, CEP 60811-905 Fortaleza, Ceará, Brasil

[3]Universidade Federal de Pernambuco (UFPE)
Av. Prof. Moraes Rego, 1235, CEP 50670-901 Recife, Pernambuco, Brasil

[4]Faculdade Vale do Salgado (FVS)
Rua Monsenhor Frota, 609, CEP 63430-000 Icó, Ceará, Brasil

Abstract. *Mobile devices are increasingly present in people's daily lives. However, despite the evolution of new generations of smartphones, the amount of information and the complexity of the procedures delegated to these devices still impose processing restrictions, mainly related to energy consumption. One solution that is being used for this problem is the technique known as offloading. In recent years, several offloading support platforms have been proposed. This paper focuses on one of these platforms, called CAOS. Despite successfully performing offloading tasks, CAOS still has issues such as low scalability. In this study, we describe the process of migrating from CAOS to a new architecture based on microservices, highlighting the decisions and practices that were adopted on this journey.*

Resumo. *Os dispositivos móveis estão cada vez mais presentes no dia a dia das pessoas. No entanto, apesar da evolução das novas gerações de smartphones, a quantidade de informações e a complexidade dos procedimentos delegados a esses dispositivos, ainda impõem restrições ao processamento, principalmente relacionado ao consumo de energético. Uma solução que vem sendo utilizada para esse problema é a técnica conhecida como offloading. Nos últimos anos, várias plataformas de suporte ao offloading foram propostas. Este trabalho tem foco em uma dessas plataformas, denominado CAOS. Apesar de realizar tarefas de offloading com êxito, o CAOS ainda apresenta problemas como baixa escalabilidade. Neste estudo, descrevemos o processo de migração do CAOS para uma nova arquitetura baseada em microsserviços, evidenciando as decisões e práticas que foram adotadas nessa jornada.*

1. Introdução

Os dispositivos móveis, especialmente *smartphones*, são objetos cada vez mais presentes no cotidiano das pessoas. A evolução desses dispositivos permitiu a criação de diversas aplicações para auxílio em tarefas relacionadas à mobilidade urbana, troca de mensagens,

entre outras. Apesar da substancial melhoria das novas gerações de dispositivos móveis, a quantidade de informações e a complexidade de procedimentos delegados a estes dispositivos ainda impõe restrições para processamento de certas tarefas, principalmente em relação ao consumo de energia. Isto é especialmente problemático para aplicações móveis sensíveis a contexto, uma classe particular de aplicações móveis que utiliza informações obtidas do ambiente de execução do usuário, para adaptar seu comportamento em prol de benefícios para a experiência do usuário. Uma abordagem promissora para mitigar tal problema é a *Mobile Cloud Computing*(MCC).

Segundo [Dinh et al. 2013], MCC tem por objetivo provisionar um conjunto de serviços equivalentes aos da nuvem, adaptados à capacidade de dispositivos com recursos restritos, de modo a trazer melhorias de desempenho das aplicações ou economia de energia nos dispositivos. Ao longo dos últimos anos, soluções baseadas no conceito de MCC foram propostas para auxiliar na descentralização do processamento de dados e operações, diminuindo o consumo energético dos dispositivos [Artail et al. 2015, Ferrari et al. 2016, Liao et al. 2015]. Boa parte destas soluções baseia-se no uso de um técnica conhecida como *offloading* [Kumar and Lu 2010], onde processamento e dados são migrados de nós com baixo poder de processamento ou armazenamento, para dispositivos com mais recursos computacionais. Neste contexto, este trabalho foca em uma destas propostas: o CAOS[Gomes et al. 2017].

O CAOS permite a implementação, na plataforma Android, de aplicações móveis sensíveis a contexto com suporte ao *offloading* de processamento e dados contextuais. Experimentos com esta plataforma mostraram ganhos tanto no desempenho quanto na economia de energia em diversos tipos de cenários. Por restrições de espaço, os detalhes sobre a plataforma CAOS não puderam ser apresentados. No entanto, no estudo [Gomes et al. 2017] a plataforma CAOS é apresentada com maiores detalhes. O projeto arquitetural da versão inicial do CAOS criou um conjunto de serviços para dar suporte às decisões sobre o *offloading* de dados e processamento. Porém, estes serviços foram implementados com forte interdependência entre eles, criando um sistema monolítico com problemas para manutenibilidade, configuração e implantação. Tais características também levaram a um problema no suporte à escalabilidade do CAOS, permitindo somente a escalabilidade vertical, ou seja, aumentando recursos do nó onde a plataforma servidora seja implantada.

Para tratar questões previamente mencionadas em softwares monolíticos, uma abordagem recente que tem recebido muita atenção é o uso de microsserviços, em que aplicações são separadas em serviços independentes que interagem entre si para realizar as funcionalidades do sistema [Taibi et al. 2017]. A migração de softwares monolíticos para microsserviços apresenta diversos benefícios como melhores formas de gerenciamento de disponibilidade, escalabilidade de partes específicas do software, capacidade de utilização de diferentes tecnologias, redução do *time-to-market* e melhor compreensão da base do código [Fan and Ma 2017]. Este artigo apresenta um relato do processo de migração da plataforma CAOS para uma arquitetura de microsserviços, destacando o processo e decisões arquitetônicas que foram tomadas neste contexto de migração.

2. Processo de Migração da Plataforma CAOS

Na Figura 1 são apresentados os quatro passos do processo de migração seguidos para construção da nova versão do CAOS baseado em microsserviços: **(i)** Concepção e refinamento da proposta; **(ii)** Definição dos microsserviços; **(iii)** Definição das tecnologias; e **(iv)** Desenvolvimento, testes, melhorias e concepção da nova arquitetura. Estas fases foram divididas em onze etapas, detalhadas a seguir.

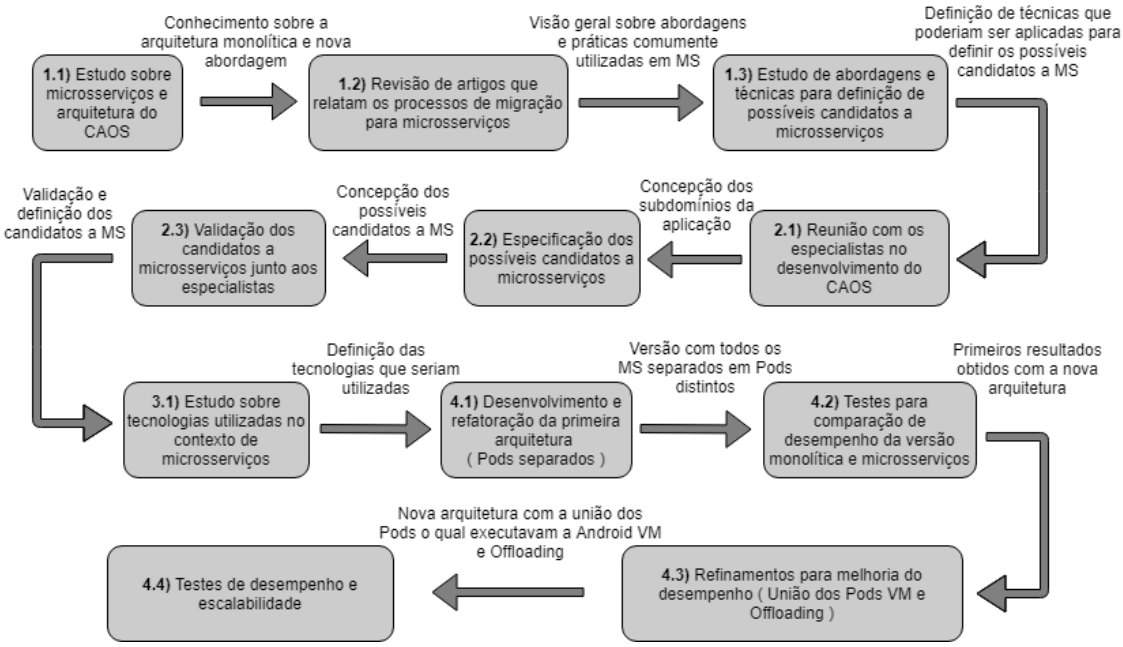


Figura 1. Processo utilizado na migração do CAOS

A etapa inicial do processo foi destinada à análise da versão inicial do CAOS. Nesta etapa, foram analisadas as funcionalidades, documentação, tecnologias, módulos e dependências que compõem a solução monolítica. Em conjunto, foram estudados conceitos e fundamentos da arquitetura microsserviços, buscando assimilar os benefícios que essa abordagem poderia proporcionar à solução CAOS. Com isso, foi possível obter uma visão inicial acerca dos benefícios que poderiam ser adquiridos com essa migração. Dentre eles, destacava-se uma escalabilidade mais eficiente, pois em momentos de alta demanda, seria possível escalar somente os recursos que estavam sendo sobrecarregados.

Na segunda etapa, foram revisados diversos artigos que relatavam a experiência de migração para a arquitetura de microsserviços, possibilitando uma visão geral das técnicas e práticas que poderiam ser utilizadas no contexto de migração proposto. Durante a revisão foi observado que os benefícios da arquitetura de microsserviços não são garantidos automaticamente, e só podem ser alcançados através de uma cuidadosa decomposição do software existente [Richardson 2018]. Nessa ocasião, foi percebido que para iniciar a migração para microsserviços, devem ser identificados os possíveis candidatos a microsserviços e que comumente é utilizada uma abordagem empírica para definição dos serviços da aplicação, ou seja, os serviços são definidos a partir do conhecimento dos *stakeholders* (desenvolvedores, projetistas e arquitetos) do software. No entanto, para realizar tais tarefas podem ser utilizadas técnicas, modelos e padrões de reenge-

nharia já existentes na engenharia de software [Balalaie et al. 2015], [Taibi et al. 2017], [Fan and Ma 2017], [Dragoni et al. 2017], [Balalaie et al. 2016].

A terceira etapa teve por objetivo a escolha de uma abordagem a ser usada na refatoração do CAOS. Foram identificadas três possíveis abordagens: *Empirical Definition* (ED), ***S**ervice-oriented **P**rocess for **Re**engineering **a**nd **D**evops (SPReaD)* e *Domain Driven Design* (DDD). A ED permite a definição dos microsserviços baseado no entendimento acerca da arquitetura do sistema, possibilitando identificar os serviços que poderiam funcionar de forma individual [Taibi et al. 2017]. O SPReaD possibilita a reengenharia de sistemas legados, integrando os aspectos de DevOps para o direcionamento e concepção de serviços [Justino 2018]. O DDD entende o sistema como um grande domínio que pode ser dividido em subdomínios. Cada subdomínio corresponde a uma parte diferente do negócio. Os serviços devem ser desenvolvidos de acordo com os subdomínios da aplicação[Richardson 2018].

Baseado nos artefatos gerados nas etapas anteriores, na quarta etapa foi realizada uma reunião com três dos especialistas que participaram do desenvolvimento da plataforma CAOS. Nessa ocasião, foi realizada uma entrevista semiestruturada para melhor compreensão sobre a forma como o CAOS foi arquitetado, projetado e construído. Procurou-se identificar as dependências que haviam entre os componentes e quais as funcionalidades que a plataforma ofertava, possibilitando com isso a elaboração de uma breve concepção dos subdomínios da aplicação.

Na quinta etapa, foram discutidas as metodologias encontradas na etapa 1.3. Dentre as estudadas, a que mais se adequou ao contexto de migração proposto foi a *Domain Driven Design*. A escolha dessa técnica é justificada em razão da pré-divisão dos componentes da solução. Ainda em sua versão monolítica já era possível perceber claramente os subdomínios que representavam a ferramenta como um todo. Nesse sentido, o uso do DDD se mostrou bastante eficaz, pois diversos aspectos que essa metodologia possibilita eram almejados na nova solução. Esta metodologia tem como premissa a divisão do sistema em pequenos módulos ou subdomínios de um domínio principal, obedecendo os seguintes princípios: Favorecimento do reuso; Alinhamento do código com o negócio; Mínimo de acoplamento e Independência da Tecnologia.

Na sexta etapa, foi realizada uma nova reunião com os mesmos especialistas citados na etapa 2.1. Esse encontro visou apresentar e validar os candidatos a microsserviços obtidos com o uso do DDD no contexto da plataforma CAOS. Os candidatos a microsserviços foram validados de acordo com as funcionalidades que eles exerciam e de acordo com a função de cada um no contexto do domínio principal.

Além disso, como já relatado, a plataforma CAOS também permite o *offloading* de dados contextuais. Entretanto, como o esforço técnico no processo de migração era alto, a presente migração limitou-se a tratar somente dos microsserviços que possuíam relação com o *offloading* de processamento, não abrangendo as funcionalidades de aquisição e gerenciamento contextual. Entre os componentes que compõe a parte de *offloading* de processamento, foram identificados seis possíveis candidatos:

- **(I) Microsserviço de Autenticação e Descoberta**, responsável por identificar os dispositivos que estão se conectando a plataforma, e fornecendo um meio de acesso aos demais serviços da solução;

- **(II) Microsserviço de Monitoramento**, responsável por monitorar e registrar diversas métricas dos dispositivos móveis, além de receber e registrar as informações dos processos de *offloading* local e remoto. Essas métricas são utilizadas posteriormente para construção da árvore de decisão que é utilizada para se decidir sobre a viabilidade ou não do *offloading* de um método;
- **(III) Microsserviço de Tomada de Decisão**, que é o serviço que constrói as árvores de decisão para cada método de aplicações suportadas pelo CAOS, levando em consideração as métricas dos *smartphones*, bem como os tempos de execução local e remota.
- **(IV) Microsserviço de *Offloading*** é responsável por receber os dados do método de um dispositivo e enviá-lo ao serviço de máquina virtual Android. Ao término do procedimento, o mesmo devolve o resultado ao dispositivo o qual solicitou o processamento;
- **(V) Microsserviço de Android Virtual Machine** é responsável por instanciar uma máquina virtual Android para realizar os processos de *offloading* requisitados pelo serviço de *Offloading*.
- **(VI) Microsserviço de Banco de Dados** encapsula um repositório responsável por registrar, fornecer e manter as informações geradas pelos demais microsserviços;

Na sétima etapa, foram estudadas e selecionadas ferramentas que pudessem auxiliar no desenvolvimento e implantação da nova arquitetura de microsserviços.Após a revisão realizada na etapa 1.2, cogitou-se a possibilidade de utilização das ferramentas `NetflixOSS`[1] (*Netflix Open Source Software Center*). Entretanto, percebeu-se que essas ferramentas criam muitas dependências, com impacto diretamente no desempenho, requisito chave para uma solução de *offloading*. Diante disso, analisaram-se diversas ferramentas que comumente eram utilizadas no desenvolvimento de microsserviços, a fim de identificar as que mais se adequavam ao cenário da plataforma CAOS e que pudessem fornecer as funcionalidades que são necessárias no ambiente de microsserviços. Entre as soluções encontradas, foram escolhidas o *Docker*[2] para conteinerização dos microsserviços e *Kubernetes*[3] para gerenciamento dos contêineres. Com isto, o projeto de refatoração do CAOS fez com que os microsserviços do lado servidor fossem transformados em imagens/contêineres do Docker, sendo gerenciados pelo Kubernetes.

Na oitava etapa, foi concebida a primeira versão do CAOS em microsserviços, batizada de CAOS MicroServices (CAOS MS). Nessa versão, a solução contava com todos os microsserviços separados e isolados em diferentes Pods no Kubernetes. Os módulos do lado cliente (dispositivo móvel) permaneceram inalterados, exceto por alguns ajustes na comunicação por conta de particularidades do Docker. Tal fato garante uma compatibilidade entre aplicações desenvolvidas para ambas versões da plataforma. Ressaltamos que somente os componentes do lado servidor passaram pelo processo de migração. Conforme pode ser observado na Figura 2, toda a comunicação entre o dispositivo móvel e os microsserviços ocorre de forma individual. Cada Pod representa um processo separado e individualizado dos microsserviços, que podem ser escalados e encerrados isoladamente. Um Pod é a menor unidade dentro de um cluster Kubernetes. Esse mecanismo permite a

[1]URL: `https://netflix.github.io`, último acesso em 10 de Julho de 2019.
[2]URL: `https://www.docker.com`, último acesso em 10 de Julho de 2019.
[3]URL: `https://kubernetes.io`, último acesso em 10 de Julho de 2019.

execução dos contêineres construídos no Docker[con 2018]. Vale ressaltar que em caso de falhas em algum microsserviço, os demais não serão afetados, e uma nova instancia do serviço onde ocorreu a falha será iniciada automaticamente pelo Kubernetes.

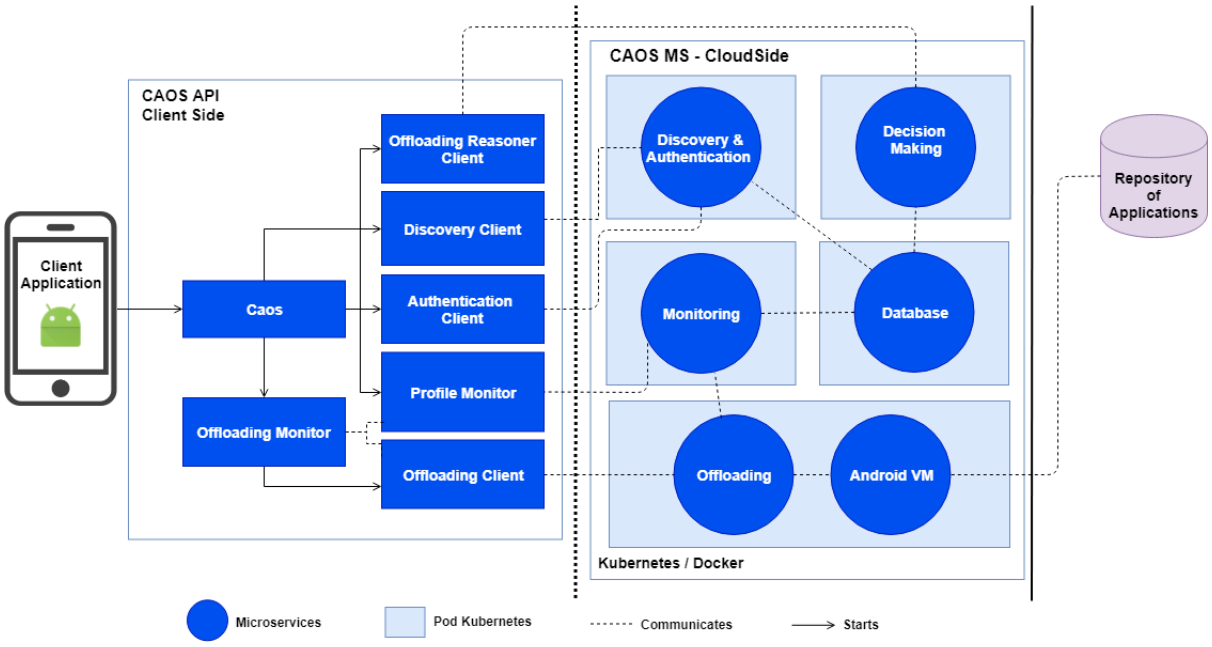


Figura 2. Arquitetura da plataforma CAOS MS.

Na nona etapa, foram realizados testes de desempenho para validação da nova versão concebida. Foram realizadas comparações de desempenho entre a versão monolítica e a utilizando microsserviços. Para a realização dos testes de desempenho, foi instalada a aplicação `BenchImage` em dois *smartphones*. Essa aplicação já havia sido utilizada por [Gomes et al. 2017] em testes na validação da versão monolítica do CAOS. A aplicação BenchImage permite a aplicação de diversos filtros em imagens de diferentes resoluções, os quais demandam um razoável nível de processamento por parte dos *smartphones*. Dois tipos de filtros são fornecidos pela aplicação: `Cartoonizer` e `RedTone`. Além disso, as imagens disponibilizadas na aplicação variam de 0,3 megapixels até 8,0 megapixels, de modo que quanto maior a resolução, mais recursos computacionais são exigidos para realizar o processamento do filtro.

Nos experimentos com a aplicação BenchImage, foi utilizado o filtro *RedTone* em imagens com diferentes resoluções (0,3MP, 2MP e 8MP). Os experimentos foram repetidos 30 (trinta) vezes com todos os parâmetros citados e com ambos *smartphones*. Após a finalização dos testes, foi possível perceber que o tempo médio para realização do *offloading* na arquitetura de microsserviços ficou muito superior quando comparado a versão monolítica. O tempo médio para realizar o processamento, quando comparado com a versão monolítica, aumentou em 8,32 milissegundo(ms), equivalente a um aumento de 134%. Apesar dos diversos benefícios que foram proporcionados com a nova arquitetura, o tempo de execução invalidava a proposta, pois é um dos fatores primordiais para adoção de uma estratégia de *offloading*.

Na décima etapa, foram realizados novos procedimentos de testes para identificar o causador da lentidão encontrada na nona etapa. Nestes testes, foi capturado o tempo de

execução em diferentes momentos do processo de *offloading*, com objetivo de se determinar os gargalos da execução remota dos métodos migrados. Então foi possível perceber que a separação dos microsserviços *Android VM* e *Offloading* em diferentes Pods no Kubernetes aumentava consideravelmente o tempo da execução do processo, pois após o microsserviço *Offloading* receber os argumentos da aplicação, era necessário acessar o serviço de descoberta do Kubernetes e transferir os dados recebidos para o Pod o qual possuía o serviço de *Android VM*. Esse mesmo processo era repetido após o término da execução do processo do *offloading*, praticamente duplicando o tempo necessário para devolver o resultado ao dispositivo que havia requisitado o processamento. Diante desse fato, foi realizada a união dos microsserviços *Android VM* e *Offloading* em um único Pod. Essa alteração impossibilitou o escalonamento individual desses microsserviços. No entanto, ambos continuam com processos únicos e independentes (i.e., as alterações realizadas em qualquer um deles não impactam no funcionamento, manutenção e evolução do outro).

Na última etapa, após a união dos microsserviços *Android VM* e *Offloading*, os mesmos experimentos de desempenho realizados na etapa 4.2 foram repetidos. Com os resultados adquiridos foi possível perceber que o tempo para realização do *offloading* se manteve muito próximo em ambas arquiteturas e que em alguns casos, o tempo do CAOS MS foi levemente menor. Em seguida, foram realizados testes de escalabilidade com 20 (vinte) dispositivos reais, a fim de verificar a eficácia em um ambiente com um maior número de requisições. O teste de escalabilidade mostrou ganhos significativos em relação a antiga arquitetura, pois durante o teste uma maior quantidade de dispositivos conseguiu executar as tarefas de *offloading* com êxito e em menor tempo, comprovando a eficácia na escalabilidade da nova arquitetura.

3. Conclusões

Este artigo apresentou um relato da experiência sobre a migração da plataforma de *offloading* monolítica denominada CAOS, para uma arquitetura microsserviços. Com isso, pôde-se perceber que a migração para arquiteturas de microsserviços impõe diversos desafios ao time de desenvolvimento, pois exige alto conhecimento acerca do código fonte, comunicação entre componentes e arquitetura do software proposto, além de bom domínio em ferramentas e tecnologias que auxiliam na construção de softwares distribuídos. No entanto, com base no que foi apresentado, a nova arquitetura trouxe maior flexibilidade em diversos aspectos da solução, como: implantação, manutenção, escalabilidade e evolução, possibilitando que tais funções fossem realizadas em microsserviços isolados.

Destacamos que o processo apresentado no estudo visou evidenciar os passos que foram adotados para realizar a migração, não foi visado a criação de um modelo de processo para possíveis migrações de arquiteturas monolíticas para microsserviços. No entanto, as práticas adotadas podem servir de base para outras migrações que visam desempenho, como unir os microsserviços que necessitam de grande velocidade de comunicação em um único Pod.

Como trabalhos futuros, pretende-se conduzir novos experimentos relacionados aos aspectos arquiteturais da versão com microsserviços, evidenciando as melhorias que herdadas pelas funcionalidades presentes na solução. Além disso, serão refatorados os

módulos de *offloading* de dados presentes na versão monolítica, para atuar na nova arquitetura em microsserviços.

Referências

(2018). Concrete — kubernetes. https://www.concrete.com.br/2018/02/22/tudo-o-que-voce-precisa-saber-sobre-kubernetes/. (Acessado em 13/07/2019).

Artail, A., Frenn, K., Safa, H., and Artail, H. (2015). A framework of mobile cloudlet centers based on the use of mobile devices as cloudlets. In *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on*, pages 777–784. IEEE.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2015). Migrating to cloud-native architectures using microservices: an experience report. In *European Conference on Service-Oriented and Cloud Computing*, pages 201–215. Springer.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2016). Microservices architecture enables devops: migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52.

Dinh, H. T., Lee, C., Niyato, D., and Wang, P. (2013). A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611.

Dragoni, N., Lanese, I., Larsen, S. T., Mazzara, M., Mustafin, R., and Safina, L. (2017). Microservices: How to make your application scale. *arXiv preprint arXiv:1702.07149*.

Fan, C.-Y. and Ma, S.-P. (2017). Migrating monolithic mobile application to microservice architecture: An experiment report. In *AI & Mobile Services (AIMS), 2017 IEEE International Conference on*, pages 109–112. IEEE.

Ferrari, A., Giordano, S., and Puccinelli, D. (2016). Reducing your local footprint with anyrun computing. *Computer Communications*, 81:1–11.

Gomes, F. A., Rego, P. A., Rocha, L., de Souza, J. N., and Trinta, F. (2017). Caos: A context acquisition and offloading system. In *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 957–966. IEEE.

Justino, Y. d. L. (2018). Do monolito aos microsserviços: um relato de migração de sistemas legados da secretaria de estado da tributação do rio grande do norte. Master's thesis, Brasil.

Kumar, K. and Lu, Y.-H. (2010). Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(4):51–56.

Liao, L., Qiu, M., and Leung, V. C. (2015). Software defined mobile cloudlet. *Mobile Networks and Applications*, 20(3):337–347.

Richardson, C. (2018). Pattern: Decompose by subdomain. https://microservices.io/patterns/decomposition/decompose-by-subdomain.html. (Acesso em: 11/04/2018).

Taibi, D., Lenarduzzi, V., Pahl, C., and Janes, A. (2017). Microservices in agile software development: a workshop-based study into issues, advantages, and disadvantages. In *Proceedings of the XP2017 Scientific Workshops*, page 23. ACM.

Uma análise da relação entre code smells e dívida técnica auto-admitida

Felipe Gustavo de S. Gomes[1], **Thiago Souto Mendes**[2], **Rodrigo O. Spínola**[3], **Manoel Mendonça**[1], **Mário Farias**[4]

[1] Universidade Federal da Bahia, [2]Instituto Federal da Bahia, [3]Universidade Salvador, [4]Instituto Federal de Sergipe

{felipegustavo,mgmendonca}@dcc.ufba.br, thiagosouto@ifba.edu.br, rodrigo.spinola@unifacs.br, mario.andre@ifs.edu.br

Resumo. *Code smells indicam possíveis problemas na implementação de um sistema que podem levar à necessidade de refatoração do seu código. Eles podem ser detectados automaticamente e são considerados indicadores de presença de Dívida Técnica (DT). Contudo, estudos indicam que considerar apenas code smells na tarefa de detecção da presença de itens de dívida é insuficiente. É necessário utilizar estratégias de detecção complementares, como a utilização de informações extraídas a partir de comentários de código. Porém, ainda são poucos os estudos sobre a relação entre code smells e DT auto-admitida. Este trabalho analisa três projetos open source para investigar a relação existente, em termos de sobreposição e complementariedade de itens de DT identificados, utilizando detecção via code smells e DT auto-admitida. Os resultados indicam que as informações de comentários podem complementar as informações de code smells e apoiar os desenvolvedores a identificar DT.*

Abstract. *Code smells are an indication that there may be implementation problems in a system and may lead to the need to refactor its code. They can be detected automatically and are considered indicators of the presence of Technical Debt (TD). However, studies indicate that considering only code smells in the task of detecting the presence of debt items is insufficient. Complementary detection strategies, such as using information extracted from code comments, must be used. Although, there are still few studies about the relationship between code smells and self-admitted TD. This paper analyzes three open source projects to investigate the existing relationship, in terms of overlap and complementarity of identified TD items, using detection by code smells and self-admitted TD. Results indicate that comments information can complement code smells information and support developers to identify TD.*

1. Introdução

Code smells, surgem de escolhas de implementação de um sistema que não obedecem a princípios amplamente aceitos de um bom projeto de software [Zazworka et al. 2013]. Eles são uma indicação inicial de que pode haver sérios problemas na implementação do sistema [Fowler 1999]. Esses problemas podem dificultar a evolução do software e levar à necessidade refatoração do seu código [Fontana et al. 2012].

Por sua vez, o conceito de Dívida Técnica (DT) tem ajudado profissionais e pesquisadores a discutirem problemas associados à evolução do software [Seaman and Guo 2011]. Ele contextualiza o problema de tarefas de desenvolvimento de software pendentes como um tipo de dívida que traz um benefício de curto prazo para o projeto, mas que pode precisar ser paga com juros mais tarde no processo de desenvolvimento [McConnell 2008]. Por exemplo, a evolução de uma classe mal projetada tende a ser mais dispendiosa do que uma implementada considerando boas práticas de projeto.

A identificação de DT é uma etapa essencial no processo de seu gerenciamento. Algumas pesquisas têm focado na identificação automática de DT. Isto traz vantagens em termos produtividade e facilitaria a adoção do gerenciamento de DT. Todavia, a maioria destas técnicas é baseada na detecção de *code smells*. O problema desse tipo de abordagem é que ela é capaz de detectar apenas alguns tipos de dívidas, notadamente dívidas de código e de projeto [Alves et al. 2016].

Por outro lado, programadores explicitam a existência de soluções incompletas e temporárias que exigem retrabalho através de comentários em código. Neste caso, tem-se o conceito de DT Auto-Admitida (DTAA) [Potdar and Shihab 2014]. As DTAAs tratam de um espectro mais amplo de dívidas. Elas podem ser mais assertivas quanto à variedade de itens identificados uma vez que permitem obter informação de contexto explicitada por desenvolvedores [Farias et al. 2015]. Estratégias baseadas em análise de comentários podem ser utilizadas para a identificação automática de itens de DTAA. Um exemplo é o Contextualized Vocabulary Model for identifying TD on code comments (CVM-TD), que trata-se de um vocabulário contextualizado com expressões e termos proposto por Farias *et al.* [2015] para apoiar a identificação de DTAA através da análise de comentários de código fonte.

Apesar de serem os principais indicadores para identificação de DT citados na literatura, os *code smells* podem não trazer todas as informações necessárias para identificar os diferentes tipos de DT que podem ocorrer no software, como dívida de requisitos e defeito [da Silva Maldonado et al. 2017]. Em teoria, uma abordagem combinada das duas técnicas, análise de *code smells* e análise de comentários, poderia melhorar a identificação de itens de DT e ampliar o espectro de itens de DT identificados automaticamente.

Este artigo investiga a relação entre a detecção automática de *code smells* e DTAA extraída utilizando CVM-TD, a fim de avaliar a sobreposição e complementariedade das duas técnicas. Para atingir o objetivo, foi realizada uma análise quantitativa e qualitativa de três projetos *open source*. Os resultados do estudo indicaram que as informações de comentários de código podem complementar as informações produzidas pelos *code smells* e apoiar equipes de desenvolvimento a identificar DT. A análise quantitativa mostrou que instâncias de DTAA tendem a se relacionar com os *code smells Duplicated Code*, *God Class*, *Long Method* e *Complex Method*. Por outro lado, análise qualitativa mostrou que comentários possuem informações com sugestões e percepções do desenvolvedor que poderá ajudá-lo a decidir se pagará aquela dívida naquele momento e como resolvê-la.

O restante do artigo está organizado da seguinte forma. A Seção 2 discute alguns trabalhos relacionados. A Seção 3 descreve o contexto e o planejamento do estudo. A Seção 4 apresenta e discute os resultados quantitativos obtidos. A Seção 5 apresenta alguns resultados da análise qualitativa que foi realizada. A Seção 6 discute as ameaças à

validade do estudo. Por fim, a Seção 7 apresenta as considerações finais deste trabalho e direcionamentos para trabalhos futuros.

2. Trabalhos Relacionados

Em seu trabalho, Wehabi *et al.* [2016] examinaram a relação entre DTAA e a qualidade do código através da investigação se: **(i)** arquivos com DTAA têm mais defeitos que arquivos sem dívida; **(ii)** modificações nos arquivos com DTAA introduzem defeitos; e **(iii)** se modificações relacionadas a DTAA tendem a ser mais difíceis. Nesse estudo foram analisados cinco projetos *open source*. Os pesquisadores constataram que **(i)** não existe uma clara tendência em relação a defeitos e DTAA; **(ii)** modificações relacionadas a arquivos com DTAA induzem menos defeitos futuros do que modificações em arquivos sem DT; e que **(iii)** arquivos com DTAA são mais difíceis de serem modificados. O estudo também indicou que apesar da DT ter impactos negativos, os seus efeitos não são relacionados com defeitos, mas sim com a redução da manutenibilidade do sistema.

Mensah *et al.* [2018] introduziram um esquema de priorização composto principalmente por identificação, exame e estimativa de esforço de retrabalho de atividades priorizadas a fim de auxiliar na tomada de decisão antes do lançamento de uma versão do software. Usando o esquema proposto, os pesquisadores realizaram uma análise exploratória em quatro softwares *open source* para investigar como a DTAA pode ser minimizada. Foram identificadas quatro causas principais de DTAA: *code smells*, tarefas complicadas e complexas, testes inadequados e performance de código inesperada. Os resultados mostraram que, entre todos os tipos de DT, as dívidas de projeto eram mais propensas a *bugs*.

3. Planejamento do Estudo

A fim de avaliar a relação entre *code smells* e DTAA, foram analisados três projetos *open source* desenvolvidos em Java. Os projetos considerados foram ArgoUML[1] (versão 0.34), JFreeChart[2] (versão 1.0.19) e Apache Ant[3] (versão 1.10.2). O ArgoUML e o JFreeChart foram escolhidos, pois são conhecidos na comunidade científica na realização de estudos sobre comentários de código fonte [Farias et al. 2015] [da Silva Maldonado et al. 2017]. Por outro lado, o Apache Ant foi escolhido por ter uma grande quantidade de comentários de código. Na Tabela 1, é apresentada a quantidade de arquivos analisados e a quantidade total de linhas de comentários e de código. Os dados utilizados nas análises encontram-se disponíveis no endereço `http://bit.ly/dtaastudy`.

Inicialmente, foram detectados os *code smells* utilizando a ferramenta RepositoryMiner (RM) [Gomes et al. 2017]. O RM é uma ferramenta extensível para a mineração de repositórios de software para suportar a identificação automática de DT. Os *code smells* escolhidos para este estudo foram: *Brain Class*, *Brain Method*, *Complex Method*, *Data Class*, *Feature Envy*, *God Class*, *Long Method* e *Duplicated Code* [Fowler 1999] [Kerievsky 2005] [Lanza and Marinescu 2006]. Os *code smells* escolhidos estão entre os mais comuns e podem ser detectados pelo RM.

[1]https://github.com/stcarrez/argouml
[2]https://github.com/jfree/jfreechart
[3]https://github.com/apache/ant/

Logo após, foram extraídos os comentários de código fonte com algum indício de DT. A análise de comentários foi realizada utilizando o RM através do seu módulo de integração com o eXcomment [Farias et al. 2015]. O eXcomment é uma ferramenta capaz de identificar e classificar a DT através da análise de comentários de código. A ferramenta é baseada em mineração de texto e utiliza um vocabulário contextualizado, o CVM-TD, composto por padrões [Farias et al. 2015]. Esses padrões são um conjunto de palavras, códigos de DT e termos de engenharia de software que permitem a identificação daquele comentário como um item de DT. Em um comentário pode haver um ou mais padrões. Além disso, eXcomment também classifica os padrões entre os seguintes tipos de DT: código, projeto, arquitetura, construção, defeito, documentação, pessoas, requisitos, teste e desconhecido (ou seja, são considerados como uma DT, mas não foi possível determinar o tipo exato).

Apesar das análises realizadas através do RM nos projetos de software terem considerado diferentes tipos de granularidade em relação aos elementos do software, como classes e métodos, neste estudo essa granularidade não foi considerada na análise dos resultados. Para isso, as informações de *code smells* e comentários de código fonte referentes a classes e métodos foram agrupadas em função dos arquivos nos quais elas foram identificadas. Dessa forma, foi avaliada a existência de sobreposição entre arquivos reportados por ambas as estratégias como contendo itens de dívida e analisado o tipo de informação disponibilizado por cada técnica a respeito daquele arquivo.

Nas seções a seguir, são apresentados os resultados e discussões das análises quantitativa e qualitativa dos projetos selecionados. Na análise quantitativa, são apresentados os comparativos das informações dos arquivos dos três projetos analisados com as relações existentes entre *code smells* e comentários de código. Na análise qualitativa, devido a falta de espaço, são apresentados os resultados parciais da verificação dos comentários do projeto ArgoUML, que foi selecionado por ser o maior projeto entre os três.

4. Análise Quantitativa

Foi realizada uma análise quantitativa através do estudo das combinações entre *code smells* e comentários com o objetivo de verificar possíveis relações. Foi considerado como sendo uma relação se um arquivo apresenta simultaneamente comentários selecionados pelo eXcomment e *code smells* indicando a presença de dívida.

Na Tabela 1, é apresentada a quantidade de arquivos com *code smells*, arquivos com comentários de DT e de relações encontradas. Ao analisar os dados é possível verificar que existe uma alta taxa de relação nos projetos Apache Ant (76% (quantidade de relações / quantidade de arquivos com *code smells*)), ArgoUML (73%) e JFreeChart (59%), com uma média de 68% do total possível de relações, mostrando que *code smells* e DTAA tendem a caminhar juntos.

Com o objetivo de analisar quais tipos de DT têm relações mais fortes, foram gerados gráficos de bolha (ver Figura 1) com a combinação das informações sobre os tipos de *code smells* e os tipos de dívidas identificados pelos comentários. Analisando os gráficos, é possível notar um padrão entre os três projetos. Notou-se que os *code smells* tendem a se relacionar fortemente com quase todos os tipos de DT, exceto com dívidas de teste, pessoas e documentação. Nota-se também que os *code smells*: *Duplicated Code*, *God Class*, *Long Method* e *Complex Method* tendem a ser maiores indicadores de DT

Tabela 1: Dados dos Projetos Analisados.

Projeto	# Arquivos	LOC Comentários	LOC Código	# Arquivos Comentários	# Arquivos Code Smells	# Relações
Apache Ant	1220	103393	137732	622	276	209
ArgoUML	1870	148051	173855	995	451	327
JFreeChart	1017	144851	144342	475	364	215

que os demais, principalmente para dívida de projeto, defeito, código e desconhecido. Além disso, é possível observar que os comentários são capazes de complementar as informações dos *code smells*, mostrando relações com outros tipos de DT que vão além da qualidade do código fonte, como documentação, requisitos, construção e defeitos.

É importante também destacar a elevada quantidade de relações de dívidas com o *code smell God Class* nos três projetos. O *God Class* refere-se a classes que tendem a centralizar a inteligência e a maioria do trabalho do sistema [Lanza and Marinescu 2006]. *God class* também é um dos principais indicadores para identificar DT de projeto [Alves et al. 2016]. Nesse cenário, a combinação da identificação do *code smell God Class* e de comentários com indicando DTAA pode aumentar as chances de identificação e priorização do pagamento das dívidas do projeto.

O projeto JFreeChart não apresenta relações de DTAA com *Data Class*, isso ocorreu pois existiam poucas ocorrências do *code smell* (apenas quatro em todo o projeto). Também, destaca-se o fato do projeto Apache Ant ter quantidade de relações próximas aos dos outros projetos mesmo tendo uma quantidade de linhas de comentários consideravelmente menor. Isso pode indicar que os desenvolvedores do projeto Apache Ant costumam reportar com mais frequência problemas através de comentários no código fonte.

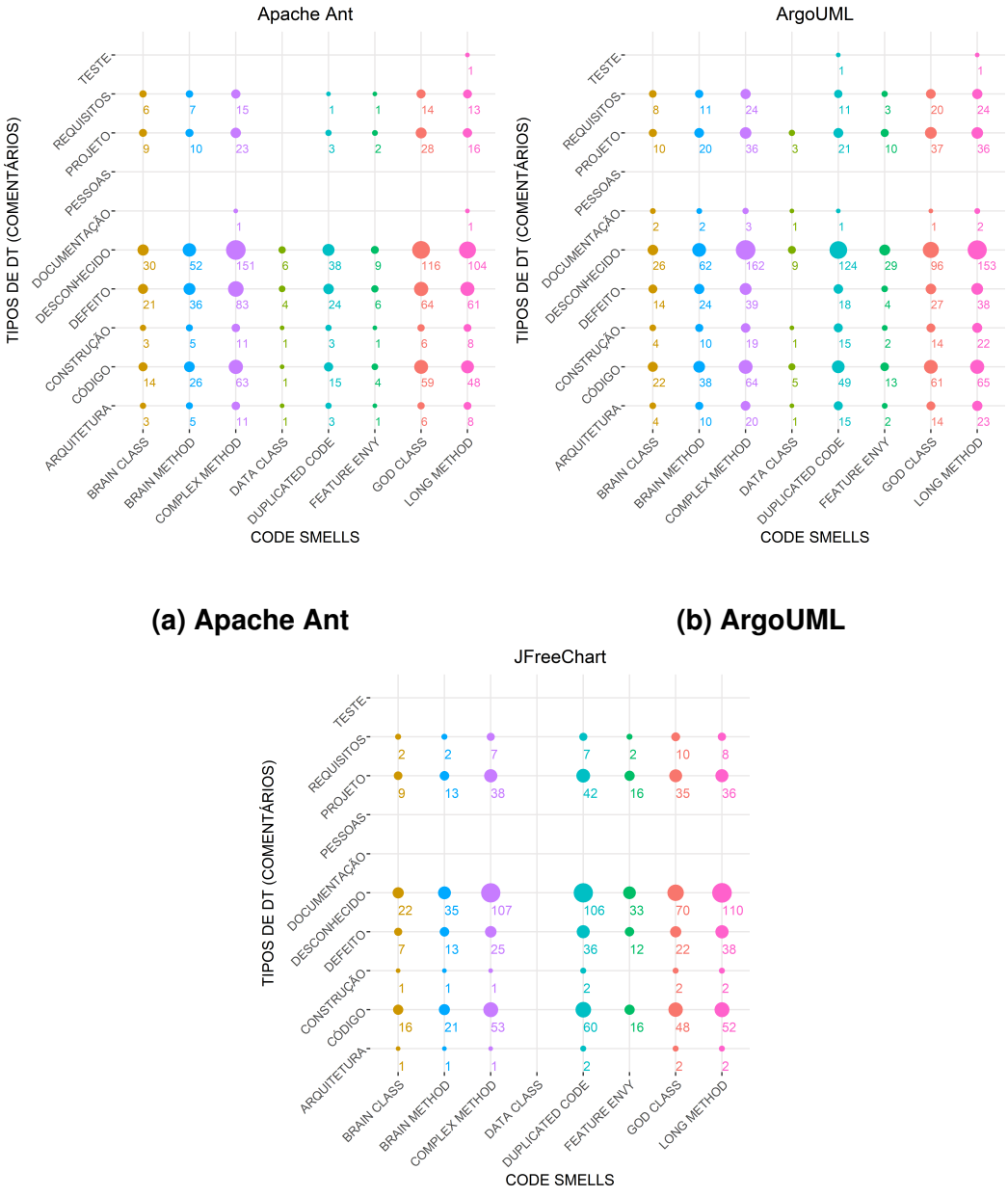
5. Análise Qualitativa

Foi realizada uma análise qualitativa através da leitura dos comentários de código fonte do projeto ArgoUML. Foram analisados todos os arquivos que apresentaram relações entre *code smells* e DTTA. Nas subseções a seguir será apresentada uma breve análise das relações dos comentários com os *code smells Duplicated Code* e *Long Method*.

5.1. Identificação de *Long Method*

O *Long Method* indica que um método, função ou procedimento está crescendo muito. A métrica utilizada para a detecção deste *code smell* é referente ao número de linhas de código [Fowler 1999]. Foram encontrados vários casos onde existem comentários apontando uma sugestão de melhorias em métodos classificados com *Long Method*, como exemplo, o **Comentário #01** que está localizado em um método que apresenta um *Long Method*. Neste caso, o desenvolvedor deixa registrada uma sugestão de melhoria na implementação do método, ele sugere a quebra de método grande e complexo em vários outros métodos menores e mais simples. Neste método, que contém 315 linhas de código, foram encontrados os seguintes *code smells*: *Brain Method*, *Complex Method*, *Long Method* e *Duplicated Code*, *code smells*, que se alinham com o problema apresentado no comentário.

Comentário #01 - AbstractMessageNotationUml: `TODO: This method is too complex, lets break it up.`



(a) Apache Ant **(b) ArgoUML**

(c) JFreeChart

Figura 1: Relação de tipos de DT e *code smells* dos projetos analisados.

5.2. Identificação de *Duplicated Code*

A ocorrência de *Duplicated Code* pode gerar um aumento no tamanho do código e dificultar o processo de sua manutenção [Lanza and Marinescu 2006]. Por exemplo, se for necessário alterar vários trechos de forma semelhante, fatalmente um trecho ou outro será esquecido. No estudo foram encontrados casos de comentários indicando a ocorrência de código duplicado tanto em partes que continham o *code smell Duplicated Code* como em partes que não possuíam. Em alguns desses casos, o desenvolvedor informa que existe uma duplicação do código e que ela deve ser corrigida, por exemplo: **Comentários #02** e **#03**. No **Comentário #02**, que referencia uma ocorrência do *code smell Duplicated*

Code, é descrita uma situação de um trecho de código que foi copiado de outra classe devido a um erro em uma dependência e que será removido quando o erro for resolvido. No segundo exemplo, o **Comentário #03**, que não tem relação com *code smell* algum, o desenvolvedor expressa a duplicação de código entre dois métodos. No caso do **Comentário #03**, não foi identificado o *Duplicated Code*, pois o trecho duplicado era muito pequeno, mas mesmo assim foi considerado um problema pelos desenvolvedores, mostrando que as informações de comentários podem complementar a de análise de smells.

> **Comentário #2 - FigComposite:** `TODO: All code below here is duplicated in FigBaseNode. The reason is the GEF defect - http://gef.tigris.org/issues/show_bug.cgi?id=358 Once we have taken a release of GEF with that fix we can remove this code.`
>
> **Comentário #03 - StateMachinesHelperMDRImpl:** `TODO: getAllPossibleSubvertices and getAllSubStates are duplicates - tfm?`

6. Ameaças à Validade

Foram identificadas algumas ameaças à validade, que são descritas seguindo a classificação usual de ameaças internas, externas, construção e conclusão:

- **Ameaças internas**: O estudo foi baseado em conjuntos de dados extraídos de três projetos consolidados. A principal ameaça é a ocorrência de falsos positivos na detecção de comentários e *code smells*, uma vez que a ocorrência excessiva deles pode comprometer os dados e distorcer as análises. Para melhorar a precisão na detecção dos *code smells* foram utilizados os valores de limiares apresentados por Lanza e Marinescu [Lanza and Marinescu 2006], amplamente adotados na comunidade científica. Portanto, as ameaças à validade interna foram reduzidas, apesar de não ser possível garantir a eliminação dos falsos positivos gerados via análises automáticas.
- **Ameaças externas**: Os resultados encontrados nas análises não podem ser generalizados para outros sistemas. É necessário realizar estudos adicionais com um maior número de projetos de software considerando diferentes contextos e dimensões para se obter maior confiança na validade externa dos resultados.
- **Ameaças de construção**: A análise foi realizada sem considerar os diferentes níveis de granularidade (classes e métodos), podendo resultar na consideração de relações inexistentes entre ocorrências de DTAA e *code smells* nas análises. Contudo, essa análise mais detalhada não é uma tarefa trivial e este é um estudo inicial.
- **Ameaças de conclusão**: Apesar do estudo ter considerado três projetos, o conjunto de dados é bastante amplo para análise devido ao número identificado de itens de DT. Além disso, os softwares ArgoUML e JFreeChart foram utilizados em vários trabalhos sobre identificação de DT [Farias et al. 2015] [da Silva Maldonado et al. 2017]. Todavia, a validade de conclusão será melhor sustentada após a replicação do estudo em outros projetos, para o qual os dados e informações coletadas permitirão um aprimoramento do estudo atual.

7. Considerações Finais

Neste trabalho, foi conduzido um estudo para avaliar a relação entre *code smells* e DTAA através da avaliação de três projetos *open source*. Os resultados do estudo corroboram

com a noção geral da literatura dos impactos dos *code smells* na qualidade do código fonte. Além disso, também são apresentadas relações ainda pouco exploradas na literatura, como os impactos dos *code smells* na construção, requisitos, defeito e documentação, indicando que *code smells* podem ser indicadores de problemas tanto na avaliação da qualidade interna como externa do projeto. Os resultados também mostram que em alguns casos a utilização de comentários de código fonte pode ajudar na complementação de informações que não poderiam ser obtidas apenas com o uso de *code smells*. Como trabalhos futuros, pretende-se estender o estudo e realizar análises em outros projetos e com a participação de engenheiros de software para triangulação de dados.

Referências

Alves, Nicolli, S., Mendes, T. S., Mendonça, M. G., Spínola, R. O., Shull, F., and Seaman, C. (2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, 40:100–121.

da Silva Maldonado, E., Shihab, E., and Tsantalis, N. (2017). Using natural language processing to automatically detect self-admitted technical debt. *IEEE Transactions on Software Engineering*, 43(11):1044–1062.

Farias, M. A., de Mendonça Neto, M. G., da Silva, A. B., and Spínola, R. O. (2015). A contextualized vocabulary model for identifying technical debt on code comments. In *7th MTD*, pages 25–32. IEEE.

Fontana, F. A., Ferme, V., and Spinelli, S. (2012). Investigating the impact of code smells debt on quality code evaluation. In *Proceedings of the MTD*, MTD '12, pages 15–22, Piscataway, NJ, USA. IEEE Press.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, USA.

Gomes, F., Mendes, T., Carvalho, L., Spínola, R., Novais, R., and Mendonça, M. (2017). Repositoryminer – uma ferramenta extensível de mineração de repositórios de software para identificação automática de dívida técnica. *CBSoft - Salão de Ferramentas.*

Kerievsky, J. (2005). *Refactoring to patterns*. Pearson Deutschland GmbH.

Lanza, M. and Marinescu, R., editors (2006). *Object-Oriented Metrics in Practice*. Springer-Verlag Berlin Heidelberg, New York.

McConnell, S. (2008). Productivity variations among software developers and teams: The origin of 10x. *10x Software Development*.

Potdar, A. and Shihab, E. (2014). An exploratory study on self-admitted technical debt. In *ICSME, 2014 IEEE International Conference on*, pages 91–100.

Seaman, C. and Guo, Y. (2011). Advances in computers. In Zelkowitz, M. V., editor, *Chapter 2 - Measuring and Monitoring Technical Debt*, volume 82 of *Advances in Computers*, pages 25 – 46. Elsevier.

Zazworka, N., Spínola, R. O., Vetro', A., Shull, F., and Seaman, C. (2013). A case study on effectively identifying technical debt. In *Proceedings of the 17th EASE*, EASE '13, pages 42–47, New York. ACM.

Clustering Similarity Measures for Architecture Recovery of Evolving Software

Douglas E. U. Silva[1], **Roberto A. Bittencourt**[1], **Rodrigo T. Calumby**[1]

[1] UEFS – Universidade Estadual de Feira de Santana
Av. Transnordestina, s/n, Novo Horizonte
Feira de Santana – BA, Brasil – 44036-900

douglaseusilva@gmail.com, roberto@uefs.br, rtcalumby@uefs.br

Abstract. *Automated software architecture recovery of module views from source code is a challenging research issue. Different similarity measures are used to evaluate clustering algorithms in the software architecture recovery of module views. However, few studies seek to evaluate whether such measures accurately capture the similarities between two clusterings. This work presents an evaluation of six clustering similarity measures through the use of intrinsic quality and stability measures and the use of ground truth architectures proposed by developers. The results suggest that the MeCl metric is the most adequate to measure similarity in the context of comparison with ground truth models provided by developers. However, when the architectural models do not exist, the Purity metric shows the best results, as measured by the correlation with the intrinsic Silhouette coefficient.*

1. Introduction

Clustering algorithms are quasi-automatic techniques that seek to identify clusters of similar software entities from their features. In general, we may state that clustering software entities into modules, from a relation of similarity between them is a way to modularize a system. These modules formed by such clustering algorithms are usually called clusters.

Previous work by Wu et al. (2005) extracted monthly versions of some open source systems and applied different clustering algorithms to them, defining stability and authoritativeness metrics to evaluate the algorithms. Informally, stability indicates that when a system undergoes changes, generated clusters must reflect these changes at the architectural level. On the other hand, authoritativeness evaluates how resulting clusters resembles a clustering created by a software architect, i.e., an architectural model. Unfortunately, their work used architecture views based on file allocation in source code directories as the authority, and not a reference model of system modules explicitly defined by software architects. More recent work by Garcia et al. (2013) and Lutellier et al. (2015) evaluated several algorithms against five open source systems from the perspective of comparison with reference models and four different similarity metrics. Although those papers advance the process of evaluating clustering techniques for software architecture recovery, one needs to define the best clustering similarity metrics by taking into account appropriate clustering similarity criteria.

This work evaluates six similarity metrics, taking into account their authoritativeness and stability as well as their discriminatory power, and the comparison with intrinsic clustering metrics and with variability measures of the evolving software.

2. Background

Agglomerative hierarchical algorithms are a class of clustering algorithms that start with singleton clusters and, after $N - 1$ steps, all items are contained in a single cluster. Typically, rule of thumb cutoff points stop the clustering process before reaching the single cluster in order to generate meaningful clusterings.

2.1. Clustering Similarity Metrics

The work of Wu et al. (2005) proposed the evaluation of clustering algorithms from three criteria, from which we use two. *a)* **Authoritativeness.** Clusters generated by the algorithms should resemble some authority. Therefore, authoritativeness compares clusterings computed by an algorithm against a reference model clustering. *b)* **Stability.** Similar clusterings should be produced by similar versions of a software system. Therefore, stability assessment compares clusterings of consecutive versions of the target system.

Next, we present some clustering similarity metrics from the literature.

Precision-Recall. Defined by Anquetil and Lethbridge (1999) for the field of software architecture recovery, Precision-Recall, as a measure of similarity between two software clusterings A and B, is computed based on the comparison of entity pairs, and is defined as follows: *a)* Precision: Percentage of intra-cluster pairs in clustering A that are also intra-cluster pairs in clustering B. *b)* Recall: Percentage of intra-cluster pairs present in clustering B that are also intra-cluster pairs in clustering A.

To balance the precision and recall values, we use the $F1$ measure which is defined as the harmonic mean of precision and recall.

B-Cubed Precision-Recall. The *B-Cubed* version of Precision-Recall was created by associating precision and recall for each item in the clustering. The B-Cubed metric defines precision as the number of items in the same cluster belonging to its category, and recall as how many items in its category appear in the cluster.

As in the Precision-Recall metric, we also use the F1-measure to combine the precision and recall values computed by the B-Cubed metric. For simplicity, we will call the F1-measure of B-Cubed Precision-Recall as $B\text{-}Cubed\text{-}F1$.

Purity. Purity is a metric defined to quantify how much a cluster C_i is "pure", that is, how many items are correctly identified when comparing two clusterings A e B. This way, Purity can be used as a similarity metric to computer the similarity between two clusterings.

MoJo and MoJoSim. Tzerpos and Holt (1999) defined a dissimilarity metric between two architectural clusterings called *MoJo*. This metric is based on the number of operations to transform one clustering into another. ***Move*** entails removing one entity from one cluster and allocating it to another, and ***Join*** implies joining two existing clusters, decreasing the number of clusters of one.

To measure similarity between two clusterings, we define MoJoSim:

$$MoJoSim(A, B) = 1 - \frac{MoJo(A, B)}{n} \quad (1)$$

where n is the number of entities to be clustered.

EdgeSim. This metric was defined by Mitchell and Mancoridis (2001a) to deal with the shortcomings of MoJo, that does not take into account the edges between entities.

Given a graph $G = (V, E)$ representing the structure of a system, V being the set of source code entities, and E the set of weighted dependencies between entities, EdgeSim counts a set of pairs of edges which are either intra-cluster or inter-cluster in both clusterings A and B. This collection of edges is called the Υ set. From the computation of the Υ set, ***EdgeSim(A, B)*** is defined as:

$$EdgeSim(A, B) = \frac{weights(\Upsilon)}{weights(E)} \tag{2}$$

where $weights(\Upsilon)$ is the sum of the weights of the edges of the Υ set.

MeCl. Designed by Mitchell and Mancoridis (2001) to complement EdgeSim, MeCl measures the similarity between two clusterings from a different perspective, considering both vertices and edges.

$$MeCl(A, B) = 1 - \frac{weights(\Upsilon_B)}{weights(E)} \tag{3}$$

where $weights(\Upsilon_B)$ is the sum of the weights of the set of edges that are intra-cluster in A but are inter-cluster in B, generating costs when inserting new inter-cluster edges. When the edges are not weighted, their weight is taken as equal to one. MeCl is not reflexive, i.e., $MeCl(A, B) \neq MeCl(B, A)$, thus we take the minimum of both to measure MeCl.

2.2. Intrinsic Metrics of Clustering Quality

To evaluate the metrics, we compared the values of authoritativeness to an intrinsic metric of clustering quality named *Silhouette coefficient*. This is an internal clustering evaluation method used when there is no ground truth to compare to. It produces higher values when clusters are compact and well spaced from each other. Equation 4 shows its definition.

$$S = \frac{b(i) - a(i)}{\max\left(a(i), b(i)\right)} \tag{4}$$

where $a(i)$ is the average distance of the item i to all other items in its cluster, and $b(i)$ is the average distance to all the items in the closest cluster to the given i.

The Silhouette of a cluster is the average of the Silhouettes of the cluster items. The Silhouette of a clustering is the average of the clusters' Silhouettes. The Silhouette value ranges from -1 to 1, indicating a good clustering quality as it approaches 1.

2.3. Variability Metrics for Evolving Software Systems

To evaluate the adequacy of a stability metric, we need intrinsic measures of software variability. Thus, we used information on software system variability over time. Given the current version n of a system and its next version $(n + 1)$, we may derive measures of change existence and magnitude, which we name **deltas**. We used two types of deltas: one in the number of lines of code (ΔLOC) and another in the number of classes ($\Delta Classes$), both computed between two consecutive versions of a software system.

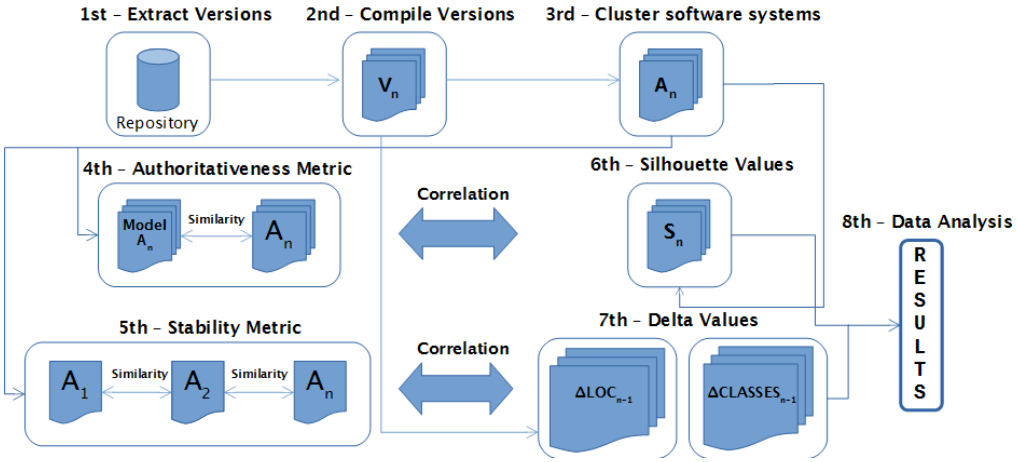


Figure 1. Experimental Design.

3. Experimental Design

Figure 1 shows the experimental design of this study. In the first step, weekly versions of the open source software systems are extracted. In the second, these versions are compiled and their designs, i.e., simplified system graphs, are extracted from the compiled files. In the third, we apply the clustering algorithms to the extracted designs. In the fourth and fifth steps, the clusterings are evaluated through authoritativeness and stability metrics. In the sixth, authoritativeness values are correlated with Silhouette values. In the seventh, stability values are correlated with ΔLOC and $\Delta Classes$. Finally, in the eighth, the results of authoritativeness and stability are studied in terms of descriptive statistics, and the computed correlations are analyzed to compare the clustering similarity metrics.

Table 1 lists the algorithms we used in our evaluation. The agglomerative algorithms compute similarity between clusters through information retrieval (IR) techniques. The IR techniques use only the vocabulary of code identifiers (e.g., class names, method names), following a pipeline of tokenizing, stop-word removal, normalizing, stemming, tf-idf computation and LSI reduction. We used the cut-off points 75 and 90 for the agglomerative algorithms such as presented in the work of Wu et al. (2005). In our evaluation, we use the software systems listed in Table 2. All of them are open source software systems developed in Java.

Table 1. Clustering Algorithms Used in the Analysis.

Name	Type	Linkage Rule
SL75	Hierarchical	Single
SL90	Hierarchical	Single
CL75	Hierarchical	Complete
CL90	Hierarchical	Complete

Table 2. Target systems with ground truth architectures.

System	Analysis Interval	# Classes	# Modules
SweetHome3D	03/08/09 to 02/28/10	142 to 165	9
Ant	10/29/06 to 10/21/07	487 to 511	16
Lucene	03/21/10 to 03/13/11	473 to 513	7
ArgoUML	11/19/06 to 11/11/07	1388 to 1524	19

4. Results

Here we describe our evaluation in terms of authoritativeness and stability.

A good clustering similarity metric is expected to have sufficient discriminatory power to identify differences between the clustering generated by the clustering algorithm and the clustering generated by experts: small differences should produce large values of authoritativeness while large differences should produce small values of authoritativeness. In a preliminary evaluation, we computed authoritativeness for each metric.

In addition, we compared the authoritativeness values of each metric with the Silhouette values of the algorithmic clustering. To do so, we computed the Silhouette in the clusterings generated by the algorithms in each system and measured the Pearson correlation between the values of authoritativeness and Silhouette.

Since the Silhouette generates values from -1 to 1, we normalized Silhouette values through the expression $S_{normalized} = \frac{S+1}{2}$ to bring Silhouette to the metrics scale.

Figure 2 illustrates data concentration and dispersion for the authoritativeness computed with each metric. Looking at the box plots, when comparing model-based metrics with each other, MeCl had higher median values of authoritativeness with a good dispersion range, followed by EdgeSim (with the exception of ArgoUML), and then Purity, MojoSim, B-Cubed-F1, followed by F1, with greater dispersion. For all box plots in this section, the axis named as *pr* is the F1-measure of the Precision-Recall values, while the axis named as *bcubed* is the F1-measure of the B-Cubed Precision-Recall values.

Figure 3 shows the correlations of each metric with Silhouette. We noticed from them that, in the per system graph, the measures of Purity, B-Cubed-F1, and F1 more strongly correlated with Silhouette. In the per metric graph, there was a trend of reducing the correlation of each metric with Silhouette with the increase of system size.

A good clustering similarity metric is expected to have sufficient discriminatory power to identify differences between consecutive versions of systems: small changes should produce large values of stability while large changes should produce small values of stability. In a preliminary evaluation, we first computed stability for each metric.

In addition, we compared the stability values of each metric with the deltas in both lines of code and in the number of classes between consecutive system versions. To do so, we computed the clustering stability produced by the algorithms for each system and measured the Pearson correlation between the stability for each metric and the ΔLOC or the $\Delta Classes$.

Figure 4 illustrates data concentration and dispersion for the stability values computed with each metric. We noticed that most metrics maintain high stability values, which is consistent with minor weekly changes. However, we also noticed that the stability of most metrics produced outliers with less stability, which captures situations where slightly larger changes occurred in the systems. Comparing the metrics by looking at the box plots, we noticed that MeCl had higher median values of stability, followed by EdgeSim (with the exception of ArgoUML), MojoSim, Purity and B-Cubed-F1. F1, on the other hand had very low stability values.

On the other hand, it is important to compare stability oscillations with variations of the evolving software itself, measured by ΔLOC and the $\Delta Classes$. It is expected that

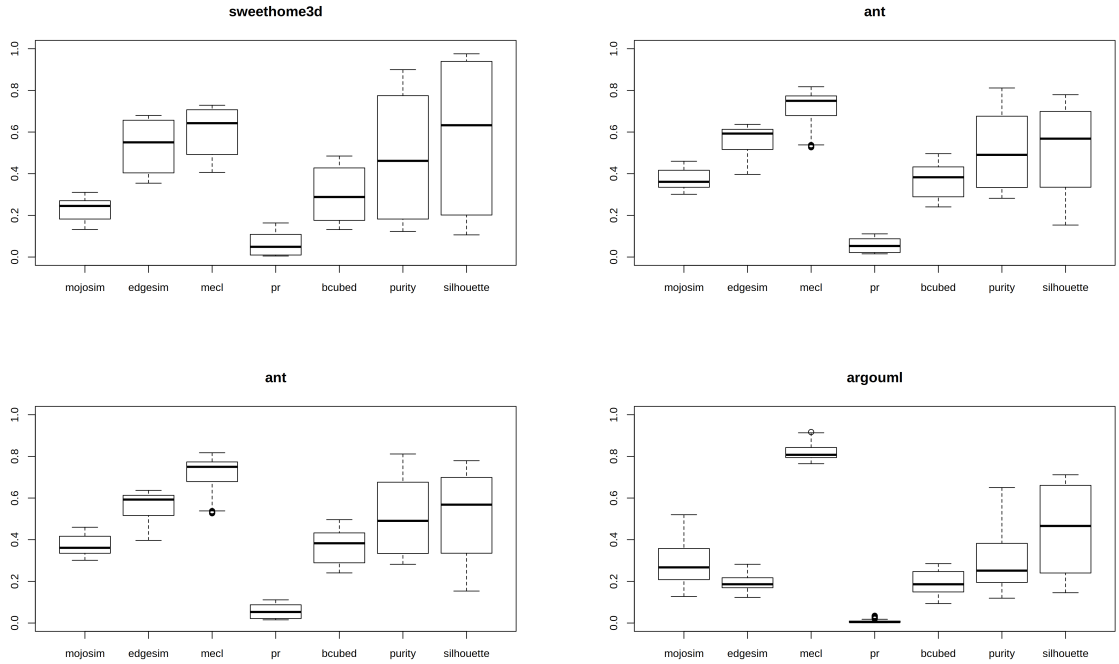


Figure 2. Authoritativeness and Silhouette per System

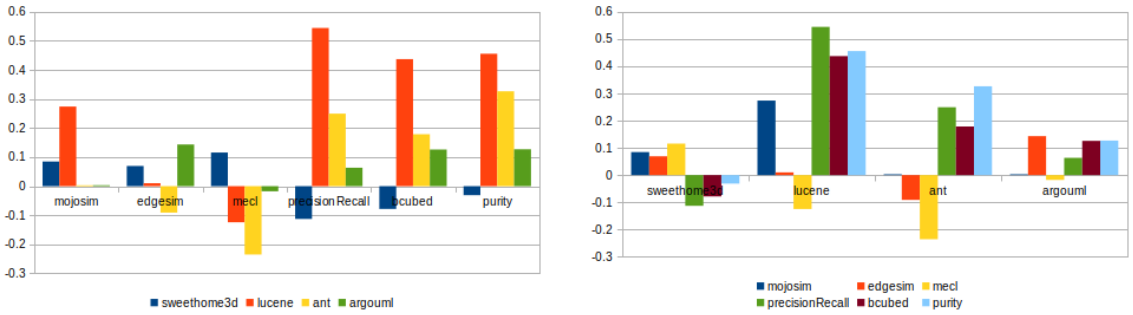


Figure 3. Correlations of Authoritativeness with Silhouette per System and per Metric.

stability correlates inversely with the deltas, i.e., the greater the changes in the software from one week to the next, the lower the stability.

Figures 5 and 6 show the correlations of each stability metric with ΔLOC and $\Delta Classes$. Since all correlation values were negative, we inverted their signs to facilitate understanding. The graphs show that, except for F1, all metrics obtained high correlations with both deltas in the graphs per system. However, with smaller systems, correlations were higher with software engineering metrics, and, with larger systems, they were higher with the classification metrics. In the per metric graph, there was a trend of correlation increase from small (SweetHome3D) to medium systems (Lucene), with later decrease with medium (Ant) and large systems (ArgoUML).

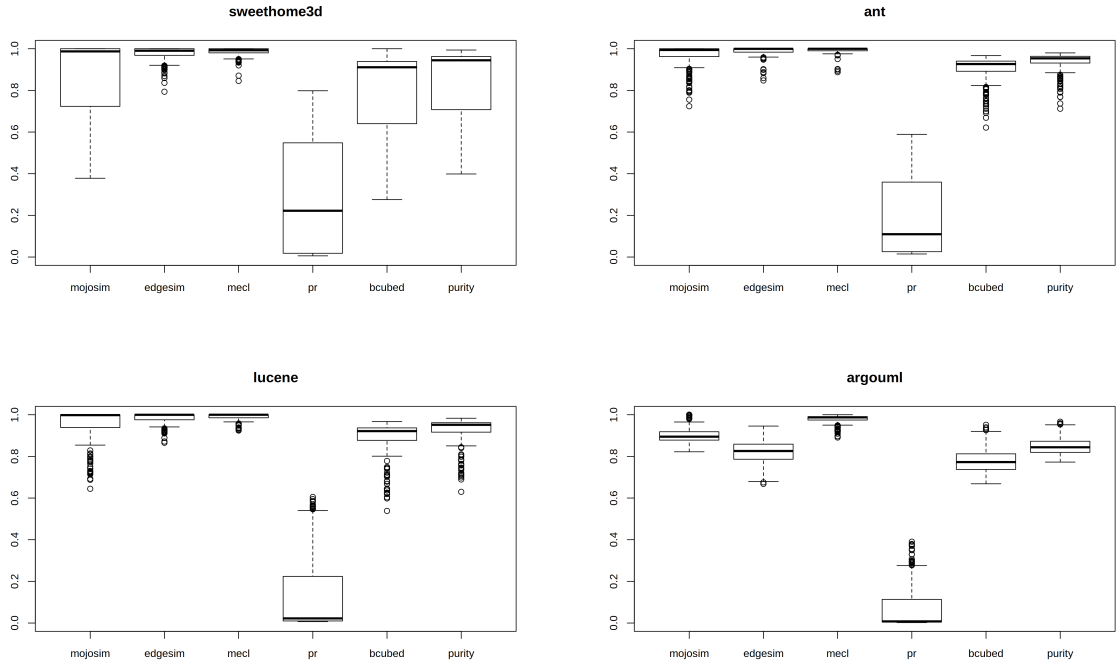


Figure 4. Stability per System

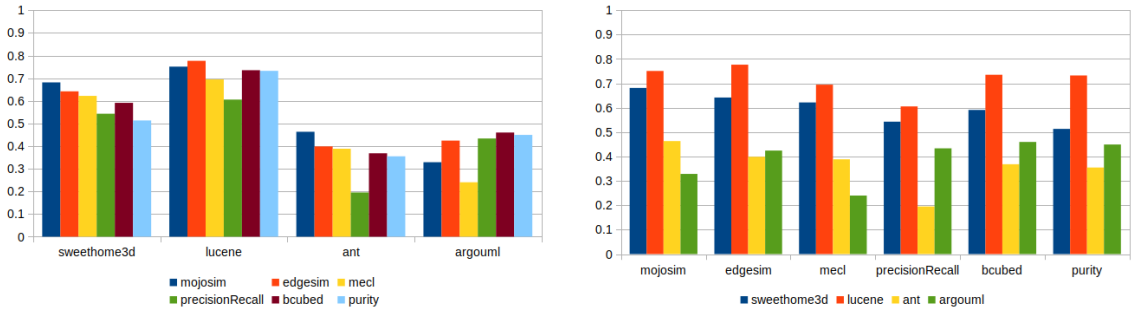


Figure 5. Correlations of Stability with ΔLOC per System and per Metric.

5. Discussion

From our experiments, most metrics generally show good authority and stability results except for F1 as shown in the box plots (Figures 2 and 4). However, when we look closely at the dispersion and concentration of stability and authoritativeness, we find that MeCl values are better when compared to the other metrics for both dimensions.

For authoritativeness, MeCl presents a range of values between 0.6 and 0.8, larger than the other metrics. This leads us to believe that, in the existence of reference models and from the data that we had available, MeCl would be the best metric to be used. However, if we consider Silhouette as the best intrinsic measure of clustering quality, Purity, B-Cubed-F1 and F1 show stronger correlation between authoritativeness and Silhouette.

For stability, all metrics showed high stability values. Even so, MeCl showed the highest median values, close to 1, in all systems, in addition to also showing outliers. With these results, MeCl would be the best candidate for stability. When looking at the

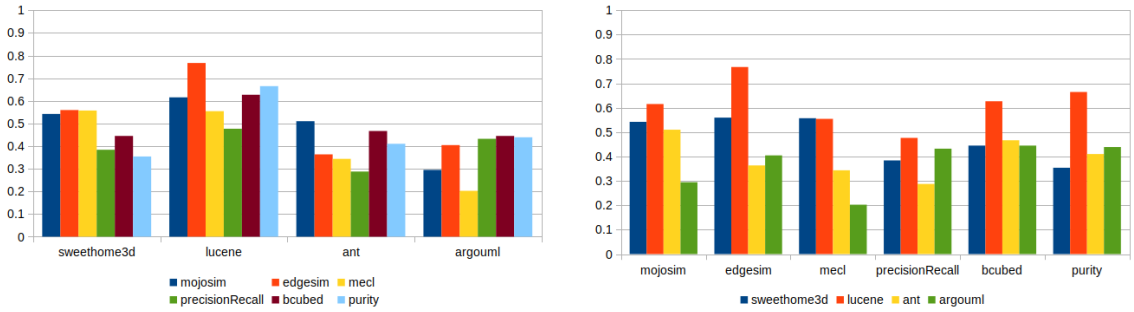


Figure 6. Correlations of Stability with $\Delta Classes$ per System and per Metric.

correlations with ΔLOC and $\Delta Classes$, we identify a trend of higher correlations for B-Cubed-F1, Purity, and F1 metrics as systems grow. Initially, all metrics show a high correlation around 0.7. However, the traditional classification metrics manage to maintain values somewhat larger with ArgoUML when compared to the software engineering metrics (e.g, MojoSim, EdgeSim and MeCl) with ArgoUML.

6. Conclusions

This work evaluated six clustering similarity metrics as candidates for measuring the quality of clusterings generated by clustering algorithms. For authoritativeness, MeCl stands out from the other metrics followed by EdgeSim (except for ArgoUML), then by Purity (greater dispersion), MojoSim, B-Cubed-F1, and F1 when the metrics are compared to each other. However, in the absence of reference models, the correlation with the Silhouette is stronger with Purity, B-Cubed-F1, and F1 metrics. For stability, MeCl also stands out with high median values in all cases. Correlations with ΔLOC and $\Delta Classes$ increased from Sweethome3D to Lucene and were lower for both Ant and ArgoUML.

As future work, we intend to evaluate agglomerative algorithms based on structural dependencies and to compare them with those based on information retrieval.

References

Anquetil, N. and Lethbridge, T. C. (1999). Experiments with clustering as a software remodularization method. In *6th Working Conference on Reverse Engineering*.

Garcia, J., Ivkovic, I., and Medvidovic, N. (2013). A Comparative Analysis of Software Architecture Recovery Techniques. In *Int'l Conf. Automated Software Engineering*.

Lutellier, T., Chollak, D., Garcia, J., Rayside, D., Kroeger, R., Tan, L., Rayside, D., Medvidovic, N., and Kroeger, R. (2015). Comparing Software Architecture Recovery Techniques Using Accurate Dependencies. In *37th Int'l Conf. Software Engineering*.

Mitchell, B. S. and Mancoridis, S. (2001). Comparing the decompositions produced by software clustering algorithms using similarity measurements. In *International Conference on Software Maintenance*.

Tzerpos, V. and Holt, R. C. (1999). MoJo: a distance metric for software clusterings. In *6th Working Conference on Reverse Engineering*.

Wu, J., Hassan, A. E., and Holt, R. C. (2005). Comparison of Clustering Algorithms in the Context of Software Evolution. In *21st International Conf. on Software Maintenance*.

How Workspaces Influence Software Development?
Preliminary Results of a Systematic Literature Review

Victor G. J. Costa[1], Cesár França[2]

[1]CESAR School – Recife – PE – Brazil

[2]Departamento de Computação
Universidade Federal Rural de Pernambuco (UFRPE) – Recife – PE – Brazil

vgjcosta@gmail.com, cesar@franssa.com

Abstract. *This work brings to light the preliminary results of a literature review showing what physical elements of workspaces have been studied in software engineering, and what are their known impacts on software development performance. A systematic literature review has been conducted, covering a period of 16 years of publications in software engineering field. Seven dimensions of workspace factors were mapped and we present the impacts of these factors on elements of software development performance, such as communication and collaboration. This article evidences the fact that there is not a generally accepted best model for software development workspaces, and that there is still much room for investigation in this topic*

1. Introduction

An organization's workspace constitutes a powerful role that provides for its employees' ways to carry out their work activities [1]. The term "office layout" refers to how the arrangement and boundaries of workspaces are laid out. Office layout has a major social and economic importance for companies as employees spend a significant amount of time in the organization spaces to performing their roles [2]. A wide body of literature indicates and several studies from different disciplines haves already demonstrated that workplace's physical environments impacts on the perception, behavior, and performance of people at work [3].

The workspace should provide a way that maximize efficiency, enabling individuals to work with ease. It means that the workspaces needs to have characteristics that support individual and group interactions through spaces, furniture and technologies that enhances communication, coordination, and collaboration [4].

In the present study, we are interested in understanding the aspects related to the arrangement and layout of the workplace and the impact of this variables on the work of a software development related environment. The research reported in this article was guided by the following question: What aspects of office layout and arrangement could boost or hinder the performance of a software development environment?

This paper is organized in the following way: In the next section, a comprehensive literature review has been provided to support our findings. Section 3 presents the research method and details of the conducted systematic literature review. In Section 4, the results along with discussions are presented. In section 5, the conclusions, limitations and future work are addressed.

2. Literature Review

In relation to an organization, an office refers to a specific area where individuals perform professional and business activities [5]. Traditionally and according to the literature, office types can be described as traditional (sometimes also referred as enclosed offices or cell offices), which are usually made up of private spaces delimited by walls, rooms and partitions that accommodates a small number of people or they can be described as open plan, that has no divisions and can accommodate a large number of individuals. In case of traditional workplaces, they have all the needs to do the job reunited at the same place.

In addition to the traditional and open plan types of offices, in the literature it is possible to find another term called agile workspaces or agile offices that have characteristics related to traditional and open plan types of office, but presents an another kind of work approach composed of a great variety of work configurations, such as: shared desks, informal spaces, relaxation rooms and contemplative spaces, so occupants can do their activities without a previously defined space [6].

Keeling et al [6] in their study compared agile workspaces with traditional types of office design such as open spaces and private environments to investigate the effects on privacy, crowding and satisfaction at work. The result has shown that agile workspaces is a distinct typology from open plan typologies and cellular offices.

Santos et al [7] still mentions the semi-open office type, that consists of placing sub-teams close to each other to promote face-to-face communication and that can be considered as an alternative when the adoption of the open plan type is not possible due to space limitations.

Rola et al [2] demonstrated an office layout model for agile IT projects managed by the Scrum framework. The proposed model is based on a cellular office layout inspired by honeycombs and composed by five cell types: conference cell, social/kitchen cell, chill out cell, development team cell and product owner cell. The model proposed by the authors restructured an existing open plan office space for the needs of an Agile project that follows the principles of the Scrum framework.

Another factor related to the work environment is the localization where a project or the team perform their activities. Co-located teams refers to individuals who are geographically close to perform their activities and has the benefit of direct, efficient and face-to-face communication, in addition to close customer proximity [8]. A distributed team is characterized by teams or most individuals in a project who are working geographically dispersed [9]. There is another approach related to team localization, called radical collocation, also known as war rooms, which is a strategy that involves putting the team (including a customer representative) in a single room [10].

Teasly et al [10] carried out a study to know if a radically collocated team lead to a higher productivity, shorter schedules, high costumer and team satisfaction and if there are any improvements in project performance. The results shown that, when people are radically collocated, there is a significant increase on productivity and timeliness due to continuous communication which improves consequently the team awareness.

Clarke and O'Connor [11] proposed a reference framework to understand the characteristics of the situational factors that affect the software development process. The researchers conducted a case study through observation and interview with this reference framework in a small-sized company and concluded that the company has a current proper software process according to situational factors. The organization has not designed, redesigned or adapted its software process using the framework [12].

3. Methods

Due to the exploratory character of this work, a systematic literature review was conducted, an evidence-based method proposed by Kitchenham et al [13], for conduct software engineering studies, which was inspired by an approach for synthesizing evidence on medicine studies.

This theme demands research in a multidisciplinary approach, not only related to Software engineering. Therefore, to ensure a good starting point, we opted to perform an "ad-hoc" search in this preliminary study, to get a comprehensive map of the relevant terminology. In the next steps of this research, we plan to perform a systematic review with automated searches to enrich and complement the present findings.

We performed searches on the following engines: ACM Digital Library, IEEE Xplore and SCOPUS to find articles published between 2002 and 2018. We used the following words: "Agile", "Agile practices", "Software Engineering", "Software Development", "Workplace environment", "Physical settings", "Office layout", "Office type", "Office rearrangement", "Facilities Management" , "Office buildings", "Seating", "Crowding", "Occupant density", "Team collocation", "Dispersed teams" and "Situational factors".

Manual searches were performed on the bibliographic sources reported on this study. First, all the titles were read, and obvious irrelevant papers were removed. Then, an inclusion and exclusion process were applied for the potentially relevant papers, as described below.

Papers are eligible according to the inclusion criteria: (a) Studies related to the design and arrangement of the physical work environment, (b) Studies reporting the influence of the workplace and its constituents related to the performance and the success of the organization (c) Studies written in English (d) Studies published after 2002 and (e) The most recent version of the study.

We selected 17 relevant studies in the sources that we searched with the set of words listed previously, which then accounted as the final selection of studies.

The quality of the papers was simple evaluated using a set of criteria developed by the Centre for Reviews and Dissemination (CRD) Database of Abstracts of Reviews of Effects (DARE). The criteria are based on four questions: (a) Are the review's inclusion and exclusion criteria well described and appropriate? (b) Is the literature search likely to have covered all relevant studies? (c) Did the reviewers assess the quality/validity of the included studies? (d) Were the basic data/studies adequately described?.

Our findings are treated following the qualitative meta-summary method proposed by Sandelowski and Barroso [14]. This method consists in five techniques:

extracting of findings, editing findings, grouping findings, abstracting findings and calculating frequency and intensity effect sizes.

4. Results

We identified a wide range of content, classifications and terminologies that are related to this approach, which may lead to a misinterpretation and also sometimes an inconsistent use of language in accordance to what related by Clarke and O'Connor [11], in their study.

Regarding to the work environment we considered two types. As physical we associated to people who are working in the same physical ambient. As virtual, we are considering the work that is related to a distributed approach and are supported by technology.

We have defined seven core dimensions. Below we explain each one of them.

Environment use practices: Under this dimension, there are characteristics that are related to practices and manners of how individuals occupies the workspace.

Layout pattern: Refers to the arrangement and how the boundaries of workspaces are laid out. Here, there where factors like Combi office, which according to Danielsson and Bodin [15] are characterized by having individual workstations in either an individual room or an open plan office.

Macro-environment: This dimension is defined by characteristics related to the whole workspace environment. It is composed by factors like the “Multiple teamwork environment” which refers to teams that are working on the same project in a specific office [2].

Micro-environment: This dimension treats about elements of office spaces, like spaces such as common areas, meeting spaces and others singular rooms, also there is a correlation with collocation practices.

Office design practices: Here, there are the factors that are related to characteristics and practices about the conception of office workspace design.

Software engineering practices: Under this dimension, there are factors linked to already known and explored software engineering practices that can be easily found on literature. "On-site customer", for example, brings a customer representative to work with the project team physically [8].

Support tool: This dimension is defined by both physical and digital tools that are used to perform the work on the organization.

Workplace characteristics could generate outcomes that impacts in a positive or in a negative manner different domains that are related to the work environment as presented in Figure 1. These domains are driven by a technical or an operational approach and are also related to human aspects.

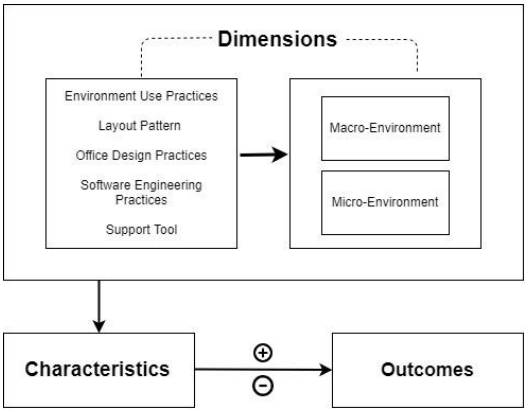


Figure 1. The relation between dimensions, characteristics and outcomes.

In relation to the characteristics, some of them were presented in the definition of the seven dimensions described above. To avoid a possible misinterpretation, we labeled the characteristics with same meaning in unique classifications, since different words were found in the literature for the same meaning. We have defined 77 characteristics, but due to the limitation on the size of this paper, we are presenting 10 of them in Table 1.

Table 1. List of characteristics

Characteristics
Cellular office / Collaborative spaces / Co-located agile team / Combi office / Common areas / Distributed agile team / Hoteling / On-site costumer / Open plan office / Traditional offices

Characteristics are grouped in 36 distinct groups. In Table 2 we are listing 10 of them.

Table 2. Groups of characteristics

Groups of characteristics
Agile practices / Collocation / Common areas / Documentation / Face-to-face communication / Flexibility / Half-cubicles / Individual workspaces / Information wall / Meeting spaces

We grouped our outcomes in 8 distinct groups of domains as follows:

Business: In general, this category is related to the operational environment of the organization, for example, financial and contractual considerations as payments, structure and material costs.

Communication: This classification is related to the whole communication process. Such as the mechanisms, tools, techniques, the quality and the effectiveness involved to transmit or receive a message.

Customer: Here are characteristics related to the customer involvement. Practices and actions to deliver the customer needs, to create value and to maintain or strengthen customer relations.

Management: Related to the characteristics under the domain of management's control of the project, that should provide the needs to perform the work.

Organization: We are considering here the factors that are associated with the organization environment such as the physical working arrangement and the facilities to

house the project [11], including the space maintenance, coordination and operations, also the organizational structure and the worker availability (localization) through spaces.

Personnel: This classification is related to human aspects under the work environment. We are considering the human aspects of individual or group of individuals, including characteristics of their attitudes, behaviors, competencies and experiences under their efforts as part of an organization.

Product: Characteristics of the product or the service that are in development by the project that involves performance requirements, configuration and architecture demands, software and hardware capabilities, development, maintenance and deployment phases, reuse approach and required quality [11].

Prerequisites: This classification is regarding the needs to meet the project procedures and requirements in term of processes, standards, policies and common practices to ensure that the project is in accordance to its objectives.

Regarding to the groups of outcome domains we have defined above, we have identified 51 distinct types. In Table 3 we are listing 10 of them.

Table 3. List of Outcomes

Outcomes
Clarify of ideas, problems and issues / Collaboration / Communication Quality / Coordination / Individual work and problem-solving / Organizational structure / Performance / Shared knowledge and information / Visual or acoustic distractions / Work planning

In Table 4 we are presenting our results in a limited view, since our findings resulted in considerable amount of data that are not possible to show completely due to limitation on the size of this paper. The complete table is available at the URL: bit.ly/HowWorkInfluSoftDev

Table 4. Sample of findings and correlations

Study ref.	Environment Type	Dimension	Group of Characteristics	Characteristics	Effect	Outcome Category	Outcome
Art.01	Physical	Software engineering practices	Face-to-face communication	Face-to-face communication	Positive Impact	Communication	Reliability of Information
Art.02	Physical	Software engineering practices	Team characteristics and abilities	High skilled team	Positive Impact	Personnel	Performance
Art.03	Physical	Software engineering practices	Multitasking	Team members with different work tasks	Negative Impact	Personnel	Visual or acoustic distractions
Art.04	N/A	Software engineering Practices	On-site costumer	On-Site costumer	Positive Impact	Communication	Communication quality
Art.05	Physical	Layout pattern	Open plan	Open plan office	Negative Impact	Personnel	Individual work and problem-solving

4.1. Discussions of results

Layout pattern was the dimension that presented more data in our findings, even appearing in only 9 studies (52.94%) and not being the dimension with most number of studies retrieved, it confirms the existence about a great concern by the researchers on

the topic of how offices are designed and arranged. The most dominant group of characteristics was Open plan that appeared in 8 studies (47,06%) and was the group of characteristics with the greater amount of data collected. When considering workplace strategies, open plan has become the preferred choice since its introduction in the 1960s and 1970s in north America [1]. Despite the well-known limitation related to employee accommodation, in our findings, Cellular office has presented only characteristics and outcomes with positive effects differently from the Open plan layout that has both positive and negative effects. According to Zhu [5], every office layout has its pros and cons. The best is the one who meets the organization and their employee needs.

As related to environment type, in all the 17 studies there are data related to the physical type of environment, in other hand, for the Virtual environment we have found data in only 8 studies (47,06%). Personnel was the category of outcomes with the greater amount of data collected once we could retrieve data related to this category in all the 17 studies. It could be explained by the fact that the influence of human aspects on organization's performance it's a subject that interests' researchers and companies from diverse areas. Collaboration along with Visual and acoustic distractions were the two most general dominant outcomes in our findings. They were both part of the Personnel outcome category. In case of Collaboration, it was possible to find data in 11 studies (64,71%) and for Visual and acoustic distractions we could found data in 8 studies (47,06%). In case of Visual and acoustic distractions, despite expectations, our findings demonstrated more outcomes with positive effect (62,5%) than negative (37,5%) in relation to the total amount of data.

5. Conclusion

The amount of data collected shown that there is still much room for investigation, once the 17 selected studies generated considerable results in accordance with the set of variables that we have defined and with the aims of this study. Also, the discussions presented in this paper, point to the need for deeper analysis regarding to our findings, as they were performed with an initial view related to this subject of study.

Once the present paper was conducted as a preliminary study our data and analysis may be limited. Future research is required to expand our findings and to bring new insights. As our findings were supported by a manual search approach in this study, one of the next objective is to perform a systematic review with automated searches to enrich and complement our research.

References

[1] M. C. Davis, D. J. Leach and, C. W. Clegg (2011). The Physical Environment of the Office: Contemporary and Emerging Issues. International Review of Industrial and Organizational Psychology, Vol. 26, 193-235.

[2] Paweł Rola, Dorota Kuchta, and Dominika Kopczyk (2016). Conceptual model of working space for Agile (Scrum) project team. Journal of Systems and Software, Vol 118, 49–63.

[3] Y. Hua, V. Loftness, R. Kraut, and K. M. Powell (2010). Workplace Collaborative Space Layout Typology and Occupant Perception of Collaboration Environment. Environment and Planning B: Planning and Design, 37(3), 429–448.

[4] Deepti Mishra, Alok Mishra, and Sofiya Ostrovska (2012). Impact of physical ambiance on communication, collaboration and coordination in agile software development: An empirical evaluation. Information and Software Technology, 54(10), 1067–1078.

[5] Lihong Zhu (2013). The physical office environment in technical services in ARL libraries. Library Collections, Acquisitions, and Technical Services 37, Issues 1–2, 42–55.

[6] Trevor Keeling, Derek Clements-Croome, and Etienne Roesch (2015). The Effect of Agile Workspace and Remote Working on Experiences of Privacy, Crowding and Satisfaction. Buildings, 5(3), 880–898.

[7] Viviane Santos, Alfredo Goldman, Eduardo Guerra, Cleidson De Souza, and Helen Sharp (2013). A pattern language for inter-team knowledge sharing in agile software development. Proceedings of the 20th Conference on Pattern Languages of Programs (PLoP '13), Art. 20, The Hillside Group, USA.

[8] Markus Hummel, Christoph Rosenkranz, and Roland Holten (2015). The Role of Social Agile Practices for Direct and Indirect Communication in Information Systems Development Teams Information Systems Development Teams. Vol. 36, Art.15.

[9] Helen Sharp, Rosalba Giuffrida, and Grigori Melnik (2012). Information Flow within a Dispersed Agile Team: A Distributed Cognition Perspective. In Agile Processes in Software Engineering and Extreme Programming. XP (2012). Lecture Notes in Business Information Processing, Vol 111. Springer, Berlin.

[10] Stephanie D. Teasley, Lisa A. Covi, M. S. Krishnan, and Judith S. Olson (2002). Rapid Software Development Through Team Collocation. IEEE Transactions on Software Engineering. 28(7), 671–683.

[11] Paul M. Clarke and Rory V. O'Connor (2012). The situational factors that affect the software development process: Towards a comprehensive reference framework. Information and Software Technology, 54(5), 433–447.

[12] Görkem Giray, Murat Yilmaz, Rory V. O'Connor, and Paul M. Clarke (2018). The Impact of Situational Context on Software Process: A Case Study of a Very Small-Sized Company in the Online Advertising Domain. Systems, Software and Services Process Improvement. Proceedings of the 25th European Conference, (EuroSPI 2018), Bilbao, Spain, Vol. 896, 28-39, Bilbao, Spain.

[13] B. A. Kitchenham, T. Dyba, and M. Jorgensen. (2004). Evidence-based software engineering. Proceedings of the 26th International Conference on Software Engineering, (ICSE '04), IEEE Computer Society, Washington DC, USA, 2004, 273–281.

[14] M. Sandelowski and, J. Barroso (2007). Handbook for Synthesizing Qualitative Research, Springer Publishing Company, New York.

[15] C. B. Danielsson and L. Bodin (2008). Office Type in Relation to Job Satisfaction Among Employees. (2008), 636–668.

Análise e melhoria do processo de reengenharia de *software*: um estudo de caso

Guilherme Mendonça de Moraes[1], Prof. Dr. Edgard Costa Oliveira[2]

[1]Programa de Pós-Graduação em Computação Aplicada (PPCA) - Universidade de Brasília (UnB), CEP: 7090-970 – ICC Centro, Módulo 14, Subsolo CSS 361, Campus Universitário Darcy Ribeiro – Brasília, DF - Brasil

[2]Faculdade de Tecnologia – Engenharia de Produção (FT/ERP) – Universidade de Brasília (UnB), DF - Brasil.

guilherme.moraes@aluno.unb.br, ecosta@unb.br

Abstract. *Although many authors seek to relate risk management to failure or success in software reengineering projects, the union of the two themes remains a major challenge for organizations that contract or provide such products and services. This article aims to analyze and improve the process of software reengineering through risk management. The methodology adopted was the case study of a project to change the technology of software in a large financial institution, which resulted in the improvement of the software reengineering process in the company.*

Resumo*. Apesar de muitos autores procurarem relacionar gestão de riscos com fracasso ou sucesso em projetos de reengenharia de software, a união das duas temáticas ainda é um grande desafio para as organizações que contratam ou fornecem produtos e serviços desse tipo. Esse artigo tem por objetivo a análise e melhoria do processo de reengenharia de software por meio da gestão de riscos. A metodologia adotada foi o estudo de caso de um projeto de mudança da tecnologia do software em uma instituição financeira de grande porte, o que resultou na melhoria do processo de reengenharia de software dessa instituição.*

1. Introdução

Com o avanço tecnológico e o crescimento das necessidades de mercado, é comum que as empresas optem por realizar a mudança de sua estrutura tecnológica atual visando a compatibilidade com outras linguagens, banco de dados e plataformas, como também o aumento do desempenho, usabilidade, acessibilidade, segurança e qualidade [Pinto e Braga, 2004; UTFPR, 2010; Machado, 2011].

A mudança da tecnologia visa não somente adaptar o processo de *software* ao de negócio [Weerakkody e Currie, 2003], mas também adequá-lo às novas tecnologias, corrigindo erros conhecidos em ambiente produtivo, como itens de qualidade expostos pela ISO 25010 (2011): funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade, fazendo com que o aplicativo cumpra requisitos específicos em tempo real [Vogel-Heuser et al., 2014].

De acordo com Pressman (2005) essa atividade de reengenharia de sistemas de computacionais consome até 70% dos esforços durante todo o ciclo de vida do sistema.

Além da reengenharia de *software* ser uma tarefa onerosa, existe um alto risco nesse processo de mudança da tecnologia [Pressman , 1995; Reis et al., 2003; Vogel-Heuser, 2010], pois o sistema legado precisa continuar funcionando sem que essas modificações causem um efeito negativo no mesmo em ambiente produtivo.

Juntamente com essas alterações existem os riscos que as mesmas representam para o negócio da instituição [Weerakkody e Currie, 2003; Sneed, 2005; Chen et al., 2009], pois a necessidade é que a toda a estrutura dos aplicativos seja alterada sem que o programa em produção fique comprometido.

Conforme a ISO 31000 (2009) risco é o efeito da incerteza nos objetivos. Em outras palavras, é algo que pode causar um desvio nos propósitos comuns. Nesse contexto, o objetivo está relacionado às expectativas de qualidade, funcionamento correto e conformidade com requisitos legais e institucionais do *software* migrado, tendo como base que suas vulnerabilidades estão ligadas diretamente à ausência de determinados controles [Elahi et al., 2010].

Dessa forma, o presente trabalho tem por objetivo a análise e melhoria do processo de reengenharia de *software* desenhando o processo atual (*AS-IS*), analisando as vulnerabilidades do processo, propondo um desenho do mesmo (*TO-BE*) com as melhorias utilizando ferramentas e técnicas para gestão de riscos.

O estudo de caso em questão será aplicado a um projeto de mudança da tecnologia de aplicativos computacionais em uma instituição financeira de grande porte, que por sigilo terá sua identificação nesse artigo como Instituição Financeira Creditar.

2. Desenho do processo atual (*AS-IS*) de reengenharia de *software*

Devido à falta de gestão de riscos nos processo de migração da tecnologia do *software*, a Instituição Financeira Creditar reportou um incidente em ambiente produtivo após migração de um aplicativo computacional e o mesmo tendo passado por todas as atividades de desenvolvimento, teste e homologação, que creditou várias vezes o salário de clientes, que por sua vez causou um grande prejuízo financeiro, dano à imagem da instituição:

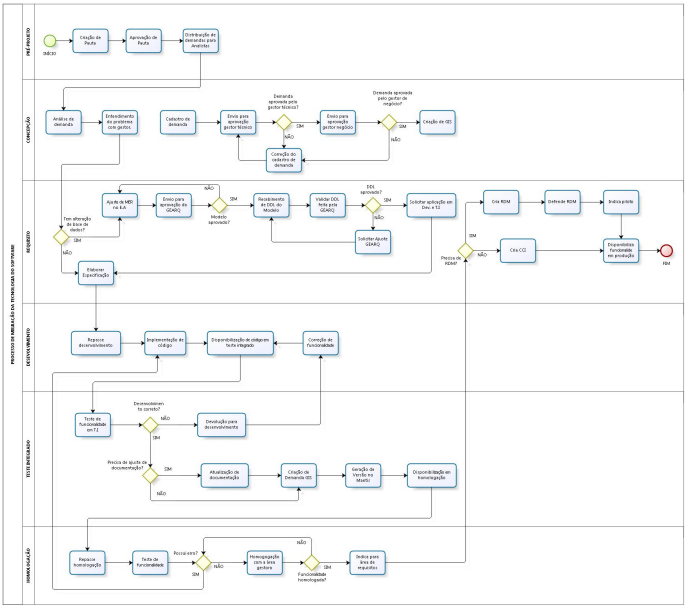


Figure 1. Diagrama do processo atual de reengenharia de *software* da Instituição Creditar

Utilizando a técnica de mapeamento de processos BPMN - *Business Process Model and Notation* [Chinosi e Trombetta, 2012] e o programa Bizagi, realizado o desenho AS-IS do processo de migração da tecnologia de *software* onde o mesmo foi dividido em 6 grandes áreas sendo: pré-projeto, concepção, requisitos, teste integrado e homologação conforme imagem anterior.

3. Identificação dos riscos no processo atual de reengenharia de *software*

Como a proposta da ISO 31000 (2009) é a identificação, análise, avaliação e tratamento dos riscos, é necessário em primeiro ponto realizar a identificação dos mesmos relacionado à todas as atividades da organização. Para tal tarefa, foi utilizada a técnica *Brainstorming* que é uma ferramenta indicada pela ISO 31010 (2009) e uma forma obter uma lista completa dessas ameaças. Após tais tarefas, foi construído um desenho com uma estrutura da análise SWOT - *Strengths*, *Weaknesses*, *Oportunities*, *Thereats*, porém, considerando apenas as oportunidades e ameaçadas identificadas no Projeto Renovação CCO da instituição estudada:

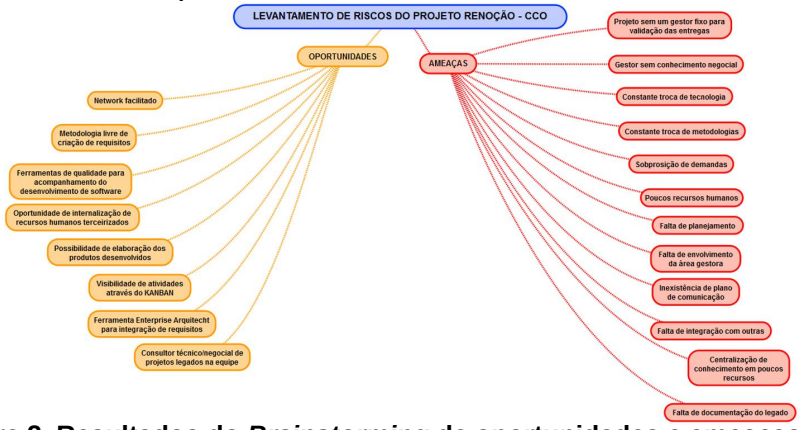


Figure 2. Resultados do *Brainstorming* de oportunidades e ameaças no processo de reengenharia de *software*

Para identificar o que foi considerado como ameaça, foram utilizados ferramentas, técnicas e melhores práticas para reengenharia de *software* e serviços de TI como: *Rational Unifield Process* [IBM, 2001], SCRUM [Schwaber e Sutherland, 2011], *Xtreme Promming* [Beck e Gamma, 2000], ITIL [OGC, 2011], ISO 25010 (2011) e o PMBOK (2018) e com isso criada uma base de conhecimento após essa revisão bibliográfica para identificar os controles aplicáveis às vulnerabilidades no processo de reengenharia de aplicativos do estudo de caso, como a seguir:

Quadro 1. Modelo CGU (2018) de base de conhecimento com a documentação dos controles

Aplicação: **Gestão de Projetos**	Ativo: **Tecnologia**	Tipo: **Norma**	*Knowledge base*: **ISO 25010 (2011)**
Descrição do controle	Deve-se criar um documento de Interoperabilidade.		
O quê?	Artefato para documentar a capacidade do produto de *software* de interagir com um ou mais sistemas especificados.		
Por quê?	Mitigação dos riscos de erro de integração do novo sistema migrado com outros *softwares* necessários.		

Após o *Brainstorming* e com ajuda de gestores do negócio, algumas partes interessadas e um gestor de riscos, chegou-se ao conhecimento da existência de cerca de 150 vulnerabilidades em todo o processo de reengenharia de *software*, das quais foram selecionadas 25 para análise, avaliação e tratamento, sendo elas de maior relevância para a instituição de acordo com os especialistas envolvidos no nessas atividades.

Foram utilizadas como referência a norma ISO 25010 (2011), PMBOK (2018), o RUP [IBM, 2001], ITIL V3 [OGC, 2011], Xtreme Promming [Beck e Gamma, 2000] e o SCRUM [Schwaber e Sutherland, 2011] para identificação dos controle aplicáveis às vulnerabilidades no processo de mudança da tecnologia do *software*. Com isso, foi realizada a documentação dos riscos como tais como esses:

Quadro 2. Identificação de riscos com a especificação do impacto e aplicação dos controles (Adaptado pelo próprio autor do modelo CGU (2018))

Etapa	Vulnerabilidades - Fatores de Risco	Impacto	Controles Aplicáveis
Pré-projeto	Divergência de planejamento e execução de pauta e especificação das necessidades e expectativas de cada projeto	Mudanças não mapeadas	É necessário identificar todas as necessidades, premissas, restrições, requisitos iniciais e principais responsáveis pelo projeto de forma a mitigar o risco de mudança de escopo, aumento do escopo, aumento de custo e prazo.
		Falta de compatibilidade com outras funcionalidades	
		Erros desconhecidos em ambiente produtivo	
		Descumprimento de requisitos negociais	
		Aumento do custo do projeto	
		Atraso no projeto	
Concepção	Falta de entendimento do sistema legado e todas as funcionalidades internas e externas impactadas por determinada mudança como também os erros existentes.	Erros desconhecidos em ambiente produtivo	Deve-se criar um documento de Interoperabilidade para documentar a capacidade do produto de *software* de interagir com um ou mais sistemas especificados.
		Atualização desnecessária de *software* ou nova funcionalidade	
		Erros desconhecidos em ambiente produtivo	Deve-se criar um documento de Capacidade do produto de *software* de prover um conjunto apropriado de funções para tarefas e objetivos do usuário especificados.
		Insatisfação do usuário final	
		Aumento do custo do projeto	Deve-se realizar a proteção frente a erros de usuários: como produto consegue prevenir erros dos usuários.
		Aumento do prazo do projeto	

4. Análise e avaliação dos riscos no processo de reengenharia de *software*

Com a utilização das ferramentas e técnicas: Análise SWOT e Análise de Restrição e Premissas [PMBOK, 2018], o gestor negocial da instituição estudada, juntamente com o gestor de riscos conseguiram identificar qual o grau de riscos de vulnerabilidade pontuando a probabilidade de determinado risco se concretizar como também o impacto que o mesmo causaria ao processo e mudança da tecnologia dos sistemas. Essas informações foram documentadas em uma matriz de probabilidade e impacto [ISO 31010, 2011] na qual foi utilizada uma pontuação de probabilidade de 0,1 a 0,9, sendo 0,1 para pouco provável e 0,9 para muito provável que determinado risco se torne um incidente, e no caso do impacto de 1 a 5, sendo 1 para impacto muito baixo e 5 para

impacto muito alto, caso o incidente aconteça. Para fins de cálculos, os valores foram multiplicados entre si gerando uma classificação de *score* com o grau de cada risco:

Tabela 1. Avaliação dos riscos (próprio autor adaptado da ISO 31010, 2009)

AVALIAÇÃO DOS RISCOS – MATRIZ DE PROBABILIDADE E IMPACTO			
Riscos	Probabilidade	Impacto	*Score*
Mudanças não mapeadas	0,7	5	3,5
Falta de compatibilidade com outras funcionalidades	0,7	5	3,5
Erros desconhecidos em ambiente produtivo	0,3	4	1,2
Descumprimento de requisitos negociais	0,1	2	0,2
Aumento do custo do projeto	0,9	2	1,8
Atraso no projeto	0,9	3	2,7
Erros desconhecidos em ambiente produtivo	0,3	5	1,5
Desenvolvimento desnecessário de atualização de *software* ou nova funcionalidade	0,2	1	0,2
Insatisfação do usuário final	0,4	5	2
Aumento do custo do projeto	0,3	3	0,9

O *score* foi avaliado tendo como base a matriz de probabilidade e impacto [PMBOK, 2018], considerando apenas as ameaçadas:

Tabela 2. Matriz de Probabilidade e Impacto (Adaptado de PMBOK (2018))

Matriz de Probabilidade X Impacto					
Probabilidade	**Ameaças**				
0,9	0,9	1,8	2,7	3,6	4,5
0,7	0,7	1,4	2,1	2,8	3,5
0,5	0,5	1	1,5	2	2,5
0,3	0,3	0,6	0,9	1,2	1,5
0,1	0,1	0,2	0,3	0,4	0,5
Gravidade	1	2	3	4	5

5. Proposta de melhoria no processo de reengenharia de *software* (*TO-BE*)

Após identificação, análise e avaliação de risco e aplicação dos controles às vulnerabilidades identificadas no processo mudança da tecnologia do *software*, como resposta a essas ameaças encontradas no anterior (*AS-IS*), foi elaborado um desenho evoluído do mesmo (*TO-BE*) como forma de mitigar os riscos identificados anteriormente, como por exemplo:

1. Na forma de elicitação de requisitos, foi identificado a necessidade de entender o sistema legado através de documentações existentes e principalmente a opinião de especialistas técnicos por *Brainstorming* e à partir daí, a elaboração de uma primeira documentação contendo as possíveis regras, telas (se aplicável) e entidades de banco de dados (se aplicável);

2. Como documentação principal de mapeamento de necessidade e desenho de funcionalidades e suas respectivas regras, foi utilizado o conceito de elaboração de interface adaptando a ideia do RUP [Kruchten, 2004; Sommerville, 2011] de forma que todos os *stakeholders* consigam opinar sobre a viabilidade, necessidades e necessidades do negócio inerente as funcionalidade migrada;

3. Processo de validação de regras, mensagens, telas (quando aplicável) e entidades de banco de dados (quando aplicável) – é cíclica iniciando no gestor técnico, equipe de desenvolvimento, representando do usuário final do *software* migrado e por último o gestor do negócio, fazendo cada vez mais a documentação e detalhamento do serviço/sistema [ITIL, 2007].

A seguir, o desenho do processo melhorado (*TO-BE*):

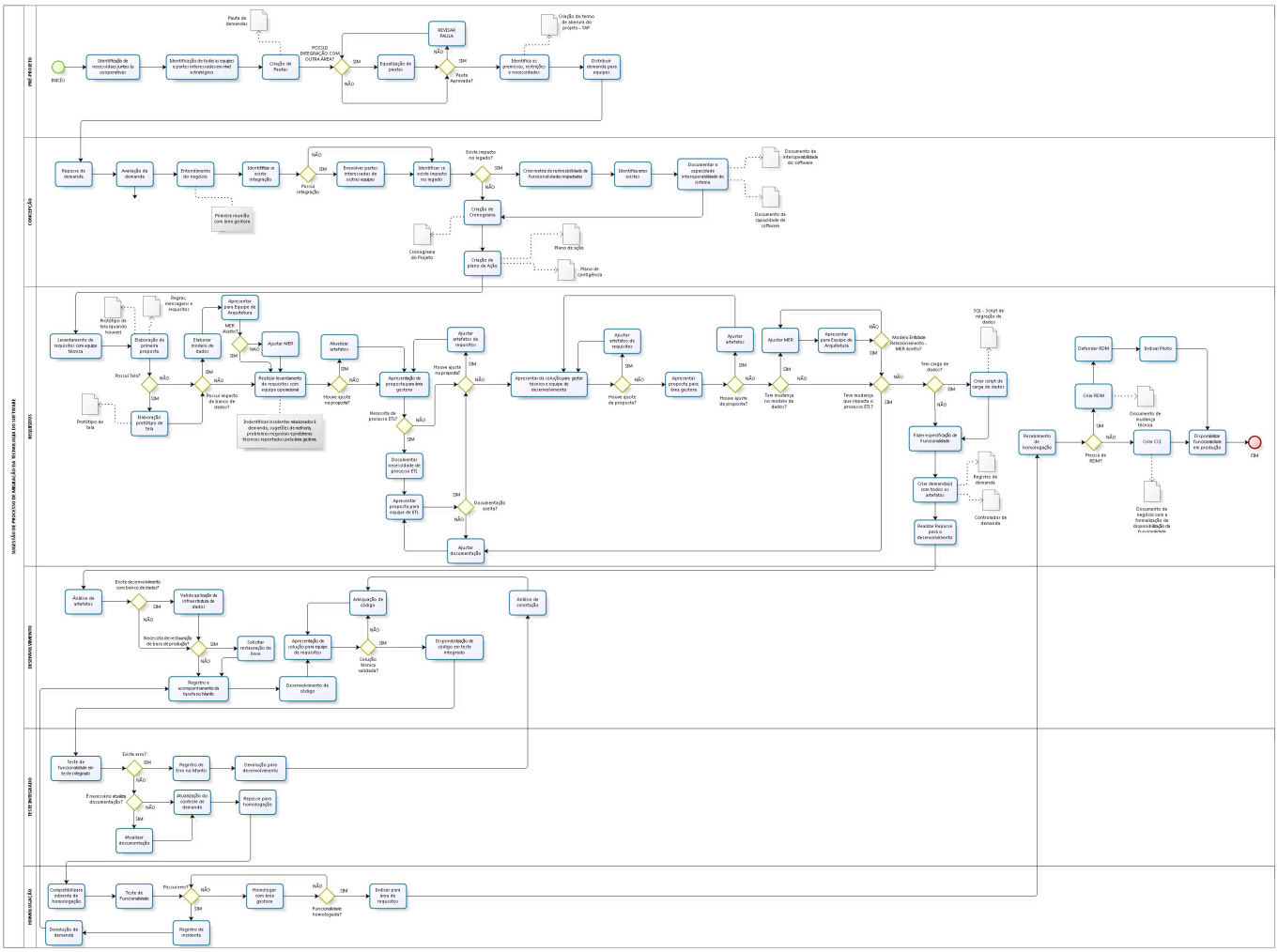


Figure 3. Desenho *TO-BE* do processo reengenharia de *software* da Instituição Financeira Creditar

Através do desenho do projeto melhorado, a área gestora pôde ter uma visão de quais atividades podem ser aplicadas para mitigar os riscos identificados no mapeamento anterior, tendo em vista que a versão aprimorada apresenta ações e documentos para reengenharia de *software* baseadas em controles de mercados como boas práticas e normas.

6. Conclusão

O estudo em questão buscou propor a aplicação da gestão de riscos como forma de análise e melhoria do processo de reengenharia de *software*, mapeamento ferramentas e técnicas para identificar, analisar, avaliar e tratar os riscos que permeiam esse processo, como também sendo as respostas à esses riscos o processo já melhorado utilizando boas práticas, normas e *frameworks*.

Foram utilizas técnicas como brainstorming, matriz de probabilidade e impacto, mapeamento de processos, apoio de especialistas assim como na revisão da literatura para identificar os pontos de atenção e riscos no processo de reengenharia de *software* e como forma de resposta a esses riscos na elaboração do processo melhorado TO-BE.

Foi utilizado como análise um projeto de mudança de tecnologia do *software* da instituição financeira em questão que possui todas as características necessárias para geração de insumo da pesquisa em questão, que além das ferramentas e técnicas, contou com o apoio de um especialista de riscos e um gestor organizacional para validação do processo melhorado. O resultado final mostrou-se bem eficiente após validação do gestor de negócio e do especialista de risco, pois conseguiu revelar os riscos que o processo de reengenharia de *software* como também mostrou as vulnerabilidades que o processo possuía, ou seja, os pontos de atenção que o , por meio de um processo melhorado, permitiu a redução dos riscos de mudança, do não cumprimento de requisitos e do não atendimento de prazo e custos e até mesmo da extinção do projeto.

Após criação e aplicação do processo melhorado, observou-se a redução dos riscos de erros em ambiente produtivo dos *software*s migrados, menor tempo de desenvolvimento, menos mudanças nos requisitos e funcionalidades e em consequência disso, diminuição do tempo e custo de desenvolvimento de todo o processo. Como recomendação para trabalhos futuros, propomos a aplicação de outras ferramentas e técnicas para gestão do risco como também uma forma automatizada para análise e avaliação do riscos identificados.

Referências

ABNT ISO GUIA 31.000:2009, Gestão de Riscos – Princípios e diretrizes, 1.ª edição, Rio de Janeiro, ABNT, 2009.

ABNT ISO GUIA 31.010:2012, Gestão de Riscos – Técnicas para o processo de avaliação de riscos, 1 ª edição, Rio de Janeiro, ABNT, 2012.

Beck, K., & Gamma, E. (2000). Extreme programming explained: embrace change. addison-wesleyprofessional.

Cagnin, M. I. (2005). Parfait: uma contribuição para a reengenharia de *software* baseada em linguagens de padrões e frameworks (Unpublished doctoral dissertation). Universidade de São Paulo.

Chaves, L. L. (2004). Sistemas legados e a aplicação de processos de reengenharia de *software*. FEA – USP , 21–23.

Chen, C. C., Law, C. C., & Yang, S. C. (2009). Managing ERP implementation failure: a project management perspective. IEEE transactions on engineering management , 56 (1), 157–170.

Chinosi, M., & Trombetta, A. (2012). BPMN: An introduction to the standard. Computer Standards & Interfaces, 34(1), 124-134.

De Bakker, K., Boonstra, A., & Wortmann, H. (2010). Does risk management contribute to it project success? a meta-analysis of empirical evidence. International Journal of Project Management, 28 (5), 493–503.

Elahi, G., Yu, E., & Zannone, N. (2010). A vulnerability-centric requirements engineering framework: analyzing security attacks, countermeasures, and requirements based on vulnerabilities. Requirements engineering, 15(1), 41-62.

Ibbs, C. W., & Kwak, Y. H. (2000). Assessing project management maturity. Project management journal, 31 (1), 32–43.

ISO ISO/IEC 25.010 (2011), *Software* Product Quality. International Organization for Standardization.

ISO/IEC 13.335-1 (2004) Information technology - Security techniques - Management of information and communications technology security.

ITIL V3 - Information Technology Infrastructure Library (2011). OGC (Office for Government Commerce.

Kruchten, P. (2004). The rational unified process: an introduction . Addison-Wesley Professional.

MACHADO, F (2011). Análise e gestão de requisitos de *software*: onde nascem os sistemas.[sl]: Editora érica;

Ministério da Transparência e Controladoria Geral da União – CGU (2018). Metodologias de gestão de riscos;

Pinto, H. L. M., & Braga, J. L. (2004). Sistemas legados e as novas tecnologias: técnicas de integração e estudo de caso. Informática Pública, Belo Horizonte, 7 (1), 48–69.

PMBOK (2018). Guia PMBOK 6ª Edição - Um Guia do Conhecimento em Gerenciamento de Projetos. PMI.

PRESSMAN, R. S (1995). Engenharia de *Software*. São Paulo: Makron Books;

Pressman, R. S. (2011). Engenharia de *Software* – Uma abordagem Profissional. 7 ed., ISBN 978-85-8055-044-3.

Rational IBM. Rational Unified Process (2001) – Best Practices for *Software* Development Teams. Rational *Software* White Paper, rev. 11/01.

Reis, C. R., & de Mattos Fortes, R. (2003). Caracterização de um processo de *software* para projetos de *software* livre (Unpublished doctoral dissertation). Universidade de São Paulo.

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

Sneed, H. M. (2005). Estimating the costs of a reengineering project. In 12th working conference on reverse engineering (wcre'05) (pp. 9–pp).

SOFTWARE ENGINEERING INSTITUTE – SEI (2010). CMMI® for Development, Version 1.3.

Sommerville, I. (2011). *Software* engineering 9th edition. ISBN-10, 137035152.

UTFPR (2010). A manutenção de *software* nas empresas. Disponível em: <http://pg.utfpr.edu.br/dirppg/ppgep/ebook/2010/CONGRESSOS/ADM/25.pdf>, Acesso em: 07 de abril de 2019.

Vogel-Heuser, B., Legat, C., Folmer, J., & Feldmann, S. (2014). Researching evolution in industrial plant automation: Scenarios and documentation of the pick and place unit (Tech. Rep.).

Weerakkody, V., & Currie, W. (2003). Integrating business process reengineering with information systems development: issues & implications. In International conference on business process management (pp. 302–320).

How do Technical Factors Affect Developers in Mobile Software Ecosystems

Caio Steglich[1], Sabrina Marczak[1], Rodrigo dos Santos[2], Luiz Pedro Guerra[1], Luiz Henrique Mosmann[1], Cleidson de Souza[3], Fernando Figueira Filho[4], Marcelo Perin[5]

[1]Escola Politécnica, PPGCC – PUCRS
Porto Alegre – RS – Brasil

[2]Programa de Pós-Graduação em Informática – UNIRIO
Rio de Janeiro - RJ - Brasil

[3]Programa de Pós-Graduação em Ciência da Computação – UFPA
Belém - PA - Brasil

[4]Departamento de Informática e Matemática Aplicada – UFRN
Natal - RN - Brasil

[5]Pesquisador Independente
Porto Alegre – RS – Brasil

{caio.borges, luiz.guerra, luiz.mosmann}@acad.pucrs.br
sabrina.marczak@pucrs.br, rps@uniriotec.br, cleidson.desouza@acm.org
fernando@dimap.ufrn.br, mperin25@gmail.com

Abstract. *The Software Evolution area brings applications to the Mobile era in which users want to use these applications on their mobile devices. A Mobile Software Ecosystem (MSECO) is the kind of ecosystems in which developers build applications to attend the needs of mobile technologies users (e.g., Android and iOS). Literature explains that the capability to attracting and retaining people (i.e., developers and users) is essencial to MSECO sustainability, i.e., to the MSECO survive along the years. In a previous work, we conducted a literature review that identified 6 factors that may influence developers to participate in an MSECO. In this study, we present a Field Study aiming to understand how these 6 identified factors may have influenced practitioners in real life projects.*

Resumo. *A área de Evolução de Software traz os aplicativos para a era da mobilidade, aonde os usuários desejam usar esses aplicativos em seus dispositivos móveis. Um ecossistema de software móvel (MSECO) é o tipo de ecossistema em que os desenvolvedores criam aplicativos para atender aos usuários de tecnologias móveis (por exemplo, Android e iOS). A literatura explica que é fundamental para a sustentabilidade do MSECO a capacidade de atrair e reter pessoas (ou seja, desenvolvedores e usuários) para sobreviver ao longo dos anos. Em um trabalho anterior, realizou-se uma Revisão de Literatura, na qual foram identificados 6 fatores que podem influenciar os desenvolvedores a participar de um MSECO. Neste estudo, apresenta-se um Estudo de Campo que teve como objetivo identificar como esses 6 fatores influenciaram profissionais da área em projetos reais vivenciados pelos mesmos.*

1. Introdução

A sociedade tem evoluído constantemente, principalmente em termos tecnológicos. Visando atender novas demandas das pessoas, soluções de softwares têm migrado mais e mais para plataformas móveis, fazendo com que os Ecossistemas de Software Móvel (do inglês, MSECO) tornem-se cada vez mais presentes, atraindo usuários e desenvolvedores.

Os Ecossistemas de Software (ECOS) são sistemas complexos no quais tem sido produzido soluções em software na atualidade. Estes ECOS são compostos basicamente por um conjunto interno e externo de desenvolvedores, uma comunidade de especialistas de domínio em serviço e uma comunidade de usuários, oferecendo assim, soluções relevantes para satisfazer as necessidades desses usuários [Bosch and Bosch-Sijtsema 2010].

Tipicamente, um ECOS pode ser visualizado por três dimensões [Campbell and Ahmed 2010]: a dimensão técnica, que compreende as tecnologias que compõem o ECOS (*e.g.*, recursos de desenvolvimento, linguagens de programação, etc); a dimensão de negócio, que compreende elementos como as vendas de aplicações, linhas de produção, entre outros; e a dimensão social, que é composta pelas pessoas que participam do ECOS, como, por exemplo, usuários ou desenvolvedores desse ECOS.

ECOS Móvel são ECOS que atuam sobre o contexto móvel [Fontao et al. 2015], ou seja, aqueles que tem como objetivo a produção de aplicações para dispositivos móveis (e.g. *smartphones*). Esta classe de ECOS possui dois grandes líderes de mercado, o Android e o iOS [Mallinson 2015]. Pela importância que estes ECOS têm, torna-se fundamental o estudo sobre sua sustentabilidade, sendo esta baseada em dois pilares principais: i) a adaptação a novas tecnologias, recursos e tendências e ii) a atração e manutenção de membros na comunidade do ECOS [Dhungana et al. 2010]. Portanto, pesquisas são necessárias para explorar estes dois pilares e identificar o papel da tecnologia, *i.e.*, a dimensão técnica, utilizada no desenvolvimento e sustentabilidade destas aplicações e MSECO como um todo.

Este trabalho tem como objetivo investigar como fatores técnicos podem influenciar os desenvolvedores de um ECOS móvel na escolha e permanência de um determinado ecossistema para atuarem. Para isso, baseado em um mapeamento sistemático da literatura anterior de autoria dos autores deste artigo que identificou esses fatores [Borges et al. 2019], realizou-se um estudo de campo com desenvolvedores de ECOS móvel, que contou com 20 profissionais. Neste estudo, buscou-se que os participantes explicassem, a partir de suas experiências, como os fatores influenciaram suas decisões de participar ou se manter participando em um ECOS móvel. Identificou-se como os 6 fatores advindos da literatura influenciaram os desenvolvedores a participarem em um MSECO.

O restante deste artigo está organizado como segue: a Seção 2 discorre sobre a metodologia utilizada para a condução desta pesquisa, a Seção 3 apresenta os resultados a partir do processo utilizado, a Seção 4 discute o impacto dos resultados obtidos, a Seção 5 explica as limitações desta pesquisa, e a Seção 6 conclui o artigo.

2. Metodologia de Pesquisa

Este estudo é baseado em um estudo anterior, um mapeamento sistemático da literatura [Borges et al. 2019], que identificou 29 fatores relacionados às três dimensões de um ECOS. Usou-se aqui os 6 fatores referentes à dimensão técnica, sendo esses:

- (F1) Recursos técnicos necessários ou desejados pelo desenvolvedor
- (F2) Desempenho do hardware das aplicações
- (F3) Plataforma facilmente configurável
- (F4) Padrões de interface e aparência das aplicações
- (F5) Segurança dos dados
- (F6) Acessibilidade e suporte a diferentes perfis de usuários

2.1. Estudo de Campo

Para a realização do estudo de campo sobre os fatores técnicos identificados, vinte desenvolvedores foram convidados para expressarem como tais fatores os influenciaram, tanto para começar quando para continuar participando de um ECOS móvel.

Singer *et al.* (2008) [Singer et al. 2008] explicam que um estudo de campo visa investigar como praticantes de alguma atividade lidam com a prática ou resolvem problemas dentro de seus respectivos contextos. Portanto, a partir de um estudo de campo e suas respectivas técnicas de coleta e análise de dados, capturou-se evidências para nosso estudo. Ainda, seguindo as recomendações dos mesmos autores [Singer et al. 2008], desenvolvedores foram convidados, sendo selecionados por conveniência e disponibilidade, contanto que cumprissem o requisito de terem experiência com desenvolvimento em tecnologias móvel. Estes desenvolvedores, todos brasileiros, têm colaborado com o MSECO Android (6 dos participantes) ou iOS (5), ou ambos (9). A experiência dos mesmos com estes MSECO varia entre 1 a 9 anos, e os portes das empresas em que trabalham são distintos–pequena (9 desenvolvedores), média (7) ou grande (4).

As entrevistas duraram em média 25 minutos, sendo a menor delas de 20 minutos e a maior de 30 minutos. Todas entrevistas foram realizadas pessoalmente, gravadas com a autorização do participante e transcritas para análise de texto seguindo a técnica de *Card Sorting* de Spencer [Spencer 2009]. Os resultados são apresentados a seguir, na Seção 3.

3. Fatores Técnicos e sua Influência sobre os Desenvolvedores

A partir das entrevistas com os 20 desenvolvedores de ECOS móvel, identificou-se a importância que os mesmos acreditam que cada fator possui. Como resultado, tem-se que a Segurança dos dados, Padrões de interface e Recursos técnicos desejados são considerados os fatores mais importantes para se motivar um desenvolvedor a participar de um MSECO. Fatores como o Desempenho do hardware ganha grande importância para que o desenvolvedor se mantenha no ECOS.

Além disso, os desenvolvedores discorreram sobre como eles veem cada um dos fatores, conforme apresentado a seguir.

(F1) Recursos técnicos desejados pelo desenvolvedor

Os recursos técnicos foram considerados importantes para começar a participação em um ECOS móvel por 14 desenvolvedores entrevistados, sendo que alguns destes destacam que os recursos técnicos são usados como critério de decisão (D3, D10) para que ingressassem em um ECOS móvel - *"A qualidade do trabalho depende muito disso, e ter acesso a ferramentas de construção de aplicações é essencial para você poder escrever seu código mais rápido, mais eficiente, com maior escalabilidade"* (D10). Contudo, alguns informaram que inicialmente têm dificuldade em lidar com os recursos (D11, D16)

disponibilizados na plataforma, levando algum tempo para se habituarem a estes recursos - *"No início eu não tinha tanta noção desses recursos. Eu não conhecia ferramentas, eu não conhecia bibliotecas e esses tipos de coisas"* (D16). Porém, para continuarem participando, 17 desenvolvedores consideraram importante os recursos técnicos, sendo que o desenvolvedor que tem participado de um ECOS móvel há mais tempo entre os entrevistados reconheceu que os recursos técnicos têm melhorado consideravelmente ao longo dos últimos anos - *"Toda solução e documentação da plataforma que eu desenvolvo está em contínua atualização, a documentação está sempre atualizada e eles estão sempre adicionando novas soluções para os problemas que a gente acaba identificando"* (D4). Por fim, a carência dos recursos necessários para o desenvolvedor tende a comprometer o seu desempenho (D5, D8, D17), gerando atrasos e fazendo com que o desenvolvedor tenha que pensar em outras formas de lidar com essas deficiências - *"Sempre quando a documentação ou recursos é inexistente, o trabalho acaba sendo prejudicado, por isso que às vezes ocorrem atrasos e bugs"* (D8).

(F2) Desempenho do hardware das aplicações

Inicialmente, os desenvolvedores costumam não prestar muita atenção nas questões de desempenho, sendo essas apontadas como importante ao se iniciar a participação, segundo a opinião de 8 dos entrevistados. Para continuar participando, 18 dos 20 entrevistados reforçam a importância deste fator. O desempenho do hardware dos dispositivos, algumas vezes, torna-se uma preocupação, pois os desenvolvedores precisam lidar com uma grande variação de confi (D1, D2, D4, D9, D16, D19) dos usuários finais, a qual é cada vez maior. Entre as razões, fabricantes e modelos novos surgem seguidamente e o desenvolvedor precisa tomar certas precauções para que suas aplicações funcionem nestes diferentes dispositivos - *"Eu acho que era mais nivelado quando eu comecei. No início, iPhone e alguns dispositivos Android tinham sua performance praticamente a mesma e hoje se tem desde celulares muito potentes até celulares com pouquíssimos recursos [de hardware]"* (D1). Contudo, quando conseguem lidar com estes problemas, os desenvolvedores aumentam suas oportunidades de mercado (D3, D8), uma vez que um desenvolvedor consegue alcançar públicos maiores dentro da mesma plataforma a partir de adaptações em sua aplicação - *"Quantos mais dispositivos executarem a sua aplicação, mais pessoas vão conseguir utilizar. Deve-se levar em conta que o cliente consiga executar a aplicação no hardware que ele possuir, para que ele não precise comprar outro dispositivo apenas para essa aplicação"* (D3).

(F3) Plataforma facilmente configurável

A configuração fácil da plataforma foi considerada importante por 12 desenvolvedores para que comecem a participar em um MSECO. Porém, o desenvolvedor apresenta dificuldades ao começar (D4, D6, D8, D19) logo que escolhe uma plataforma, por possuir o conhecimento necessário para configurar o ambiente de desenvolvimento de forma que se torne mais agradável e produtivo possível - *"Eu adquiri o conhecimento de como configurar uma plataforma, então não é algo que eu julgaria tão importante pra mim, mas mais pelo fato de que eu já passei por isso mesmo"* (D19). Ainda, 14 desenvolvedores consideraram a importância da plataforma ser simples (D2, D3, D16) para continuarem suas participações em um ECOS móvel, tanto quanto a configuração - *"Quanto mais fácil seja a plataforma desde o início quando você esta desenvolvimento de aplicações, melhor"* (D3).

(F4) Padrões de interface e aparência das aplicações

Inicialmente, a importância dos padrões de interface é reconhecida por 14 dos 20 desenvolvedores entrevistados, sendo que alguns inicialmente focavam na funcionalidade e davam menos das aplicações - *"Atualmente a gente se preocupa mais com isso, antigamente não era uma preocupação nossa, a preocupação era a funcionalidade, existiam poucas aplicações"* (D7). Além disso, 17 dos 20 desenvolvedores consideram isso importante, e denotam maior foco na aparência das aplicações.

Em uma perspectiva técnica, os recursos técnicos oferecidos pela plataforma são bem recebidos pelo desenvolvedor (D4, D5, D6), pois, facilitam suas atividades quanto à composição de uma interface para suas aplicações, fazendo com que ele tenha mais tempo para outras atividades - *"Você não precisar refazer código, já tendo os recursos de interface fica muito mais fácil e simples, desde que esses recursos de interface sejam bons para o cliente, e que se consiga utilizá-los de uma maneira fácil"* (D6). Além disso, a existência de padrões de interface oferecidos pela plataforma auxilia o desenvolvedor (D3, D4) a atingir os padrões recomendados pela plataforma, fazendo com que suas aplicações sejam mais padronizadas para os usuários finais e dentro do que se utiliza no mercado - *"Deixar nos padrões, para ficar uma aparência bonita, uma aplicação que vai chamar a atenção e que vai parecer profissional"* (D3).

Além disso, o desenvolvedor deseja agradar os usuários (D2, D11, D16), pois quanto mais agradável for uma aplicação, mais ela vai atrair ou manter os usuários utilizando-na, e potencialmente mais clientes dos serviços embutidos na aplicação surgem - *"A gente desenvolve aplicações pensando no cliente, tem que ser uma experiência agradável"* (D16). Além disso, a interface de uma aplicação pode ser um diferencial dos concorrentes no mercado (D1, D9), uma vez que mesmo respeitando os padrões e utilizando os recursos oferecidos pela plataforma, o desenvolvedor pode utilizar de sua criatividade e otimizar a experiência de seus usuários - *"Isso tornou-se uma forma de diferenciar sua aplicação de outras aplicações, implementando coisas diferentes, mas acredito também que deve-se levar em consideração as questões de usabilidade, não inventando algo completamente fora dos padrões"* (D1).

(F5) Segurança dos dados

A segurança é considerada um fator importante para se começar em um ECOS móvel, segundo a opinião de 15 dos desenvolvedores entrevistados, além de ser importante para continuar, conforme 18 desenvolvedores, o que demonstra a importância deste fator desde o início, para a rotina e para a experiência dos desenvolvedores. A segurança dos dados é uma questão técnica fundamental para os desenvolvedores por se tratar de uma grande preocupação, e, por sua vez, os desenvolvedores reconhecem a importância (D1, D11, D18) de terem esse cuidado especial com a segurança de suas aplicações - *"Segurança é uma coisa essencial, por mais simples que sejam os dados que você esteja lidando, você sempre pode fazer alguma coisa"* (D18). Isso pois os desenvolvedores lidam em suas aplicações com dados sensíveis de seus usuários (D4, D5, D9, D19), que não podem ser expostos ao público - *"Deve-se assegurar os dados do próprio cliente, você tem que mostrar para o cliente que criou uma aplicação segura, que ele pode usá-la sem nenhum risco (os dados serem roubados, por exemplo)"* (D5). Além disso, os desenvolvedores possuem certo receio da responsabilidade pela segurança (D3, D8, D15) destes

dados tão importantes aos seus usuários. Contudo, dificilmente eles investigam a fundo a segurança, mas confiam nos recursos da plataforma selecionada - *"Seria mais o caso de login e senha a nossa segurança e eu nunca fui atrás pra ter, tipo, uma transmissão criptografada desses dados"* (D3).

(F6) Acessibilidade e suporte a diferentes perfis de usuários

A acessibilidade e o suporte a diferentes perfis de usuário são considerados menos importantes quando um desenvolvedor começa a participar de um ECOS móvel (apenas 4 desenvolvedores apontaram). Apesar de se tornarem mais importantes ao longo do tempo, essas questões continuam sendo pouco trabalhadas, consideradas importantes para 10 dos desenvolvedores entrevistados quando se pensa em continuar participando de um ECOS móvel. A realidade é que alguns desenvolvedores não sentem necessidade (D1, D3, D7), por nunca terem que criar uma aplicação com tais características - *"Nunca pensei assim, em desenvolver uma aplicação tão modal"* (D1).

Todavia, em seus projetos de aplicações, os desenvolvedores costumam possuir um público-alvo estipulado (D6, D14, D15) pelos seus superiores hierárquicos nas empresas ou organizações que trabalham - *"Geralmente as aplicações que a gente faz têm nicho específico. Temos que desenvolver e ir pensando neste público-alvo"* (D15). Contudo, mesmo dentro de públicos-alvo bem estipulados, um desenvolvedor deseja alcançar o máximo de clientes (D5, D11, D16) possível, pois quer atender o máximo de usuários naquele nicho, atraindo assim possíveis investimentos, inclusive em nichos mais específicos como as necessidades de acessibilidade - *"Todo mundo tem que ser capaz de usar a aplicação. O alcance de mais pessoas possível, mas é difícil saber como começar"* (D16).

4. Discussão

Para um desenvolvedor de tecnologias móveis, em alguns casos, os recursos técnicos (F1) da plataforma se tornam critério de decisão na escolha de um ECOS. Da mesma forma, o desenvolvedor percebe a importância de conseguir facilmente configurar a plataforma (F3), da segurança dos dados trafegados por suas aplicações (F5), entre outros fatores que não apenas o aproxima de um ECOS móvel como o mantém nele. Todavia, alguns fatores são considerados de menor importância pelos desenvolvedores como, por exemplo, a acessibilidade, conforme apontado por alguns entrevistados, visto que muitos projetos possuem público-alvo restrito e específico.

Existem também alguns desafios os quais um desenvolvedor deve se confrontar. Inicialmente surgem problemas, como adaptar-se aos recursos da plataforma (F1), que muitas vezes são desconhecidos, e aprender a configurar e utilizar a plataforma (F3), o que acaba tomando, no começo da jornada, um considerável tempo. Além disso, outros desafios que surgem são quando faltam recursos técnicos (F1), fazendo com que eles gastem tempo em busca de alternativas, ou quando o mesmo precisa lidar com a variação do hardware dos usuários (F2), bem como a responsabilidade com os dados sensíveis do usuário (F5) – além de problemas técnicos que podem gerar implicações legais.

O foco do desenvolvedor também acaba se alterando com o tempo. Com relação a padrões e interfaces (F4), alguns desenvolvedores afirmaram que tinham foco apenas na funcionalidade quando começavam suas atividades na plataforma. Contudo, com relação à acessibilidade e suporte a diferentes perfis de usuários (F6), alguns desenvolvedores

afirmam que seu foco está relacionado ao público-alvo que cada aplicação deve atender, o que demonstra a mudanças de foco ao longo de suas experiências.

O suporte que a plataforma presta a um desenvolvedor costuma auxiliá-lo em suas atividades, e os desenvolvedores entrevistados reconhecem que os recursos técnicos (F1) têm melhorado muito ultimamente. Quanto a padrões de interface (F4), eles também se agradam da existência de padrões e de recursos pré-elaborados para poderem modelar o *design* de suas aplicações.

Os desenvolvedores desejam alcançar mais usuários quanto possível e observam oportunidades de mercado. Por exemplo, desenvolvimento de aplicações de forma que funcionem no máximo de dispositivos possíveis (F2); e elaboração de interfaces mais elegantes que atraiam os usuários (F4) e atendam demandas de usuários com necessidades especiais (F6), para assim atingir o máximo de clientes possível. Os desenvolvedores costumam também prestar atenção nos usuários, tentando agradá-los como, por exemplo, com usabilidade de interface mais amigável (F4), ou demonstrando que o usuário pode confiar seus dados sensíveis às suas aplicações (F5).

5. Limitações e Dificuldades

Como todo estudo experimental, algumas limitações e dificuldades foram enfrentadas, entre elas: i) A análise dos dados utilizou o método de *card sorting*, muito recomendado em caso de agrupamentos. Porém, como a quantidade da dados coletados era considerável, demandando um tempo considerável para sua análise; ii) A heterogeneidade dos entrevistados não foi alta, uma vez que os desenvolvedores convidados para as entrevistas são todos brasileiros. Faz-se necessário um estudo com desenvolvedores de diversos países para que se tenha uma visão mais geral, de modo a confirmar os resultados obtidos; iii) A opinião dos desenvolvedores sofre uma maturação com o passar do tempo e de suas experiências, fazendo com que mesmo o mais experiente dos desenvolvedores entrevistados possa, em algum momento, ter novas conclusões sobre estes fatores; iv) A literatura explorada foi a de ECOS móvel. Entretanto, a literatura mais geral da área de ECOS como um todo pode apresentar mais fatores técnicos que não foram apreciados neste estudo.

6. Conclusão

A partir deste estudo, foi possível constatar que os fatores técnicos podem implicar na decisão de um desenvolvedor para participar ou se manter participando de um ECOS móvel. Alguns fatores são melhor entendidos pelos desenvolvedores – e.g., Desempenho do hardware das aplicações (F2), Padrões de interface e aparência das aplicações (F4) e Segurança dos dados (F5) –, e outros menos – e.g., Acessibilidade e suporte a diferentes perfis de usuários (F6) –, pois, pela experiência dos desenvolvedores, houve mais situações que precisassem recorrer a alguns dos fatores mais do que a outros.

Esta pesquisa também contribuiu para esclarecer algumas lacunas identificadas na literatura [de Souza et al. 2016], explorando a dimensão técnica de ECOS de forma a complementar o entendimento de elementos da dimensão social. Além disso, com a participação de desenvolvedores nas entrevistas, pode-se verificar como os fatores técnicos identificados no mapeamento sistemático da literatura afetam suas decisões. Além disso, este estudo pode auxiliar desenvolvedores novatos a escolherem um MSECO

a partir do entendimento de quais fatores são importantes para os desenvolvedores que já participam a mais tempo em um ecossistema.

Também, esta pesquisa abre possibilidades para trabalhos futuros. Por exemplo, sugere-se investigar em literatura correlata se existem outros fatores técnicos não apreciados neste estudo. Outra possibilidade é a realização de um estudo quantitativo, de forma global, a fim de consultar desenvolvedores de ECOS móvel e buscar a generalização dos resultados obtidos.

Agradecimentos

Agradecemos ao CNPq (processos 420801/2016-2 e 311256/2018-0) pelo apoio financeiro e a PUCRS pelo subsídio aos assistentes de pesquisa da graduação (Edital BPA-PRAIAS 2018). Cleidson de Souza agradece o apoio financeiro da PROPESP/UFPA.

Referências

Borges, C. S. et al. (2019). Fatores que influenciam desenvolvedores a participar de ecossistemas de software movel. Master's thesis, Pontifícia Universidade Católica do Rio Grande do Sul.

Bosch, J. and 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. and Ahmed, F. (2010). A three-dimensional view of software ecosystems. In *Proceedings of the European Conference on Software Architecture: Companion Volume*, pages 81–84, Copenhagen, Denmark. ACM.

de Souza, C. R., Figueira Filho, F., Miranda, M., Ferreira, R. P., Treude, C., and Singer, L. (2016). The social side of software platform ecosystems. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 3204–3214, San Jose, California. ACM.

Dhungana, D., Groher, I., Schludermann, E., and Biffl, S. (2010). Software ecosystems vs. natural ecosystems: learning from the ingenious mind of nature. In *Proceedings of the European Conference on Software Architecture: Companion Volume*, pages 96–102, Copenhagen, Denmark. ACM.

Fontao, A. d. L., dos Santos, R. P., and Dias-Neto, A. C. (2015). Mobile software ecosystem (mseco): a systematic mapping study. In *Proceedings of the IEEE Annual Computer Software and Applications Conference*, pages 653–658, Taichung, Taiwan. IEEE.

Mallinson, K. (2015). Smartphone revolution: Technology patenting and licensing fosters innovation, market entry, and exceptional growth. *IEEE Consumer Electronics Magazine*, 4(2):60–66.

Singer, J., Sim, S. E., and Lethbridge, T. C. (2008). Software engineering data collection for field studies. In Shull, F., Singer, J., and Sjoberg, D. I., editors, *Guide to Advanced Empirical Software Engineering*, chapter 1, pages 9–34. Springer, London, UK.

Spencer, D. (2009). *Card sorting: Designing usable categories*. Rosenfeld Media, New York, USA.

Uma Investigação da Aplicação de Aprendizado de Máquina para Detecção de Smells Arquiteturais

Warteruzannan Soyer Cunha[1], **Valter Vieira de Camargo**[1]

[1]Departamento de Computação – Universidade Federal de São Carlos (UFSCar)
Rodovia Washington Luis, km 235 – 13565-905 - São Carlos – SP – Brazil

{warteruzannan.cunha,valter}@ufscar.br

Resumo. *Uma investigação da aplicação de aprendizado de máquina para detectar os smells arquiteturais God Component (GC) e Unstable Dependency (UD) é apresentada neste trabalho. Dois datasets foram criados com exemplos coletados de sistemas reais. A acurácia, precisão e recall foram avaliadas com um conjunto de 10 algoritmos preditivos. Uma seleção de atributos foi realizada a fim de encontrar os mais relevantes para essa detecção. Os algoritmos AdaBoost e SVM (Support Vector Machine) com kernel linear alcançaram os melhores resultados para o GC e UD, respectivamente. Além disso, observou-se que alguns atributos que a princípio não seriam considerados, contribuíram para a precisão da detecção.*

1. Introdução

Embora o conceito de *smells* tenha surgido originalmente no contexto de elementos de código-fonte com baixo nível de abstração (classes, métodos, entre outros), eles também ocorrem em níveis mais altos, como na arquitetura de sistemas. O termo *smell* arquitetural foi cunhado por [Lippert and Roock 2006] para descrever a combinação de decisões equivocadas em nível arquitetural que resultam na deterioração do software, reduzindo sua qualidade. [Garcia et al. 2009] relatam que *smells* desse tipo demandam mais esforço para serem identificados e corrigidos quando comparados a outros de nível mais baixo de abstração. Exemplos de dois *smells* reportados na literatura são o *Unstable Dependency* e o *God Component*. Considerando uma definição direta e simples, pode-se dizer que o UD ocorre quando um componente A depende de um outro B que é menos estável que o A; e o GC ocorre quando um componente tem um elevado número de classes, pacotes ou número de linhas e/ou implementa mais de uma funcionalidade.

Assim, a detecção de *smells* arquiteturais, principalmente com apoio automatizado, é uma é uma temática de significativa relevância, tendo em vista que problemas existentes em nível arquitetural podem ter impactos severos na qualidade do sistema e elevar os custos de manutenção. Portanto, pesquisas têm sido conduzidas a fim de identificar esse tipo de *smell*, apoiando-se em métricas [Fontana et al. 2017], regras definidas manualmente [Velasco-Elizondo et al. 2018] e histórico de versões [Palomba et al. 2013].

Neste trabalho, defende-se a ideia de que a caracterização de *smells* possui um nível de subjetividade envolvido e deve ser realizada a partir de diferentes pontos de vista, seja pessoas ou abordagens. Isto é, caracterizar algo como *smell* depende do contexto e experiência dos engenheiros de software que conhecem o sistema. Alguns autores relatam que trabalhos que fazem o uso de métricas e/ou regras que estabelecem thresholds para a detecção de *smells* (em qualquer nível de abstração) são propensos a erros e podem

apresentar uma quantidade elevada de falsos positivos/negativos, além de não levar em conta a subjetividade dessa tarefa [Wang et al. 2018].

Visto isso, a utilização de aprendizado de máquina (AM), uma subárea da Inteligência Artificial (IA), tem sido explorada por alguns pesquisadores a fim de reduzir a quantidade de erros, melhorar a acurácia na identificação de *code smells* e incluir, de alguma forma, uma análise subjetiva sobre essa decisão. Com isso, o emprego de AM consiste em coletar exemplos de diferentes pontos de vista, generalizar esse conhecimento e classificar entradas desconhecidas [Wang et al. 2018, Moha et al. 2010]. Contudo, poucos pesquisadores têm explorado a utilização dessa técnica na detecção de *smells* arquiteturais. Até o presente momento, trabalhos ou ferramentas que identificam o UD (*Unstable Dependency*) e GC (*God Component*) utilizando AM não foram encontrados, bem como estudos que relatam quais os melhores algoritmos nesse contexto. Além disso, não foram encontrados relatos de quais seriam os atributos mais relevantes para a detecção desses *smells* utilizando AM.

Portanto, o presente trabalho busca investigar o emprego de AM na identificação de dois *smells* (GC e/ou UD). As questões de pesquisa que norteiam este trabalho são:

#QP1: Há indícios de que o aprendizado de máquina pode melhorar a acurácia da identificação dos *smells* abordados?

#QP2: Quais são os atributos mais relevantes para identificação dos *smells* abordados neste trabalho?

Como resultados, observou-se que a utilização de AM ofereceu bons resultados. O *AdaBoost* foi o melhor algoritmo para GC e, paralelamente, o SVM (*Support Vector Machine*) com kernel linear para o segundo para o UD, com acurácia de 99,17% e 99,41%, respectivamente. Além disso, foi observado que atributos (Figura 4) que não foram estudados anteriormente contribuem para essa identificação.

2. Fundamentação Teórica

Unstable Dependency: Esse *smell* ocorre quando um elemento arquitetural depende de outro menos estável que ele mesmo [Fontana et al. 2017]. Isso prejudica a manutenção do sistema, uma vez que componentes instáveis são mais suscetíveis a manutenções e erros. Geralmente, a identificação por métricas compara os valores da instabilidade de dois componentes. Entretanto, os resultados podem variar, uma vez que os engenheiros de software podem ter opiniões divergentes sobre instabilidade. Portanto, é necessário detectar esse *smell* sobre diferentes pontos de vista. O conceito de instabilidade é melhor discutido por [Martin 1994].

Too Large Packages/Subsystems (God Component): Analogamente ao *smell God Class*, *God Component* surge quando um elemento arquitetural é muito grande, tem muitas classes, interfaces, arquivos, linhas de código e/ou sub-elementos (pacotes) e/ou duas ou mais funcionalidade estão sendo implementadas, o que prejudica a coesão, manutenção, testabilidade e evolução do software [Lippert and Roock 2006].

Aprendizado de Máquina: O Aprendizado de Máquina (AM) é definido como uma subárea da Inteligência Artificial (IA), cujo o objetivo de criar e melhorar técnicas para a construção de sistemas capazes de tomar decisões baseadas em experiências acumuladas. Assim, os algoritmos de AM conseguem aprender com exemplos e deduzir uma

função (também conhecida como modelo, função objetivo, hipótese, classificador, entre outras) capaz de analisar entradas desconhecidas e rotulá-las de acordo com o conjunto esperado. No geral, o rótulo é atribuído com base nas características de cada instância [Mitchell 1997]. O conjunto de exemplos é conhecido como base de dados ou *dataset* e o rótulo como classe. Neste trabalho as funções deduzidas são chamadas de classificadores.

Validação cruzada: A validação cruzada é uma técnica de AM que consiste em dividir o *dataset* em *k* subconjuntos com a mesma quantidade de exemplos, de forma que um desses é utilizado para teste e o restante para treinamento. Assim, o subconjunto *k-1* é utilizado para teste, depois o *k-2*, e assim sucessivamente, até que todos os exemplos sejam testados [Kohavi et al. 1995].

A Equação 1 ilustra o cálculo da acurácia, na qual *tp* (*true positive*) são os verdadeiros positivos, *tn* (*true negative*) os verdadeiros negativos, *fn* (*false negative*) falso negativos e *fp* (*false positive*) os falsos positivos. A acurácia oferece, no geral, o quanto o classificador está classificando corretamente entradas desconhecidas. Na Equação 2 é apresentado o cálculo da precisão, que permite conhecer o quanto, dos exemplos classificados como verdadeiros, realmente são. Na Equação 3 o cálculo do recall, que permite saber qual a frequência com que exemplos de uma determinada classe são classificados corretamente. A médias dessas métricas podem ser mensuradas cada vez que o classificador é treinado e testado

$$(1)\ Accuracy = \frac{tp + tn}{tp + tn + fn + fp} \qquad (2)\ Precision = \frac{tp}{tp + fp} \qquad (3)\ Recall = \frac{tp}{tp + fn}$$

3. Metodologia

A metodologia para desenvolvimento deste trabalho consistiu nas seis atividades descritas abaixo.

#1 Selecionar Sistemas: Foram selecionados sistemas que atendiam aos seguintes critérios: i) ser desenvolvido em Java [1]; e ii) ser de código aberto; e iii) pertencer a uma das seguintes categorias: mobile, ferramentas, plugins, bibliotecas/utilitários, frameworks, *API* e *databases*. Esse último critério foi aplicado a fim de coletar exemplos de *smells* em tipos variados de sistemas. Foram selecionados 5 sistemas mobile, 9 ferramentas, 4 plugins, 23 bibliotecas/utilitários, 6 frameworks, 3 *apis*, 3 *databases* e 1 *middleware*. A busca foi realiza em repositórios do github e/ou repositórios próprios. Ao final, 52 sistemas foram selecionados. A lista completa pode ser encontrada no link https://gist.github.com/papersfiles/a9a4035fb4acdf1ef0b80459941a2fd.

#2 Definir atributos dos Datasets: Consistiu na escolha de um conjunto de métricas que foram utilizadas como os atributos dos *datasets*. A escolha seguiu os seguintes critérios: C1) Métricas utilizadas por outros autores na detecção de *smells* arquiteturais [Fontana et al. 2017, Martin 1994, Kumar and Sureka 2018, Sharma 2016]; e C2) Métricas que caracterizam complexidade de código, como coesão, acoplamento e tamanho. As métricas do segundo critério foram adicionadas justamente para averiguar se elas também possuem alguma influência na decisão. O conjunto escolhido pode ser visualizado no link https://github.com/papersfiles/artigo-vem/blob/master/metrics.

[1] https://www.java.com/en/download/

#3 Extrair valores das métricas escolhidas dos componentes de sistemas selecionados: Consistiu em analisar todos componentes dos sistemas selecionados e extrair os valores das métricas escolhidas anteriormente. Assim, os valores extraídos foram colocados em colunas em dois *datasets*. Para o GC, cada linha (exemplo) do *dataset* representa um componente ao passo que, para o UD, cada linda representa uma dependência entre dois componente. Portanto, a quantidade de atributos previsores do UD é duas vezes maior que a do GC, já que dois componentes são representados em cada linha.

#4 Rotular exemplos dos *datasets*: Consistiu em atribuir rótulos 0 (não é smell) ou 1 (é smell) para cada componente (no caso do GC) ou dependência (no caso do UD). Nesta etapa, a atribuição dos rótulos foi feita usando abordagens existentes. Cada entrada do *dataset* foi avaliada por mais de uma abordagem. Logo, o rótulo foi atribuído de acordo com a maioria das abordagens. Por exemplo, se três relataram a ocorrência do *smell* e duas que não, o rótulo foi 1.

#5 Balancear classes e normalizar os dados: Consistiu em balancear a quantidade de exemplos positivos e negativos para cada *smell* e normalizar[2] a escala dos valores extraídos. Isso evita que os classificadores considerem atributos com valores elevados mais importantes, o que prejudica o aprendizado.

#6 Treinar e avaliar classificadores: Concentrou-se em: i) Escolher um conjunto de classificadores; ii) Separar cada *dataset* em duas partes, uma para treinamento e outra para teste; iii) Treinar os classificadores com os dados de treino e avaliá-los de acordo com a acurácia, precisão e recall com os dados de teste. Além disso, avaliar estatisticamente quais alcançaram os melhores resultados. Utilizou-se os seguintes classificadores: *Naïve Bayes, Decision Tree, Random Forest, Knn, AdaBoost, Support Vector Machine, Redes Neurais Artificiais, XGBoost, Bagging e Dummy Classifier*.

4. Resultados e Discussões

Os parágrafos abaixo são voltados a responder as questões de pesquisa elencadas para este trabalho. Para facilitar o entendimento, segue uma explicação sobre o processo de teste simples: Uma vez que todos os exemplos dos *datasets* já foram rotulados com abordagens existentes, isto é, sob diferentes pontos de vista, parte dos dados é separada para treinamento dos classificadores e outra para teste. Logo, os classificadores são treinados somente com os dados destinados a treino, assim, não conhecem o rótulo de exemplos presentes na porção de teste. Em seguida, cada exemplo em teste é submetido ao classificador já treinado e um rótulo é retornado por esse. Com isso, é possível verificar se o classificador acertou ou não, já que rótulo dessa entrada já é conhecido. Esse processo possibilita calcular métricas como acurácia, precisão e recall. Vale ressaltar que os testes foram realizados utilizando uma técnica de validação cruzada com k=10 repetida 30 vezes. Essa técnica é amplamente aceita pela comunidade de AM e é melhor explicada na seção 2.

#QP1: Há indícios de que o aprendizado de máquina pode melhorar a acurácia quanto à identificação dos *smells* abordados? Na Tabela 1 (a) podem ser visualizadas a média da acurácia (coluna 2), precisão (coluna 3) e recall (coluna 4) dos testes realizados com o *dataset* do UD, de acordo com cada classificador (coluna 1). O

[2]https://scikit-learn.org/stable/modules/preprocessing.html

XGBoost alcançou 98,97% de precisão, o *SVM com kernel linear* alcançou 99,88% de recall e 99,41% de acurácia. Visto que o *dataset* foi oi rotulado com base em diferentes abordagens existentes, os classificadores conseguiram aprender com esses exemplos e fornecer bons resultados, alguns com acurácia acima de 99%. Entretanto, a métrica (precisão, recall, etc) que mede a eficiência do classificador deve ser escolhida com cuidado. Por exemplo, o classificador *KNN* obteve 92,23% de acurácia, muito próximo ao SVM - *Polynomial*. Entretanto, a diferença da precisão é significativa, portanto, o classificador escolhido deve ter um equilíbrio entre essas métricas.

De forma similar, os resultados para o *dataset* do GC podem ser vistos na na Tabela 1 (b). O AdaBoost alcançou a melhor acurácia e precisão, respectivamente, 99,17% e 98,84%. Por outro lado, SVM com kernel linear apresentou recall de 100%. Notou-se que alguns têm boa acurácia e precisão, porém baixo recall, a exemplo, SVM - Polynomial e Sigmoid. Isso indica que mesmo que a acurácia seja boa, a taxa de erro na identificação das classes positivas e negativa é alta. O *dataset* construído para o GC permite identificar, com alta precisão, componentes com alto custo de manutenção e direcionar os esforços de manutenção. Além disso, esses dados podem ser estendidos para atribuir a grau de severidade do *smell* encontrado; um algoritmo de regressão pode ser utilizado.

Tabela 1 (a) - Acurácia, precisão e revocação do *unstable dependency*

Algoritmo	Acurácia	Precisão	Revocação
Knn	0.9223	0.8685	0.9954
Random Forest - entropy	0.9507	0.9243	0.9819
Random Forest - gini	0.9517	0.9279	0.9795
Redes Neurais Artificiais	0.9908	0.9849	0.9969
Naive Bayes	0.6947	0.6277	0.9569
SVM - Linear	0.9941	0.9896	0.9988
SVM - Polynomial	0.9399	0.9189	0.9651
SVM - Sigmoid	0.8671	0.8543	0.8853
SVM - RBF	0.9785	0.9623	0.9959
AdaBoost	0.9904	0.9854	0.9955
Decision tree - entropy	0.9867	0.9782	0.9955
Decision Tree - gini	0.9824	0.9769	0.9882
XGBoost	0.9938	0.9897	0.9980
Bagging	0.9912	0.9896	0.9929
Dummy - stratified	0.4972	0.4973	0.4978

Tabela 1 (b) - Acurácia, precisão e revocação do *god component*

Algoritmo	Acurácia	Precisão	Revocação
Knn	0.9512	0.9118	0.9991
Random Forest - entropy	0.9795	0.9640	0.9962
Random Forest - gini	0.9772	0.9597	0.9961
Redes Neurais Artificiais	0.9879	0.9767	0.9996
Naive Bayes	0.8416	0.9282	0.7408
SVM - Linear	0.9826	0.9664	1.0000
SVM - Polynomial	0.8973	0.9750	0.8157
SVM - Sigmoid	0.8438	0.8321	0.8615
SVM - RBF	0.9675	0.9458	0.9918
AdaBoost	0.9917	0.9884	0.9951
Decision tree - entropy	0.9878	0.9868	0.9888
Decision Tree - gini	0.9707	0.9586	0.9841
XGBoost	0.9899	0.9849	0.9952
Bagging	0.9836	0.9751	0.9927
Dummy - stratified	0.5007	0.5009	0.4987

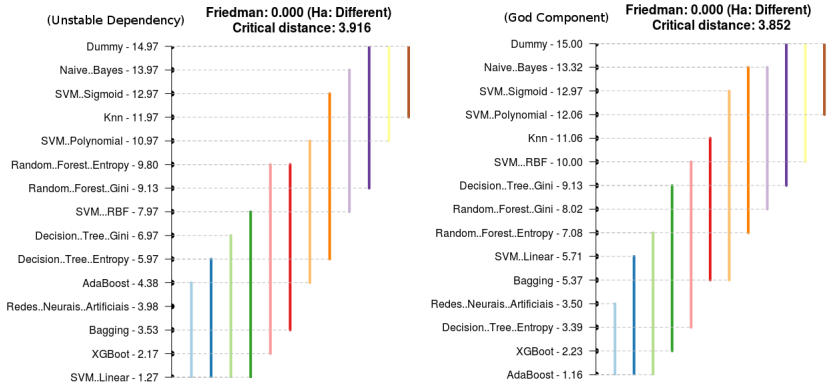
Para complementar a análise, o teste de Friedman foi aplicado a fim de verificar se os classificadores apresentam resultados estatisticamente iguais. O resultado foi igual a 0.00, confirmando que há diferença significativa. Logo, o teste Nemeneiy foi realizado pois permite verificar qual a diferença significativa entre esses resultado e, por conseguinte, descartar a utilização de classificadores que apresentam boa acurácia mas são, estatisticamente, inferiores a outros.

Como pode ser visualizado na Figura 1 (unstable dependency), o valor critico é de 3.916, isto é, caso a diferença entre classificadores fosse menor que esse valor, seriam considerados equivalentes. Por exemplo, a distância entre o SMV - Linear e XGBoost é de 0.9 (1.17 - 1.27), menor que 3.916. Logo, o SVM com kernel linear e o XGBoost podem ser utilizados de forma equivalentes neste contexto.

Para o GC, o SVM com kernel linear alcançou, em média, a acurácia de 98.26%, o que da a falsa sensação de que ele é um dos mais indicados nesse contexto. Entretanto, com o teste de Nemeneiy pode ser verificado que outros como o AdaBoost, XGBoost, *Decicion Tree - Entropy* e Redes Neurais Artificiais apresentam melhores resultados.

#RQP1 Portanto, há indícios de que o aprendizado de máquina pode melhorar a acurácia quanto à identificação dos *smells* abordados no presente trabalho. Uma vez os *datasets* foram construídos com base em outros trabalhos e ferramentas, ou seja, com

Figura 1. Teste de Nemeneiy para com o *Unstable dependency* e *God Component*



base em diferentes visões, os algoritmos de AM conseguiriam generalizar o conhecimento extraído e alcançar bons resultados quanto à acurácia, precisão e recall.

#QP2: Quais são os atributos mais relevantes para identificação dos *smells* abordados neste trabalho?

A seleção de atributos foi realizada utilizando o modelo *Extra Trees Classifier* [3], implementado na biblioteca SckitLearn [4]. Esse modelo permite atribuir a cada atributo um *score* de importância. Vale ressaltar que o nome de cada métrica pode ser visto em https://github.com/papersfiles/artigo-vem/blob/master/metrics juntamente com seu acrônimo, bem como os sufixos *max, min, std, sum, q1* e *q2* significam, respectivamente, máximo, mínimo, desvio padrão, soma, primeiro e segundo quartil. O prefixo *pck2* indica que o valor desse atributo foi extraído do componente aferente.

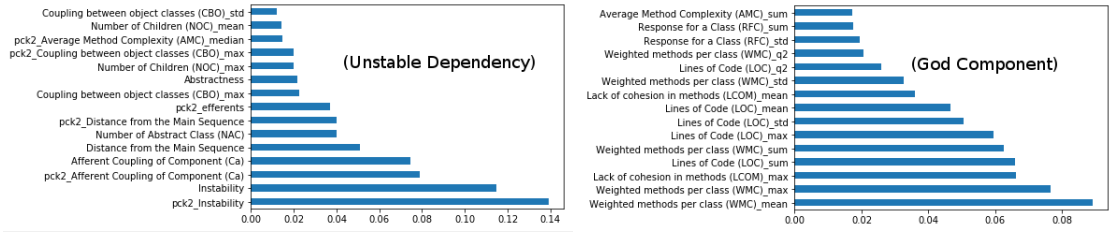
#RQP2: Para facilitar a visualização, apenas os 15 atributos [5] mais importantes para a detecção do *unstable dependency* são apresentados na Figura 2 do lado esquerdo. É importante notar que, atributos que não foram utilizados no processo de rotulação aparecem entre aqueles mais relevantes para a identificação desse *smell*, por exemplo CBO (*Coupling between object classes*) e AMC (*Average Method Complexity*).

Os atributos mais importantes [6] para a detecção do *god component* podem ser visualizados na Figura 2 do lado direito. Destaca-se a importância de alguns que não foram utilizados para a rotular exemplos do *dataset*, por exemplo, RFC (*Response for a Class*). Os três atributos mais relevantes foram *WMC_mean*, *WMC_max* e *LCOM_max*, estão relacionados à complexidade e falta de coesão, assim, é importante que o engenheiro de software fique atento à essas métricas.

Como ameaça à validade, é importante ressaltar que, a fim de remover o viés inserido pela forma de rotulação, os classificadores também foram testados sem a presença do conjunto de atributos utilizados na rotulação, assim, pôde ser verificado se o aprendizado extraído é consequência da forma de rotulação ou os classificadores realmente estavam aprendendo.

[3]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html
[4]https://scikit-learn.org/stable/index.html
[5]https://github.com/papersfiles/artigo-vem/blob/master/features_unstable_dependency.csv
[6]https://github.com/papersfiles/artigo-vem/blob/master/features_god_component.csv

Figura 2. Importância dos Atributos na Detecção dos Smells



5. Trabalhos correlatos

[Wang et al. 2018] apresentam uma estratégia para rotular os exemplos de *code smells*. A abordagem é restrita a *code smells* e não explora a identificação em nível arquitetural. [Garcia et al. 2009] realizaram um estudo para definir a taxonomia e identificação de 04 (quatro) *smells* arquiteturais. Porém, é uma abordagem totalmente manual, que prejudica a acurácia identificação. [Fontana et al. 2016] avaliam um conjunto de algoritmos para a detecção de *code smells*, a saber: *J48, JRip, Random Forest, Naive Bayes, SMO* e *LibSVM*. Entretanto concentra-se em *code smells* e não exploram a detecção em nível arquitetural. [Fontana et al. 2017] apresentam uma ferramenta nomeada *Arcan* com o objetivo de detectar três *smells* arquiteturais, a saber: i) *Cyclic Dependency, Unstable Dependency* e *Hub-Like Dependency (HL)*. A ferramenta não explora o uso de AM para essa tarefa. [Maiga et al. 2012] apresentam uma abordagem que busca identificar *code smells* utilizando aprendizado de máquina Entretanto concentra-se em *code smells* e não explora a identificação em nível arquitetural.

6. Considerações Finais

Este trabalho apresentou uma investigação do emprego de AM para a identificação de dois *smells* arquiteturais: *god component* e *unstable dependency*. Os atributos mais relevantes para a identificação foram relatados, bem como a descrição de classificadores que apresentam bons resultados nesse contexto. Os melhores classificadores para a identificação para o GC e UD foram o AdaBoost e SVM com kernel linear, respectivamente.

A base de dados construída neste trabalho pode servir como apoio para ferramentas que, automaticamente, analisam dependências de terceiros (SDKs, bibliotecas, etc) incorporadas no projeto e relatam ao engenheiro de software sobre uma possível necessidade de refatoração. Além disso, o *dataset* pode ser estendido para identificar outros *smells* que ocorrem na dependência entre elementos arquiteturais, por exemplo o *Hub Like Dependency* [Fontana et al. 2017]. Ademais, este trabalho pode servir de base para estudos que comparem diferentes estratégias de rotulação do UD (ou GC), uma vez que este apoiou-se em abordagens existentes ao invés de desenvolvedores

Como trabalhos futuros, pretende-se avaliar a detecção de outros *smells* arquiteturais como: *ambiguous interfaces, feature concentration* e *unstable interface*. Outra importante contribuição é a utilização de componentes arquiteturais representados como classes, interfaces, entre outros, não restringindo somente a pacotes.

Referências

[Fontana et al. 2016] Fontana, F. A., Mäntylä, M. V., Zanoni, M., and Marino, A. (2016). Comparing and experimenting machine learning techniques for code smell detection.

Empirical Software Engineering, pages 1143–1191.

[Fontana et al. 2017] Fontana, F. A., Pigazzini, I., Roveda, R., Tamburri, D., Zanoni, M., and Di Nitto, E. (2017). Arcan: A tool for architectural smells detection. In *2017 IEEE International Conf. on Software Architecture Workshops, ICSA Workshops 2017, Gothenburg, Sweden, April 5–7*.

[Garcia et al. 2009] Garcia, J., Popescu, D., Edwards, G., and Medvidovic, N. (2009). Toward a catalogue of architectural bad smells. In *International Conference on the Quality of Software Architectures*, pages 146–162. Springer.

[Kohavi et al. 1995] Kohavi, R. et al. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada.

[Kumar and Sureka 2018] Kumar, L. and Sureka, A. (2018). An Empirical Analysis on Web Service Anti-pattern Detection Using a Machine Learning Framework. *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, pages 2–11.

[Lippert and Roock 2006] Lippert, M. and Roock, S. (2006). *Refactoring in large software projects: performing complex restructurings successfully*. John Wiley & Sons.

[Maiga et al. 2012] Maiga, A., Ali, N., Bhattacharya, N., Sabane, A., Gueheneuc, Y.-G., and Aimeur, E. (2012). Smurf: A svm-based incremental anti-pattern detection approach. In *Reverse engineering (WCRE), 2012 19th working conference on*, pages 466–475. IEEE.

[Martin 1994] Martin, R. (1994). Oo design quality metrics. *An analysis of dependencies*, pages 151–170.

[Mitchell 1997] Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition.

[Moha et al. 2010] Moha, N., Gueheneuc, Y. G., Duchien, L., and Meur, A. F. L. (2010). Decor: A method for the specification and detection of code and design smells. *IEEE Transactions on Software Engineering*, pages 20–36.

[Palomba et al. 2013] Palomba, F., Bavota, G., Penta, M. D., Oliveto, R., Lucia, A. D., and Poshyvanyk, D. (2013). Detecting bad smells in source code using change history information. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 268–278.

[Sharma 2016] Sharma, T. (2016). Designite – A Software Design Quality Assessment Tool.

[Velasco-Elizondo et al. 2018] Velasco-Elizondo, P., Castañeda-Calvillo, L., García-Fernandez, A., and Vazquez-Reyes, S. (2018). Towards detecting mvc architectural smells. *Advances in Intelligent Systems and Computing*, 688:251–260. cited By 0.

[Wang et al. 2018] Wang, Y., Hu, S., Yin, L., and Zhou, X. (2018). Using code evolution information to improve the quality of labels in code smell datasets. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*.

O Impacto da Utilização da Ferramenta Pivotal Tracker para Monitoração de Desempenho em Times Ágeis de Desenvolvimento de Software

Dorgival Netto [1 2], Ana de Holanda[1 3], Flavio Neves[1], Cloves Rocha[1], Helena Bastos[1]

[1] Centro de Informática, Universidade Federal de Pernambuco

[2] Instituto Federal de Mato Grosso do Sul, Campus Corumbá

[3] Instituto Federal do Acre

{dpsn2, acah, fsn2, car2, hcab2}@cin.ufpe.br

Abstract. *This research aims to analyze the influence that the performance monitoring tool Pivotal has on the relationship between the members of a development agile team. An ethnographic case study was conducted in a software development company, with the unit of analysis being team members and the project manager. We realize that the tool shares with all information, schedules, and performance of each team member, allowing everyone to follow the steps of the project. Also, the spirit of unity and knowledge sharing brings a sense of trust and satisfaction among the team members.*

Resumo. *Esta pesquisa visa analisar a influência que a ferramenta de monitoração de desempenho Pivotal tem na relação entre os membros de um time ágil de desenvolvimento. Foi realizado um estudo de caso de cunho etnográfico em uma empresa de desenvolvimento de software, tendo como unidade de análise os membros do time e o gerente de projetos. Identificamos que a ferramenta compartilha com todos as informações, cronogramas e desempenho de cada membro da equipe, permitindo que todos possam acompanhar as etapas do projeto. Além disso, o espírito de união e compartilhamento de conhecimento trazem a sensação de confiança e satisfação entre os indivíduos do time.*

1. Introdução

Desenvolvedores de software de projetos que seguem os valores e princípios enunciados no Manifesto Ágil, quando estão devidamente motivados e capacitados, a partir de excelência técnica e design simples, passam a criar valor de negócio e a entregar softwares funcionais a intervalos cada vez mais curtos e regulares (Dingsøyr, 2012).

Para facilitar a implementação de projetos ágeis, pode-se utilizar ferramentas de planejamento e monitoração como, por exemplo, a Pivotal Tracker[1]. A ferramenta exibe uma lista priorizada de estórias que o time está trabalhando, todas as estórias que o *Product Owner* priorizou e, as estórias não priorizadas. A utilização de uma ferramenta de monitoração de desempenho permite o compartilhamento mútuo de informações, anseios, expectativas e desafios enfrentados pelos membros do time.

[1] https://www.pivotaltracker.com/

Portanto, é necessário investigar, em um cenário real, como os desenvolvedores de software se sentem quanto à exposição do seu desempenho para toda a equipe. Assim, nossa pergunta de pesquisa visa investigar "*Como o uso de ferramentas que dão suporte a métodos ágeis para monitoração de desempenho, em projetos de software, podem influenciar na relação entre indivíduos da equipe*".

Diante disso nossa pesquisa teve o objetivo de identificar como os desenvolvedores de software se sentem ao terem seu desempenho monitorado e divulgado para todos da equipe. Na empresa avaliada, verificamos que a ferramenta que dá suporte a metodologia ágil tem influência positiva na relação de confiança e colaboração entre os membros do time.

2. Trabalhos Relacionados

Araújo (2014) visa identificar como gerentes formam equipes de software em organizações públicas, a fim de descobrir os critérios, bem como entender de que modo eles são utilizados. O autor identificou que a maior parte dos critérios para formar equipes de software pertencem à categoria dos fatores individuais e é necessário gerenciar a influência que isso pode causar para a equipe de uma forma geral.

França *et al.* (2013) aborda a motivação como sendo um fator muito importante que precisa ser analisado em uma equipe de desenvolvimento de software. A pesquisa demonstra que a sinergia no trabalho em equipe pode influenciar no desempenho individual e comprometimento. Entretanto, a natureza repetitiva de tarefas, a alta carga de trabalho é vista como um fator que causa a desmotivação entre os membros do time.

Assim, verificamos a importância em pesquisar de que forma os membros da equipe de desenvolvimento de software se relacionam com os outros membros diante da utilização da monitoração do desempenho de suas atividades.

3. Metodologia

3.1. Design do estudo

Utilizamos uma abordagem de pesquisa qualitativa, pois estamos interessados em compreender o problema de pesquisa em um contexto particular (Merriam, 2015). Esta pesquisa possui uma abordagem através do estudo de caso com cunho etnográfico e entrevista semiestruturadas em uma empresa de desenvolvimento de software. Um estudo de caso foi escolhido porque pode ser conceituado como uma descrição e análise intensiva de um indivíduo, grupo, instituição ou comunidade (Merriam, 2015). E tem cunho etnográfico porque procuramos observar o comportamento de um grupo particular de pessoas.

3.2. Amostra dos Participantes

A unidade de análise do estudo é composta por seis desenvolvedores e um gerente de projetos que estão inseridos em, pelo menos, uma equipe e trabalham com métodos ágeis. A amostra pode ser caracterizada como não probabilística, pois a seleção dos elementos da população depende ao menos, em parte, do julgamento do pesquisador e, podemos escolher casos ricos para o estudo (Merriam, 2015).

3.4. Preparação da Coleta

Inicialmente, foram realizadas observações da amostra, procurando acompanhar as atividades diárias dos times para entender a sua cultura e comportamentos no seu ambiente profissional. Entrevistas semiestruturadas foram realizadas, utilizando roteiros especificamente concebidos e compostos de perguntas abertas. Os roteiros foram pré-testados, através de uma entrevista piloto com um dos desenvolvedores da empresa.

Estas experiências ajudaram a refinar os roteiros de entrevista do Gerente de Projetos[2] e dos desenvolvedores[3]. Após a realização das entrevistas, ocorreu a transcrição, codificação e categorização dos dados. Também realizamos entrevistas de retrospectiva para clarificar e complementar as informações identificadas na análise dos dados. Após isso, também foi disponibilizado um diário[4] para que as atividades dos desenvolvedores fossem registradas durante uma semana (segunda-feira a sexta-feira).

3.5. Análise de Dados e Síntese

Na análise dos dados, seguimos diretrizes fornecidas por Strauss e Corbin (2008) para categorizar e sintetizar dados, para construir uma teoria baseada em evidências de como os membros de um time ágil se relacionam com a utilização de ferramentas de monitoração de desempenho. O áudio das entrevistas foi transcrito e o QSR NVIVO[5] foi usado para apoiar o processo de análise. Nós consideramos porções rotuladas de texto usando códigos e em seguida estes códigos foram relacionados dando origem às categorias que foram nomeadas seguindo um método de comparação constante (Strauss e Corbin, 2008). A Figura 1 mostra o processo de criação de uma categoria. Em seguida, as relações entre categorias foram mapeadas, levando a proposições explicativas que sustentam a estória central. Os dados foram discutidos em grupo durante reuniões e inconsistências identificadas foram tratadas tanto nas discussões quanto em explicações complementares dos participantes.

4. Resultados e Discussões

4.1. Descrição do contexto

O estudo foi realizado em uma empresa de software localizada em João Pessoa-PB, especializada no desenvolvimento de soluções para o acolhimento e o processamento de dados através de sistemas sob o conceito de SaaS (*Software as a Service*). Estas soluções são desenvolvidas nas linguagens de programação: Python, Java, C e Go. O processo de desenvolvimento ágil é baseado no Scrum, com times pequenos, reuniões diárias, participação do cliente, utilização de *backlog*, *sprints* com duração de 15 dias.

4.2. Coleta de Dados

As entrevistas foram realizadas com sete participantes nas instalações da empresa. Todas as entrevistas foram gravadas e juntas totalizaram 3:52h de tempo de áudio.

[2] Roteiro de entrevista do Gerente de Projetos: https://bit.ly/2Zu3ptI
[3] Roteiro de entrevista do Desenvolvedor: https://bit.ly/2NzAq5A
[4] Diário: https://bit.ly/33ZNoiF
[5] https://www.qsrinternational.com/nvivo/home

4.3. Relacionando fatores e construindo hipóteses

Em seguida, traçamos as relações entre os fatores que influenciam o relacionamento dos

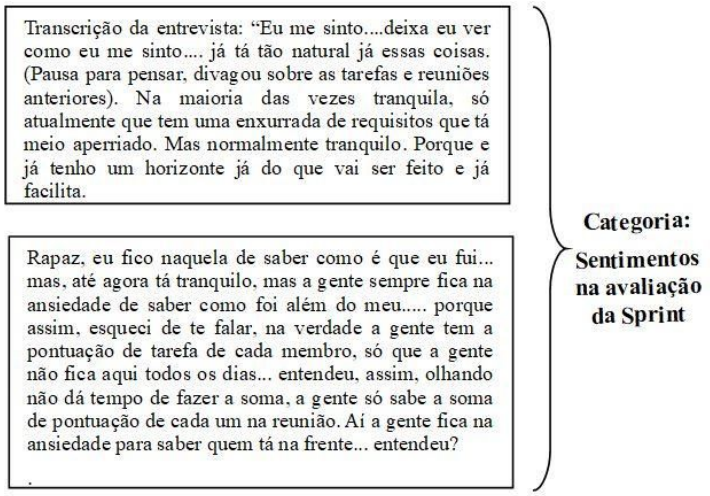


Figura 1. Construção de uma categoria por meio de codificação aberta

membros da equipe com a ferramenta. Nós construímos as proposições, organizando e conectando os elementos mais fortes da teoria, mas também, apresentando uma visão particular do fenômeno na empresa.

Proposição 1: A maneira como o processo ágil da empresa está organizado auxilia na utilização da ferramenta como principal artefato do método ágil.

Proposição 2: A transparência é um dos pilares do método ágil Scrum, o fato da empresa utilizar uma ferramenta que permita aos membros da equipe uma visão geral do projeto e conhecimento do que cada um está fazendo, influencia no bom relacionamento entre os membros da equipe.

Proposição 3: As reuniões de planejamento da *sprint* e as reuniões diárias definem a distribuição das tarefas, o acompanhamento das atividades e o andamento da realização das tarefas. Estes são critérios determinantes para a avaliação de desempenho. O fato da empresa ter uma ferramenta capaz de gerenciar esses critérios faz com que a equipe tenha uma boa satisfação em relação a ferramenta.

Proposição 4: A reunião de avaliação de desempenho causa, aos membros da equipe, tensão antes da reunião e tranquilidade após a reunião.

Proposição 5: Os problemas de especificação prejudicam o desempenho individual e da equipe. Mas, este fator pode ser reduzido devido ao bom relacionamento entre os membros da equipe que gera confiança e aumenta o desempenho.

4.4. Construção da estória central do caso

A Figura 2 apresenta a estória central. Nós usamos retângulos para representar as categorias, paralelogramos para aspectos intangíveis, linhas tracejadas para agrupar as categorias centrais e, setas para representar a forma como os fatores influenciam

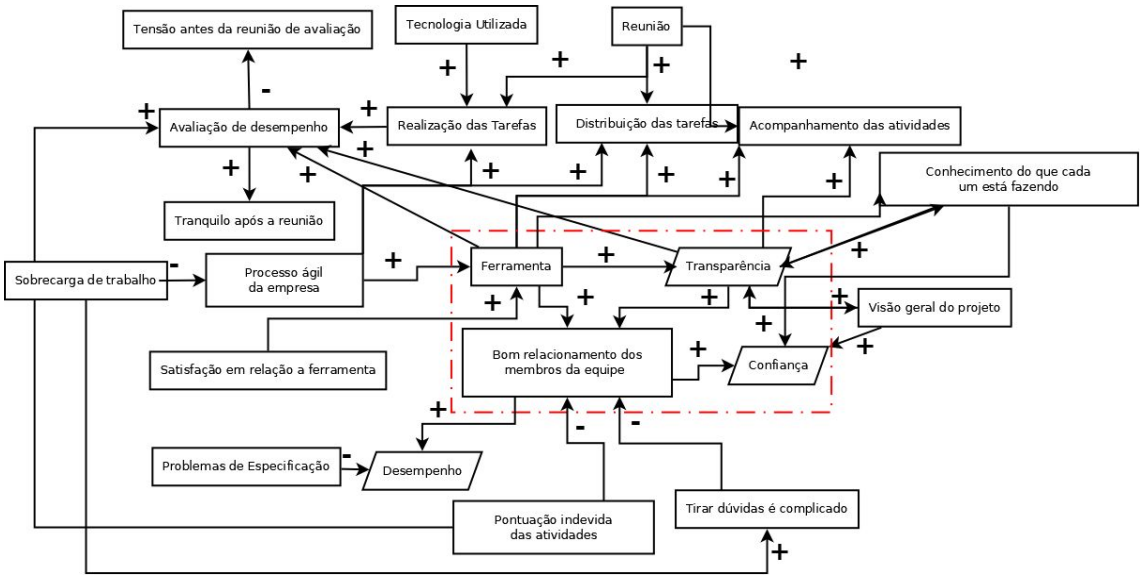


Figura 2. Estória central da pesquisa

(positiva + e negativamente -) o relacionamento entre os indivíduos de um time que utiliza ferramenta para monitoração de desempenho em projetos de software.

A implantação do processo de desenvolvimento de software, baseado nos métodos ágeis, foi fruto da iniciativa dos colaboradores e foi bem aceita pela Diretoria que decidiu patrocinar uma consultoria especializada na implantação de metodologias, adquirir uma ferramenta que suportasse esse processo e realizar o treinamento de todas as equipes no processo e na ferramenta. Como pode ser demonstrado no trecho abaixo:

- "*Não tinha como dar prazo real para o dono da empresa. Hoje dá pra dar um prazo mais real com essa forma de monitoramento. Foi uma evolução grande pra gente. Como eu tô desde o início, era meio solto mesmo*" (Entrevistado 1).
- "*Fomos nós que escolhemos a ferramenta e pedimos para que fosse adquirida. O processo foi elaborado por todos nós. Ficamos muito motivados com a utilização do processo ágil*" (Entrevistado 2).

Após a implantação da metodologia e da adoção da ferramenta Pivotal como meio do gerenciamento e monitoramento das atividades dos projetos, as equipes passaram a utilizá-la durante as suas atividades diárias. Como afirma o Entrevistado 4: "*Todos usam muito bem a ferramenta. Não teve resistência na implantação*".

Assim, após a análise das transcrições das entrevistas percebemos dois cenários.

Primeiro Cenário: Neste cenário percebemos que com as reuniões diárias e os acompanhamentos quinzenais, onde todas as atividades e resultados obtidos eram devidamente registrados na ferramenta, a comunicação do grupo melhorou substancialmente e também aumentou o conhecimento das tecnologias utilizadas.

Como todas as atividades distribuídas para a equipe eram registradas na ferramenta, bem como sua pontuação e acompanhamento, aumentou a transparência do andamento do projeto e do desempenho da equipe, realizados nas reuniões diárias e quinzenais. Possibilitando uma maior sinergia entre os membros da equipe, melhorando o relacionamento, o nível de confiança e tornando-os mais motivados.

As categorias centrais da pesquisa, apresentadas na Figura 2, demonstram que a partir da boa relação da equipe, do acompanhamento através da ferramenta surge a transparência e a confiança necessária para que todos possam agir de forma colaborativa. Relatos das entrevistas podem corroborar com esta afirmação.

- "*É a troca de experiência entre os membros da equipe e com as outras equipes. Estou ganhando experiência com meus companheiros em várias tecnologias. Estou aprendendo muito e é isto o que eu mais gosto*" (Entrevistado 5).
- "*Todo mundo se ajuda mesmo trabalhando em projetos distintos. Conhecemos as mesmas ferramentas. Na outra empresa que eu trabalhava era cada um por si. Ninguém se ajudava. Gosto muito do clima de trabalho*" (Entrevistado 4).
- "*A metodologia faz todo o processo de desenvolvimento ficar muito bem organizado, ninguém fica perdido, existem poucos atrasos*" (Entrevistado 2).

Neste cenário, os membros da equipe veem a ferramenta como um meio que proporciona o acompanhamento das atividades e o conhecimento do que cada um está fazendo. Proporcionando uma visão geral do projeto e contribuindo para o bom relacionamento entre os membros da equipe e aflorando sentimentos como confiança e companheirismo. Aumentando, por sua vez, o desempenho individual e da equipe.

- "*Vejo que fica tudo muito organizado. Todo mundo segue um padrão, reuniões diárias, tudo bem direitinho. A gente vê que a equipe fica bem centrada no projeto e consegue terminar no tempo certo*" (Entrevistado 2).
- "*O quadro e os post-its hoje estão na ferramenta. A ferramenta tem o backlog. Sem a ferramenta não deu certo, os post-it caiam do quadro no chão. Não era prático. Todos esses artefatos estão incluídos na ferramenta*" (Entrevistado 5).
- "*Fico ciente de como foi o desempenho de todos e penso o que eu poderia ter feito melhor em relação às tarefas e já penso nas novas tarefas*"(Entrevistado 1).

Segundo cenário: Neste cenário, os membros da equipe começam a utilizar os conceitos da metodologia ágil como a outra equipe. Entretanto, devido à sobrecarga de trabalho, o processo não está sendo seguido. As reuniões diárias não estão ocorrendo e a distribuição das tarefas não estão sendo registradas na ferramenta.

- "*A ferramenta é muito boa, porém não faz milagre. O trabalho depende mais das pessoas do que das ferramentas*" (Entrevistado 3).
- "*Atualmente não se está seguindo o processo.. As reuniões precisam voltar a acontecer. Existe uma sobrecarga de trabalho o que está prejudicando o trabalho porque nem todos são autogerenciados na equipe. As reuniões de feedback e desempenho são muito importantes para termos as métricas e termos o planejamento adequado*" (Entrevistado 7).

Ao analisar a Figura 2, podemos visualizar que a carga de trabalho, problemas de especificação do projeto, a não realização das reuniões diárias e a falta de comunicação entre os membros da equipe podem influenciar negativamente no bom relacionamento da equipe, consequentemente no sentimento de confiança e desempenho da equipe. Os membros dessa equipe mostraram-se desmotivados com o fato do processo não estar sendo seguido e reclamaram da falta de conhecimento do

planejamento do projeto, desconhecimento do desempenho da equipe e falta de transparência na gestão da equipe.

Cenário comum: Apesar das diferenças identificadas entre as duas equipes entrevistadas, tivemos um ponto em comum. Para ambas, a utilização de metodologia ágil na empresa é primordial para um maior acompanhamento das atividades do projeto de desenvolvimento de software. Também consideram a ferramenta um importante instrumento de monitoração e acompanhamento das atividades da equipe.

- "*Hoje a reunião não está acontecendo como deveria. Quando acontecia a reunião, eu ficava muito tranquilo porque sou muito comprometido. A comunicação é muito rápida e mesmo que eu estivesse com algum atraso, meu gerente já sabia e eu já estava tratando*" (Entrevistado 6).
- "*Atualmente não se está seguindo o processo como deveria ser em todos os projetos. As reuniões precisam voltar a acontecer*" (Entrevistado 7).

Também é consenso entre os dois cenários que a ferramenta realiza com eficiência a monitoração das atividades de cada membro da equipe e que veem de forma positiva a avaliação através de uma pontuação das atividades. Entretanto, o fato da subjetividade da definição do valor a ser pontuado é visto como algo que traz influências negativas e que podem impactar a relação de confiança e união identificados na equipe que está efetivamente utilizando a metodologia ágil em suas atividades. Segue alguns comentários que nos norteiam para essa percepção:

- "*Não tem métrica estabelecida que eu conheça. Não consegui ver os resultados das métricas*" (Entrevistado 7).
- *Nós não temos tido problemas com essa forma de trabalho então fica difícil mudar*" (Entrevistado 3).

Dessa forma, fica claro que é necessário haver uma maior atenção e relação ao estabelecimento de métricas e que estas sejam divulgadas para os membros da equipe, como forma de deixá-los cientes e de motivá-los dentro da empresa. Em contrapartida, o objetivo não é engessar o processo de avaliação dentro da empresa:

- "*Cada projeto e cada equipe tem uma forma de trabalho diferente. Na metodologia ágil você não tem um padrão. A cada projeto muda a produtividade, a pontuação e a velocidade*". (Entrevistado 3)

4.5. Construção da estória central do caso

Nesta seção, brevemente, discutiremos como aplicar nossas descobertas em casos práticos. É importante notar que estas recomendações não foram testadas na prática, de modo que só ilustram como as estratégias podem ser derivadas a partir dos resultados de uma teoria explicativa. A integração dos resultados com outros estudos de caso poderia melhorar a nossa compreensão de como a ferramenta afeta as relações no time. A utilização dos diários pelos desenvolvedores não foi realizada da maneira como esperada. Portanto, não conseguimos utilizar as informações desse artefato nas percepções em relação ao processo. Portanto, a estória central poderia ser reinterpretada com base nos resultados destes testes.

Vale salientar que algumas práticas ágeis como reuniões diárias, o monitoramento do desempenho da equipe e a alocação das atividades com a ferramenta,

tem atuado positivamente em direção a um comportamento motivado devido a transparência fornecida por essas práticas, o fortalecimento do trabalho colaborativo e a melhoria no relacionamento da equipe. Foi observado que o trabalho repetitivo em algumas fases dos projetos e os limites para as oportunidades de crescimento geram desmotivação. Estratégias de rotação de trabalho poderiam acomodar as diferentes aspirações individuais relacionadas com a variedade de tarefas. Combinado com o desenvolvimento de competências, esta estratégia também afetaria as necessidades de crescimento relacionados com a auto realização. Além disso, um plano formal de carreira, oferecendo ações de lucro ligado a resultados de desempenho, poderia criar mais oportunidades de crescimento interno, tornando a equipe mais comprometida com as metas de longo prazo da organização e evitar a rotatividade.

5. Considerações finais e trabalhos futuros

Neste artigo apresentamos os resultados de um estudo de caso qualitativo de cunho etnográfico onde investigamos como a utilização de ferramentas de monitoração de desempenho, em projetos ágeis, influenciam no relacionamento dos membros da equipe.

No primeiro momento, o processo foi implantado em todos os projetos e as equipes estavam motivadas. A realização de reuniões diárias e acompanhamentos quinzenais, onde todas as atividades, pontuações e resultados obtidos eram devidamente registrados na ferramenta, melhoraram, substancialmente, a comunicação do grupo, a obtenção da transparência do andamento do projeto e do desempenho da equipe. Possibilitando uma maior sinergia entre os membros da equipe, melhorando o relacionamento e o nível de confiança. Por outro lado, os membros de uma das equipe mostraram-se desmotivados com o fato das reuniões diárias não ocorrerem e a distribuição das tarefas não serem registradas na ferramenta.

Embora os resultados apresentados não devam ser considerados como sendo universalmente válidos, os princípios centrais da teoria e do método de pesquisa podem ajudar outros pesquisadores a reinterpretar a teoria em contextos específicos.

Referências

Araujo, F. O. Escolha e aplicação de critérios para formação de equipes de software: implicações para composição de personalidade.Universidade Federal de Pernambuco, 1–106, 2014.

Dingsøyr, T., Nerur, S., Balijepally, V., Moe, N. B. A decade of agile methodologies: Towards explaining agile software development, 2012.

França, A. C. C., Araújo, A. C. M. L., Silva, F. B. Q. Motivation of software engineers: A qualitative case study of a research and development organisation. 6th International Workshop on Cooperative and Human Aspects of Software Engineering. IEEE, San Francisco, 9–16, 2013.

Merriam, S. B., Tisdell, E. J. Qualitative research: A guide to design and implementation. John Wiley & Sons, San Francisco, 2015.

Strauss, A., & Corbin, J. Pesquisa qualitativa: técnicas e procedimentos para o desenvolvimento de teoria fundamentada. (Tradução Luciane de oliveira da Rocha). 2. ed., Porto Alegre: Artmed, 288, 2008.

Um estudo preliminar sobre o uso de uma arquitetura deep learning para seleção de respostas no problema de recuperação de código-fonte

Marcelo de Rezende Martins[1], **Marco Aurélio Gerosa**[2]

[1]Instituto de Pesquisas Tecnológicas (IPT)
São Paulo – SP – Brazil

[2]Northern Arizona University
Flagstaff, AZ, US

rezende.martins@gmail.com, Marco.Gerosa@nau.edu

Abstract. *Code retrieval techniques aim to select a code snippet to solve a question given a set of possible solutions. This article presents a preliminary study about a new approach for code retrieval, applying an answer selection deep learning architecture. We present the preliminary results of a bidirectional LSTM with convolutional network model adapted for code retrieval. After 20 runs on a evaluation dataset, our model could achieve a mean MRR of* 0.60 ± 0.02 *and a top-1 accuracy up to 51%.*

Resumo. *Dado uma questão e um conjunto de trechos de código-fonte, recuperação de código-fonte ou code retrieval busca encontrar o código que soluciona a dada questão. Este artigo apresenta um estudo preliminar sobre uma nova abordagem para o problema de recuperação de código-fonte, utilizando uma arquitetura deep learning de seleção de respostas. Apresentamos os resultados preliminares do modelo de redes neurais recorrentes bidirecionais (bi-LSTM) com redes neurais convolucionais (CNN) adaptado para o problema de recuperação de código-fonte. Após 20 execuções na amostra de teste, nosso modelo conseguiu atingir uma média MRR de* $0,60 \pm 0,02$ *e a medida top-1 o máximo de 51%.*

1. Introdução

Segundo [Allamanis et al. 2015], recuperação de código-fonte (*code retrieval*) é um problema de recuperação de informação, onde dado uma questão ou uma descrição em linguagem natural e um conjunto de possíveis trechos de código-fonte, o objetivo é recuperar o trecho de código-fonte que solucione a questão ou seja mais relevante de acordo com a descrição. Já segundo [Lai et al. 2018], dado uma questão e um conjunto de possíveis respostas, ambos em linguagem natural, seleção de respostas ou *answer selection* busca identificar qual resposta consegue responder a pergunta corretamente.

O problema do *code retrieval* busca associar um texto em linguagem natural a um código-fonte. Esta associação tem diversas aplicações como busca de código-fonte a partir de uma consulta em linguagem natural, documentação e geração de programas a partir de uma especificação, por exemplo [Allamanis et al. 2018]. Em engenharia de software, a associação entre código-fonte e texto em linguagem natural pode auxiliar na

rastreabilidade de requisitos. Além disso, pode ajudar o desenvolvedor na geração de código-fonte. Um modelo pode gerar testes de unidade a partir de uma estória de usuário.

Partindo da hipótese inicial de que software é uma forma de comunicação humana e tem propriedades estatísticas similares a corpora de linguagem natural [Allamanis et al. 2018]. Neste artigo, propusemos uma nova abordagem para o problema de *code retrieval*. Abordamos o problema sob a perspectiva do problema de *answer selection*, problema conhecido em NLP (Natural Language Processing), i.e., utilizamos soluções inicialmente propostas para um conjunto de dados formado por perguntas e respostas em linguagem natural e aplicamos em um conjunto formado por pares de perguntas e trechos de código-fonte.

Inicialmente, avaliamos o desempenho de soluções de arquiteturas deep learning comumente utilizadas no problema de *answer selection*. Optamos pela solução de deep learning, mais especificamente uma arquitetura composta por uma rede neural recorrente com uma camada de rede convolucional, devido ao bom desempenho apresentado nos dados InsuranceQA [Feng et al. 2015], um conjunto de dados de referência para avaliação de modelos em *answer selection*.

Além disso, as soluções anteriores propostas para *answer selection* tipicamente baseavam-se em técnicas de pré-processamento para extrair características relevantes para auxiliar o modelo na predição, ferramentas externas e até ferramentas linguísticas [Lai et al. 2018]. Enquanto a definição típica para deep learning é que ele aprende *deep representations*, i.e., aprende múltiplos níveis de representações e abstrações dos dados. Ele mostrou-se capaz de representar imagens, textos, dados de diferentes contextos juntos em um mesmo modelo [Zhang et al. 2019]. No nosso caso, isto é um fator importante, pois o nosso intuito é criar um modelo capaz de representar um conjunto de dados formado por textos em linguagem natural e trechos de código-fonte.

Neste artigo apresentamos os resultados preliminares de um estudo sobre o uso de arquitetura deep learning de *answer selection* no problema do *code retrieval*. Inicialmente, utilizamos a arquitetura bi-LSTM com CNN proposta por [Tan et al. 2015]. Além de propor uma nova arquitetura para este problema, avaliamos o modelo utilizando os dados de entrada da base StaQC, criada por [Yao et al. 2018]. Esta base de dados é composta de milhares de pares de perguntas e trechos de código-fonte do StackOverFlow[1].

2. Método

Conforme mencionado anteriormente, neste trabalho abordamos o problema do *code retrieval* sob a perspectiva do *answer selection*. Utilizamos a arquitetura proposta por [Tan et al. 2015], que utiliza a função de classificação *pairwise* e uma arquitetura siamesa, de acordo com [Lai et al. 2018]. Para [Lai et al. 2018], o problema de *answer selection* pode ser analisado sob duas formas: a de aprendizado e a de arquitetura.

2.1. Forma de aprendizado

O problema de *answer selection* consiste em encontrar a resposta mais relevante dado uma questão. Ele pode ser abordado como uma problema de classificação, onde o objetivo é classificar com uma pontuação melhor as respostas mais relevantes de acordo com a questão.

[1] `https://www.stackoverflow.com`

[Tan et al. 2015] utilizaram o método *pairwise*, no qual o objetivo da função é classificar as respostas corretas com uma pontuação maior que a das incorretas. Dado uma questão, o método avalia o conjunto de repostas e aprende a classificar qual resposta é mais relevante para a questão. Por exemplo, o modelo proposto por [Tan et al. 2015] e utilizado como referência neste artigo, os dados de entrada para o treinamento são triplas (q_i, c_i^+, c_i^-), onde q_i é uma questão, c_i^+ é uma resposta correta, c_i^- é uma resposta incorreta. Seja a função de perda, *hinge*, definida como:

$$L = max(0, m - h_\theta(q_i, c_i^+) + h_\theta(q_i, c_i^-)) \quad (1)$$

Onde m é a margem. Se $h_\theta(q_i, c_i^+) - h_\theta(q_i, c_i^-) < m$ então L é positivo. Quando esta condição é satisfeita, a implicação é que o sistema classifica a resposta correta abaixo da resposta incorreta, ou a questão correta pontua um pouco acima da resposta incorreta. Por outro lado, se a resposta correta tem uma pontuação maior que a incorreta por uma margem acima ou igual a m (i.e., $h_\theta(q_i, c_i^+) - h_\theta(q_i, c_i^-) \geq m$, a função de perda é igual a zero. Em resumo, a função de perda incentiva a reposta correta a ter uma pontuação maior que a incorreta por uma certa margem [Lai et al. 2018].

[Tan et al. 2015] propuseram o uso da função de similaridade *cosine*. Esta função de similaridade é comumente utilizada em problemas de *answer selection* [Feng et al. 2015].

2.2. Arquitetura

[Tan et al. 2015] propuseram uma arquitetura que utiliza uma rede neural recorrente bidirecional, mais especificamente uma rede LSTM (biLSTM) [Hochreiter and Schmidhuber 1997] e uma camada pooling para construir a representação dos vetores de entrada. Ao final, o modelo utiliza a função de similaridade *cosine* para calcular a distância entre as representações.

A arquitetura de referência que iremos utilizar acrescenta uma estrutura de rede convolucional na saída da rede bi-LSTM. Os vetores de saída da rede bi-LSTM tornam-se vetores de entrada na na estrutura CNN. O CNN auxilia na síntese da representação dos vetores. A imagem ilustrativa da arquitetura pode ser visualizada na Figura 1.

3. Experimento

Para avaliar a arquitetura proposta por [Tan et al. 2015] composta por uma rede bi-LSTM com CNN no problema do *code retrieval*, utilizamos os dados disponibilizados por [Yao et al. 2018]. Em seu trabalho, [Yao et al. 2018] coletaram mais de **147 mil** pares de perguntas e trechos de código-fonte em Python e aproximadamente **119 mil** pares de perguntas e trechos de código-fonte em SQL.

Inicialmente, utilizamos uma amostra de 62.252 pares (q_i, c_i^+) em Python. Onde q_i é o título de uma questão no site StackOverFlow e c_i^+ corresponde a um trecho de código-fonte apontado como solução para a questão q_i. Destes 62.252 pares, excluímos 2.169 pares que foram anotados manualmente. Estes 2.169 pares (q_i, c_i^+) anotados manualmente serão divididos em duas amostras: *DEV* e *EVAL*.

Conforme a Tabela 2, utilizaremos 60.083 pares (q_i, c_i^+) para treinamento. Para obter o modelo, adotamos o mesmo procedimento proposto por [Iyer et al. 2016]. No

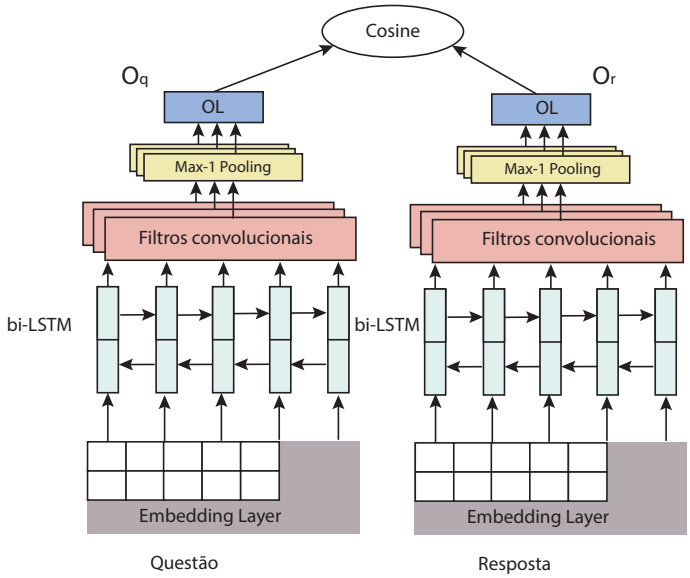


Figura 1. Figura da arquitetura proposta para o problema do *code retrieval*. Figura adaptada do [Tan et al. 2015].

Entrada	Tipo de Dado	Média	Desvio Padrão	Mínimo	25%	50%	75%	Máximo
Questão	Tamanho	$51,60$	$18,57$	13	38	49	62	150
	# de palavras	$8,9$	$3,64$	2	6	8	11	32
Código-Fonte	Tamanho	326	477	4	95	192	380	17.200
	# de palavras	$48,84$	$65,79$	0	16	31	58	3.170

Tabela 1. Estatística descritiva da amostra de 62.252 pares (q_i, c_i^+) em Python extraída do conjunto de dados disponibilizado por [Yao et al. 2018].

caso, o modelo será treinado, inicialmente, durante 80 épocas. Caso o *learning rate* fique abaixo de 0,001, o treinamento é interrompido. Ao final de cada época, o modelo é avaliado na amostra *DEV*.

A avaliação na amostra *DEV* consiste em avaliar a qualidade do modelo calculando o *Mean Reciprocal Rank* (MRR). Esta avaliação é feita da seguinte forma: Para cada par (q_i, c_i^+) na amostra *DEV*, selecionaremos outros 49 distratores $c^{'}$ da amostra de treinamento, onde $c^{'} \neq c_i^+$. Estes 50 pares (c_i, q_i) serão classificados de acordo com a função h_θ de similaridade. Posteriormente, o MRR de cada questão q_i é calculado. Ao final, calculamos a média do valor do MRR. O modelo de treinamento que obter a maior média MRR na amostra *DEV* será escolhido.

Após o término do treinamento e com o modelo com a maior média MRR na amostra *DEV* escolhido, a avaliação final é feita. Durante a avaliação final, a qualidade do modelo é avaliado na amostra *EVAL*. O procedimento é o mesmo adotado para avaliar o modelo na amostra *DEV*.

3.1. Configuração

Os dados foram representados como sequência de tokens. Utilizamos uma representação distribuída word2vec [Mikolov et al. 2013]. Diferente do *answer selection* proposto por [Tan et al. 2015] no qual ele cria apenas uma representação distribuída para a amostra

Amostras	**Quantidade de** (q_i, c_i^+)
Treinamento	60.083
DEV	1.085
EVAL	1.084
Total	**62.252**

Tabela 2. Divisão das amostras para treinamento, avaliação do modelo (DEV) e avaliação final (EVAL) conforme os critérios adotados por [Iyer et al. 2016].

inteira, nós geramos a representação distribuída para as questões e outra para os trechos do código-fonte.

Quanto ao código-fonte, foi feito um pré-processamento. Utilizamos a função disponibilizada por [Yao et al. 2018] que substitui literais numéricos e texto (*string*) por NUMBER e STRING. Os comentários são removidos e o nome das variáveis são substituídas por VAR.

Os parâmetros de execução foram os mesmos utilizados por [Tan et al. 2015]. Com exceção dos filtros na camada CNN, que reduzimos para o valor 100. O valor utilizado por [Tan et al. 2015] de 1.000, aumentou a capacidade do modelo, causando o *overfitting*.

4. Resultados preliminares

Os resultados do experimento podem ser visualizados na Tabela 3. Os resultados correspondem a média do MRR após 20 rodadas de execução utilizando o melhor modelo obtido a partir do treinamento. E o modelo foi avaliado na amostra *EVAL*.

Nestes resultados preliminares, comparamos o modelo bi-LSTM com CNN com outros dois modelos. O modelo *Embedding*, mais simples, é composto apenas por uma camada com a representação distribuída das questões e trechos de código-fonte. A saída é uma camada de *maxpool* e ao final a similaridade é calculada através da função *cosine*.

O modelo *CNN* é uma rede neural convolucional com uma camada *hidden* de entrada *HL*. Esta camada *hidden* é definida como $z = tanh(Wx + B)$. Onde W é a matriz de pesos; B é o vetor *bias*; x é o vetor de entrada; z é o resultado da função de ativação $tanh$. Este modelo utiliza também o *maxpool* e a mesma função de similaridade.

Imagens ilustrativas das arquiteturas dos modelos *Embedding* e *CNN* podem ser visualizadas nas Figura 2 e Figura 3, respectivamente.

A implementação dos modelos foi feito utilizando a biblioteca Keras. O código de implementação e o resultados preliminares estão disponíveis no repositório Git `https://github.com/mrezende/keras-language-modeling`. Além disso, os dados pré-processados podem ser visualizados no repositório `https://github.com/mrezende/stack_over_flow_python`.

De acordo com a Tabela 3, **bi-LSTM-CNN** obteve o melhor desempenho. Porém, CNN obteve um resultado muito próximo e o tempo de treinamento é muito menor. O tempo total de treinamento da rede bi-LSTM com CNN levou em torno de 48 minutos, utilizando uma GPU Tesla K80. Enquanto a arquitetura CNN levou em torno de 6s.

Em todos os modelos, utilizamos uma camada de *maxpool* e a função de simila-

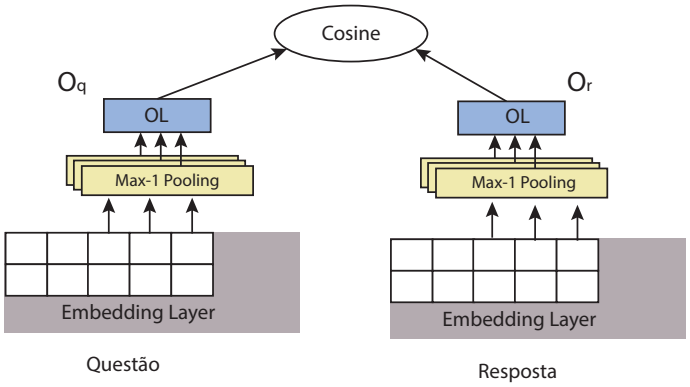


Figura 2. Figura da arquitetura *Embedding*. Figura adaptada do [Tan et al. 2015].

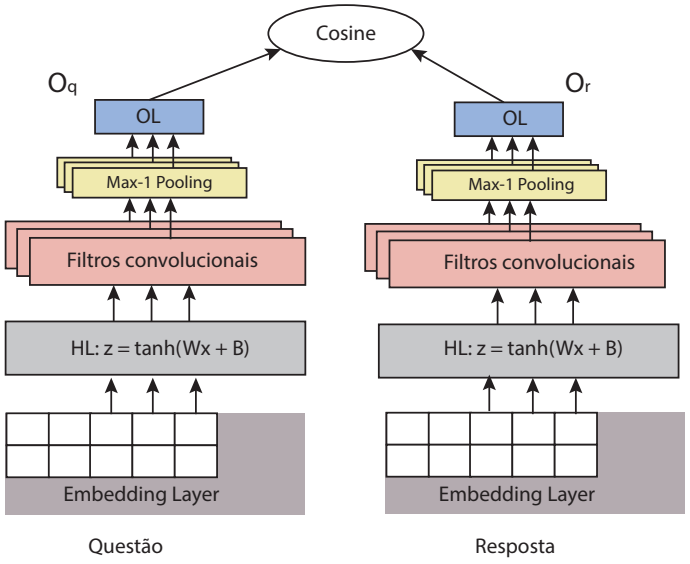


Figura 3. Figura da arquitetura *CNN*. Figura adaptada do [Tan et al. 2015].

ridade *cosine*. Utilizamos o valor de margem $0,009$ para a função de perda *hinge*, valor proposto por [Feng et al. 2015].

4.1. Ameaças à validade

Os dados utilizados no treinamento para obter o modelo final foram coletados automaticamente por [Yao et al. 2018]. [Yao et al. 2018] criaram um modelo composto por uma rede neural recorrente para obter os pares de questões e trechos de código-fonte automaticamente. E para treinar este modelo, foram utilizados os dados anotados manualmente. Estes dados anotados manualmente (amostras *DEV* e *EVAL*) são os mesmos utilizados na avaliação do nosso modelo.

Para minimizar o viés, utilizamos o mesmo procedimento de avaliação proposto por [Iyer et al. 2016]. Para cada par de (q_i, c_i^+), selecionamos aleatoriamente 49 distratores $c^{'}$ da amostra de treinamento, onde $c^{'} \neq c_i$.

Modelos	Resultados (MRR)
Embedding	$0,52 \pm 0,01$
CNN	$0,58 \pm 0,01$
bi-LSTM-CNN	**0,60 $\pm$ 0,02**

Tabela 3. Resultado preliminar do modelo bi-LSTM-CNN proposto em comparação a outros dois modelos (CNN e Embedding). Estes resultados foram obtidos a partir da amostra EVAL.

5. Conclusões

Neste trabalho, propusemos uma abordagem diferente para o problema do *code retrieval*. Nosso intuito foi abordar o problema do *code retrieval* sob a perspectiva do problema *answer selection*, já conhecido em NLP. E dado o bom desempenho dos modelos deep learning no contexto do *answer selection*, utilizamos o modelo proposto por [Tan et al. 2015].

Conforme apresentado na Seção 4, o modelo proposto por [Tan et al. 2015] apresentou um bom desempenho. O valor médio de MRR de $0,60 \pm 0,02$ é um valor expressivo. Este resultado preliminar serve como um indicativo para aprimorarmos o modelo para o problema do *code retrieval*. [Yao et al. 2018] obtiveram uma média MRR de $0,57 \pm 0,02$ e [Iyer et al. 2016] obtiveram $0,44 \pm 0,01$, porém numa amostra de pares de questões e códigos-fontes em SQL disponibilizada por [Iyer et al. 2016]. O próximo passo é avaliarmos a nossa arquitetura utilizando os mesmos dados e procedimentos proposto por [Yao et al. 2018].

Uma frente ainda a ser explorada no problema do *code retrieval* é a disponibilização de dados para avaliação dos modelos. A área de reconhecimento de imagem tem o *ImageNet* [Deng et al. 2009] um vasto banco de dados com mais de 14 milhões de imagens anotados manualmente. A área de inferência em linguagem natural para determinar se uma hipótese é verdadeira, falsa ou neutra contém mais de 570 mil sentenças em inglês anotadas manualmente [Bowman et al. 2015].

O próprio problema de *answer selection* utiliza Trec-QA [Wang et al. 2007] e InsuranceQA [Feng et al. 2015]. O trabalho de [Yao et al. 2018] anotou 4.884 pares de questões e trechos de código-fonte manualmente. Isto é um passo importante e conforme mais dados curados e organizados forem disponibilizados, mais a área de *machine learning* aplicado a código-fonte e engenharia de software tende a ganhar.

Referências

Allamanis, M., Barr, E. T., Devanbu, P., and Sutton, C. (2018). A survey of machine learning for big code and naturalness. *ACM Comput. Surv.*, 51(4):81:1–81:37.

Allamanis, M., Tarlow, D., Gordon, A. D., and Wei, Y. (2015). Bimodal modelling of source code and natural language. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2123–2132. JMLR.org.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Confe-*

rence on Empirical Methods in Natural Language Processing (EMNLP). Association for Computational Linguistics.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Feng, M., Xiang, B., Glass, M. R., Wang, L., and Zhou, B. (2015). Applying deep learning to answer selection: A study and an open task. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, pages 813–820.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Iyer, S., Konstas, I., Cheung, A., and Zettlemoyer, L. (2016). Summarizing source code using a neural attention model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083, Berlin, Germany. Association for Computational Linguistics.

Lai, T. M., Bui, T., and Li, S. (2018). A review on deep learning techniques applied to answer selection. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2132–2144, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Tan, M., dos Santos, C., Xiang, B., and Zhou, B. (2015). Lstm-based deep learning models for non-factoid answer selection. *CoRR*, abs/1511.04108.

Wang, M., Smith, N. A., and Mitamura, T. (2007). What is the Jeopardy model? a quasi-synchronous grammar for QA. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 22–32, Prague, Czech Republic. Association for Computational Linguistics.

Yao, Z., Weld, D. S., Chen, W.-P., and Sun, H. (2018). Staqc: A systematically mined question-code dataset from stack overflow. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pages 1693–1703, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Zhang, S., Yao, L., Sun, A., and Tay, Y. (2019). Deep learning based recommender system: A survey and new perspectives. *ACM Comput. Surv.*, 52(1):5:1–5:38.