UNIVERSIDADE FEDERAL DA BAHIA
UNIVERSIDADE SALVADOR
UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA

TESE DE DOUTORADO

UM ESTUDO DE CARACTERIZAÇÃO DE MUDANÇAS ARQUITETURAIS EM PROJETOS DE SOFTWARE LIVRE

TIAGO OLIVEIRA MOTTA

PROGRAMA MULTIINSTITUCIONAL DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Salvador
2021

PMCC-Dsc-0036

TIAGO OLIVEIRA MOTTA

UM ESTUDO DE CARACTERIZAÇÃO DE MUDANÇAS ARQUITETURAIS EM PROJETOS DE SOFTWARE LIVRE

Tese apresentada ao PROGRAMA MULTIINSTITUCIONAL DE PÓS GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO da UNIVERSIDADE FEDERAL DA BAHIA, UNIVERSIDADE SALVADOR e UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA como requisito parcial para obtenção do grau de Doutor em CIÊNCIA DA COMPUTAÇÃO.

Orientador: RODRIGO ROCHA GOMES E SOUZA
Co-orientador: CLAUDIO NOGUEIRA SANT´ANNA

Salvador
2021

Ficha catalográfica.

M921 MOTTA, TIAGO OLIVEIRA

Um estudo de Caracterização de Mudanças Arquiteturais em Projetos de Software Livre/ TIAGO OLIVEIRA MOTTA– Salvador, 2021.

294 f.

Orientador: RODRIGO ROCHA GOMES E SOUZA.
Co-orientador: CLAUDIO NOGUEIRA SANT´ANNA.
Tese de Doutorado– UNIVERSIDADE FEDERAL DA BAHIA, INSTITUTO DE MATEMÁTICA, 2021.

1. Arquitetura de Software 2. Mineração de Repositórios 3. Métricas de Código 4. Recuperação da Informação Arquitetural 5. Engenharia de Software.

I. SOUZA, RODRIGO R. G.. II. SANT´ANNA, CLAUDIO.
III. UNIVERSIDADE FEDERAL DA BAHIA. INSTITUTO DE MATEMÁTICA. IV. Título.

CDU 004.274

TIAGO OLIVEIRA MOTTA

"UM ESTUDO DE CARACTERIZAÇÃO DE MUDANÇAS ARQUITETURAIS EM PROJETOS DE SOFTWARE LIVRE"

Esta tese foi julgada adequada à obtenção do título de Doutor em Ciência da Computação e aprovada em sua forma final pelo Programa Multi-institucional de Pós-Graduação em Ciência da Computação da UFBA-UEFS-UNIFACS.

Salvador, 26 de abril de 2021.

Prof. Dr. Rodrigo Rocha Gomes e Souza
Orientador

Prof. Dr. Manoel Gomes de Mendonça Neto
Examinador Interno

Profª. Drª. Christina von Flach Garcia Chavez
Examinadora Interna

Prof. Dr. Paulo Cesar Masiero
Examinador Externo

Prof. Dr. Paulo Roberto Miranda Meirelles
Examinador Externo

À Floricy Oliveira Santos $(in\ memorian)$, *Isabel Santos Mota* $(in\ memorian)$ *e Stella Graça Oliveira Motta.*

AGRADECIMENTOS

A trajetória até uma defesa de doutorado se inicia desde o processo de alfabetização e se finda na defesa de uma tese inédita e na entrega de sua versão final após as correções sugeridas pela banca examinadora. Nesse espaço, gostaria de agradecer a todos que participaram dessa longa caminhada.

Inicialmente, obrigado a Deus todo poderoso. Sem você aqui não estaria, não conseguiria concluir esse estudo e essa etapa de minha vida pessoal, acadêmica e profissional.

Agradeço aos meus orixás e santos protetores. Nos momentos de profunda desilusão, fraqueza e tropeços foi onde encontrei recursos para não desistir e seguir em frente.

Agradeço à Fundação de Amparo à Pesquisa do Estado da Bahia e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico, pelo apoio financeiro concedido durante este doutorado.

Agradeço à Universidade Federal da Bahia, Universidade Salvador e Universidade Estadual de Feira de Santana pela oferta do curso.

Agradeço à minha companheira Paula Nubia Soares Dalto Motta, que acompanhou todo esse processo me apoiando incondicionalmente.

Agradeço ao Professor Rodrigo Rocha, excelente profissional, ser humano e amigo. Um orientador que todos desejariam ter! Obrigado por tudo Rodrigo!

Agradeço ao Professor Cláudio Sant'Anna, grande profissional que contribuiu, ao longo deste doutorado, com muitas considerações, intuições, indagações, reflexões e devolutivas que fortaleceram esta pesquisa de doutorado.

Agradeço aos professores das componentes curriculares que cursei durante este doutorado, todos com grandes contribuições ao meu conhecimento científico e acadêmico: Christina Flach, Cláudio Sant'Anna, Eduardo Almeida e Manoel Mendonça.

Agradeço ao professor Paulo Masiero, que durante sua passagem como pesquisador sênior-visitante pela UFBA, não hesitou em ouvir nossa proposta de doutorado e contribuir profundamente no amadurecimento dela.

Agradeço ao pesquisador-irmão Eleazar Madriz, além do apoio moral, grande colaborador durante a formulação das equações matemáticas que constam nesta tese.

Agradeço ao pesquisador-desenvolvedor Joênio da Costa. Um incansável desenvolvedor/mantenedo do Analizo. Além de prover suporte ao uso dessa ferramenta, colaborou de forma decisiva para a participação de desenvolvedores de projetos de Software Livre num dos estudos que compõem esta tese. Obrigado Joênio!

Agradeço à minha grande amiga Ana Rita Santiago. Além de uma pessoa, indescritivelmente, do bem, competente, sábia e leal, se dispôs a emprestar parte de seu tempo e intelectualidade na revisão gramatical da versão preliminar deste texto. Muito obrigado Ana!

Agradeço aos amigos-irmãos que fiz durante o período em que morei em Salvador. São conexões construidas para toda a vida: Iuri Souza, Rogers Nascimento, Tassio Vale,

Josualdo da Silva, Jaziel Lobo, Tassio Virginio, Savio Freire, Leandro Souza, Jonatas Bastos, Uoston Souza, João Emiliano, Tiago Machado, Saulo, Marcus Lordelo e André Lordelo. Sem esquecer de Marcão da portaria.

Agradeço às grandes amigas e amigos Marlos Noronha, Rosângela Souza, Tarcila Neres, Marislan Neves, Camila Félix, Hellen Juliana, Thaise Braga, Cleber Pitanga, Thiago Barbosa, Emerson Raimundo, Ronaldo Barros, Marcelo Teles e Babà Alcides.

Agradeço aos grandes profissionais que tornam nossa rotina burocrática na UFBA suave, e que também se tornaram amigos. O Serviço Público Federal precisa de mais Davilenes, Diogos, Márcios, Klebers, Mayaras, Miqueias, Samuels e Silvanas.

Agradeço aos amigos e amigas do Laboratório de Engenharia de Software: Roselane Silva, Glaucya Boechat, Karla Malta, Railana, Michelle Larissa, Leila Carvalho, Leila Karita, Caiza Almeida, Beatriz Brito, Maria Clara, Luana, Mayka, Moara, Mirela, Brunna, Dhennya, Samia, Renata, Magno Luã, Edilton, Albano, Joselito, Daniel Amador, Vitão, Junior, Nildo, Maicon, Alcemir, Alberto Vianna, Alvaro Lordelo, Matheus Passos e Danniel.

Agradeço aos profissionais do PAF, na pessoa do Sr. Luiz Fernando e de Marcão da portaria. Não posso esquecer de Antônio e Antônia da lanchonete do Instituto de Matemática. Cafés excelentes e deliciosos salgados são servidos lá. Meu muito obrigado também ao Marcio livreiro.

Agradeço à todos os pesquisadores e profissionais que dispuseram de seu tempo e intelectualidade para responder aos *websurveys* aplicados ao longo deste estudo.

Agradeço aos colegas da Universidade Federal do Recôncavo da Bahia, que por quatro anos permitiram minha dedicação a este estudo. Em especial aos amigos Érico Figueiredo, Aroldo Félix e Jadiel Pereira.

Agradeço à uma lavadeira (pássaro) que me visitou em casa por muitas manhãs. Ela (ou ele) me mostrava que o renascer do sol revigora as nossas energias e que é preciso ir à luta mesmo nas adversidades. Pois só a luta traz o êxito, não caberia a desistência à intentos tão importantes. Mostrou-me também que a natureza ama sem pensar. Apenas ama.

Agradeço aos professores que tive ao longo dessa caminhada. Inicialmente na pessoa de Tia Fátima, responsável por minha alfabetização. Nas pessoas de Tia Eurides (*in memorian*) e Tia Edite Santana, ícones de meu ensino fundamental I, pois me tornaram um sujeito mais que curioso. Nas pessoas de Gilmar Trindade, Norma Ney, Jeânia, Guadalupe, Lilian, estes do fundamental II, por me ensinarem que aprender é uma questão de querer e que nem sempre a resposta está na primeira linha. Nas pessoas do Prof. Raimundo Rocha e Profa. Célia Barros por tomar gosto pelas ciências exatas e me ensinarem a olhar a Matemática como linguagem e a Física como uma forma de observar a natureza. E por fim aos professores que tive na graduação, que me tornaram um sujeito ainda mais curioso e questionador, nas pessoas de Álvaro Degas, Ana Paula Lopes, Antônio Louro, Eduardo Rihan, Diego Frias, Anderson Mol e Vânia Cordeiro. Um destaque aos professores do Mestrado Felix Milian, Mauricio Moreau e Milton Ferreira, pois me incentivaram a continuar na trilha da investigação científica.

"Felizes os que têm fome e sede de justiça, porque serão saciados. Felizes os que são misericordiosos, porque encontrarão a misericórdia."
Biblia Sagrada, Mateus 5:6-7

"Para ser justo, é preciso ouvir a todos e se desprender do ego."
Xangô

"Se eu vi mais longe, foi por estar sobre ombros de gigantes."
Isaac Newton, 1676

"Estou fazendo um sistema operacional gratuito (apenas um hobby, não será grande e profissional como GNU) para 386/486 AT."
Linus Torvalds, 1991

RESUMO

Contexto. A literatura em vigor aponta que a escassez de documentação arquitetural predomina em projetos de software livre. Nesse sentido, diversas técnicas têm sido propostas para recuperar informações sobre a arquitetura, e a maioria delas se concentra em aspectos estruturais a partir do código-fonte, um elemento que acompanha a evolução do produto de software.
Problema. No entanto, essas técnicas não recuperam outras informações sobre a arquitetura (tampouco a sua evolução), tais como os objetivos, restrições e motivações, requisitos não-funcionais, mecanismos de acesso a dados ou de comunicação utilizados no projeto.
Objetivo. A proposta desse trabalho é caracterizar mudanças arquiteturais em projetos de Software Livre, apresentando os impactos por elas causadas no código-fonte do projeto em comparação com outros tipos de mudança em que a arquitetura não foi modificada. As hipóteses são que as mensagens de *commit* registram informações sobre mudanças na arquitetura do projeto e que esse tipo de mudança impacta o código do projeto de forma diferente em comparação com mudanças onde a arquitetura não foi modificada.
Metodologia. Inicialmente, foi produzido um estudo empírico, a partir da mineração de repositórios, no qual identificaram-se informações arquiteturais em registros de informações sobre mudanças que constam em mensagens de *commit*. A partir dos dados coletados, foi realizado um estudo exploratório que caracterizou tais mudanças sobre em que momento da evolução do projeto ocorreram, quem são os seus autores, quais tópicos arquiteturais são modificados, além das características dessas mensagens e do grupo de módulos atingidos pela mudança. Após esse estudo, foi conduzido um segundo estudo empírico, envolvendo a extração de métricas de código-fonte das versões do projeto identificadas como tendo sofrido modificações em sua arquitetura. Por fim, foi realizado um estudo para avaliar os resultados obtidos, considerando as opiniões de pesquisadores da área de Arquitetura e Engenharia de Software, assim como de desenvolvedores frente à caracterização obtida perante as mudanças arquiteturais.
Resultados. Dentre as descobertas realizadas, pode-se destacar que períodos de lançamento de novas *releases* são concentradores de mudanças arquiteturais, os principais colaboradores do projeto também são os colaboradores que mais modificam a arquitetura e todos os tópicos arquiteturais do projeto são alvo de modificações, porém em escalas diferentes. Outros aspectos importantes sobre as mudanças arquiteturais são que (i) o tamanho do código tende a aumentar; (ii) a complexidade e a coesão do código tendem a se manter; e (iii) o acoplamento verificado aumentou para duas métricas e manteve-se em outras três, quando comparadas à mudanças no projeto onde a arquitetura não foi atingida. Por fim, durante o terceiro estudo desta tese, desenvolvedores e pesquisadores confirmaram a hipótese que a documentação arquitetural é escassa em projetos de Software Livre, e a principal forma de recuperar informações sobre a arquitetura, até o momento, é

analisando o código-fonte de diversas formas. A maioria dos participantes da pesquisa concordam com os resultados obtidos por este estudo em relação à variação das métricas de código, tanto como sendo generalizáveis, como sendo verificáveis nos projetos onde pesquisam e desenvolvem. Alguns desenvolvedores apresentaram pequenas discordâncias quanto às variações de métricas obtidas. Sobre as mensagens de *commit* como fontes de informação sobre a arquitetura, a maioria de desenvolvedores e pesquisadores afirmaram ser uma fonte promissora de informações, mas fizeram a ressalva de que um cenário ideal seria combiná-las com outras informações do projeto.

Palavras-chave: Modificações Arquiteturais; Traços Arquiteturais; Compreensão de Software; Análise Estática de Código; Estudos Empíricos.

ABSTRACT

Context. The literature background indicates that the scarcity of architectural documentation predominates in free software projects. In this sense, several techniques have been proposed to retrieve information about the architecture; most of them focus on structural aspects from the source code, an element that accompanies the evolution of the software product.
Problem. However, these techniques do not recover other information about the architecture (nor its evolution), such as the goals, constraints, and motivations, non-functional requirements, data access mechanisms, or communication mechanisms used in the project.
Goal. The purpose of this work is to characterize architectural changes in Free Software projects, presenting the impacts caused by them in the project's source code compared with other types of changes without architectural implication. The hypotheses are that the commits messages record information about changes in the architectural project. This type of change impacts the project code differently compared to changes that affect architecture.
Methodology. Initially, we conducted a study where, from repository mining, we identified information about architectural changes contained in commit messages. From the data collected, we conducted an exploratory study that characterized such changes regarding (i) the moment in the project's evolution when they took place, (ii) who their authors are, (iii) which architectural topics are modified, besides (iv) the characteristics of these messages and (v) the group of modules affected by the change. After this study, we conducted a second empirical study involving the extraction of metrics from the source code of the versions of the project identified as having undergone modifications in its architecture. Finally, we conducted a study to evaluate the results obtained, considering the feedback of both developers and researchers in Architecture and Software Engineering regarding the characterization of architectural changes.
Results. Among the results, we can highlight that periods encompassing the release of new versions are concentrators of architectural changes; the main collaborators of the project are also the collaborators who modify the architecture the most; all the architectural topics of the project are subject to modifications but at different scales. About the second study, we highlight three results. The size of the code tends to increase. The complexity and cohesion of the code tend to remain constant. The coupling of the code tends to increase for two metrics and to remain constant in three others. Finally, in the third study, we hear developers and researchers about the results of the previous two studies. They confirmed the hypothesis that architectural documentation is scarce in Open Source projects. They claim that the main strategy to retrieve information about architecture is by analyzing the source code in several ways. Most research participants agree with the results obtained by this study about the variation of code

metrics. They believe that the variations are generalizable and are recurrent in the projects that they research and develop. Some developers showed slight disagreements from the variations of metrics obtained. Regarding commit messages as sources of information about architecture, most developers and researchers agree that it is a promising source of information, although the ideal scenario would be to combine them with other information from the project.

Keywords: Architectural Modifications, Architectural Traces, Software Comprehension, Static Code Analysis, Empirical Studies.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUÇÃO

Arquitetura de software é uma expressão que se refere às características de um projeto de software a partir de um elevado nível de abstração. As definições clássicas com mais citações em trabalhos acadêmicos são as de Perry e Wolf (1992), Garlan e Shaw (1993) e Bass, Clements e Kazman (1998). Tais estudos levam em consideração desde os aspectos estruturais e decisões, passando por objetivos e restrições em projetos de software e chegando à descrição das relações entre os elementos arquiteturais. Em suma, a arquitetura descreve em alto nível a organização de um projeto de software. Essa é uma área que tem ganhado destaque na pesquisa de engenharia de software nas últimas 3 décadas. Garlan (2014) destaca que nesse período o desenvolvimento de base metodológica para o tratamento do projeto arquitetural foi o principal avanço obtido. Nesse estudo ele ainda aponta as lacunas e desafios encontrados até aquela data nessa área de estudo.

Um aspecto comum da arquitetura de software são as decisões tomadas ao longo do ciclo de vida de um projeto de software. Trata-se de aspectos que sofrem mudanças, mas que se têm poucos registros e informações sobre o que, quando e por que foi mudado, podendo resultar em projetos indesejáveis e diferentes do que se almeja produzir (GURP; BOSCH, 2002; FARID; AZAM; IQBAL, 2011). Informações sobre essas mudanças estão expressas no código-fonte produzido (OREIZY; MEDVIDOVIC; TAYLOR, 1998) e do código-fonte vem as estratégias mais utilizadas para a recuperação dessas informações (MANCORIDIS et al., 1999; RIVA et al., 2004; PINZGER, 2005; KNODEL et al., 2006; GARCIA et al., 2012). No entanto, essas estratégias trazem poucas informações, pois o código retrata apenas os aspectos estruturais da arquitetura de um software.

Buscar mais informações sobre a arquitetura de um projeto de software é desafiador no contexto de Projetos de Software Livre[1]. Nesse contexto, os colaboradores geralmente são distribuídos geograficamente e raramente há documentação clara do projeto disponível (BRUNET et al., 2014). Ding et al. (2014) confirmam, em seu estudo, que, no contexto

[1]Projetos onde o usuário tem a liberdade de conhecer, modificar e distribuir o software desenvolvido. O site da Free Software Foundation apresenta informações detalhadas sobre os direitos e as licenças associadas ao conceito de Software Livre: https://www.fsf.org

de projetos de Software Livre, há escassez sobre informações arquiteturais e quando ela está disponível é feita em linguagem natural. Isso evidencia que os projetos de Software Livre dependem quase exclusivamente de comunicação por escrito, tais como mensagens de *commit*, comentários de código, canais IRC (Internet Relay Chat), listas de discussão e wikis, para colaboração e coordenação de atividades de desenvolvimento. A comunicação, que ocorre por esses meios, contém muitas informações sobre o sistema de software, a exemplo do histórico de desenvolvimento, razões para mudanças (STOREY et al., 2014; ALKADHI et al., 2017) e decisões de design (KO; DELINE; VENOLIA, 2007; HASSAN, 2008). Esses registros de comunicação, podem ser uma fonte de informações sobre mudanças no projeto, mas apresentam dificuldades de recuperação automática, uma vez que se trata de registros em linguagem natural, ao invés de uma linguagem de descrição específica, conforme já informado acima.

1.1 PROBLEMA

Promover alterações, seja de melhoria, seja de acréscimo de funcionalidades em um projeto de Software Livre, sem observar a arquitetura do projeto, pode provocar diversos efeitos indesejados. Esses efeitos vão desde abreviar a curva de decaimento estrutural (BEHNAMGHADER et al., 2017; KLEEBAUM et al., 2018) até a inserção de *bugs* em partes do projeto onde antes havia estabilidade e funcionamento (de acordo com o projetado), gerando desvios e erosão arquitetural (MANCORIDIS et al., 1999; GARCIA et al., 2012; HAITZER; NAVARRO; ZDUN, 2017).

No entanto, a escassez de informações arquiteturais é predominante nos projetos de Software Livre (DING et al., 2014). Esse problema vem sendo relatado na literatura por alguns pesquisadores, mas os avanços têm ocorrido de forma densa na direção da recuperação de informações estruturais da arquitetura. Informações sobre Mecanismos de Sincronização, Objetivos, Motivações, Restrições, Requisitos não-funcionais, Mecanismos de acesso à dados e Protocolos de comunicação não têm sido tratadas a contento. Além disso, as modificações sofridas na arquitetura implementada têm sido esquecidas nos estudos associados e alguns pesquisadores têm se dedicado a descobrir desvios entre o que foi projetado, inicialmente, e o que de fato foi implementado (MAFFORT et al., 2016; HAITZER; NAVARRO; ZDUN, 2017).

Parte dessa carência pode ser minimizada a partir de um conjunto de trabalhos disponíveis que se propõem a recuperar informações sobre a arquitetura estática de projetos de software a partir do código-fonte (MANCORIDIS et al., 1999; RIVA et al., 2004; GARCIA et al., 2012). Mancoridis et al. (1999) apresentam uma ferramenta para extração da arquitetura de projetos de software a partir do código-fonte. Essa ferramenta apresenta um grafo de dependência entre os módulos, indicando os subsistemas, mas não se refere aos elementos de decisão do projeto, nem sobre objetivos, motivações e restrições. Riva et al. (2004) apresentam uma ferramenta que efetua engenharia reversa sobre o código-fonte de projetos de software e retorna os componentes do projeto e as interações entre eles. Garcia et al. (2012) trazem à tona uma técnica para recuperação de informações estruturais da arquitetura a partir do código-fonte e da participação de engenheiros de software na validação dos resultados obtidos. No entanto, assim como

os demais trabalhos apresentados até aqui, os resultados obtidos se limitam a informar apenas uma das dimensões da arquitetura de software: a organização estrutural de seus componentes.

Em outra direção, um conjunto de trabalhos buscou a identificação das decisões tomadas ao longo de projetos de software (ROGERS et al., 2015; HESSE et al., 2016; ALKADHI et al., 2017; BHAT et al., 2017; KLEEBAUM et al., 2018; SHAHBAZIAN et al., 2018). Hesse et al. (2016) buscaram em *issues trackers* informações sobre as decisões tomadas ao longo do projeto. Eles buscaram a existência de uma estratégia dominante para a documentação desse tipo de informação. Bhat et al. (2017) construíram uma máquina de aprendizagem que procurou em *issues trackers* informações sobre decisões de design. Kleebaum et al. (2018), por sua vez, estudaram a recuperação de decisões tomadas ao longo do projeto e sugeriram técnicas para recuperar essas decisões espalhadas na organização de pacotes do projeto, além de considerarem a consistência existente entre decisões sobre um assunto correlacionado. Já Rogers et al. (2015) sugerem uma técnica para a recuperação dessas decisões registradas em *bugs reports* usando algoritmos de classificação. Ainda sobre a recuperação de decisões, Shahbazian et al. (2018) estudaram repositórios de código-fonte e ofereceram uma ferramenta que identifica e mapeia o impacto das decisões de design sobre a arquitetura do projeto. Alkadhi et al. (2017) apresentam o potencial de recuperação de informações sobre a lógica envolvida nas mudanças ocorridas no projeto de software a partir de registros de reuniões ocorridas em canais de IRC, destinados aos desenvolvedores dos projetos. Esses estudos não tiveram como foco informações exclusivamente sobre mudanças arquiteturais.

Em outro sentido, um conjunto de autores estudou o impacto causado pela indisponibilidade de informações arquiteturais para além da dimensão estrutural (PINZGER, 2005; KNODEL et al., 2006; KO; DELINE; VENOLIA, 2007; DÍAZ-PACE et al., 2016). Ko, DeLine e Venolia (2007) analisaram as atividades de desenvolvedores para identificar razões para o não-prosseguimento de suas atividades e quais soluções eles buscaram para superar esses impedimentos. Os autores relataram que as atividades mais frequentes estavam relacionadas à procura de informações para a compreensão dos módulos onde iriam atuar e que, diante da indisponibilidade dessas, esquadrinharam a identificação e explicações do desenvolvedor responsável. Quando os desenvolvedores anteriores do módulo estavam indisponíveis, a solução foi adiar a realização das tarefas. Pinzger (2005), por sua vez, indica que a maior parte do custo e esforço empregados na evolução e manutenção dos projetos de software estão relacionados ao entendimento da arquitetura do projeto e dos interesses (*concerns*) a eles vinculados. Isso se deve em parte aos métodos utilizados pelos engenheiros que estão relacionados aos mapas mentais construídos sobre o sistema ao invés de consulta à documentação destinada a esse fim. Knodel et al. (2006) citam razões para a disponibilidade, acompanhamento das mudanças e uso das informações arquiteturais no processo de desenvolvimento de software. Esse trabalho ressalta que as decisões tomadas no nível arquitetural afastam ou materializam os objetivos de negócios, requisitos funcionais e de qualidade. Já Díaz-Pace et al. (2016) reforçam a necessidade da documentação arquitetural fiel ao projeto e atualizada junto com o projeto de software. Para isso, eles propõem um método de documentação incremental, associado à evolução do código e que reflita apenas as necessidades de informações manifestadas pelos envolvidos

no projeto. Em matéria de projetos de Software Livre, Ding et al. (2014) apresentam um levantamento sobre o grau de documentação da arquitetura desse tipo de projeto. Eles apontaram que apenas 108 projetos, dentre 2000 pesquisados nos quatro principais repositórios de projetos de Software Livre, possuem algum tipo de documento arquitetural, em geral, descritos em linguagem natural, contando apenas com o modelo arquitetural e uma breve descrição do sistema e sua missão.

Em busca de informações para além da dimensão estrutural da arquitetura, um conjunto de pesquisadores propuseram abordagens para resgatar e disponibilizar outras dimensões da arquitetura e suas mudanças (HASSAN, 2008; PANICHELLA et al., 2014; GRAAF et al., 2014; HAITZER; NAVARRO; ZDUN, 2017). Panichella et al. (2014) estudaram a relação de link e complementaridade entre 3 fontes de informação (*bug trackers*, listas de discussão e registros de bate papo no IRC) de projetos de Software Livre para definir se há mentores para os colaboradores recém-chegados, o grau de sobreposição da informação nas diferentes fontes e a relação entre o papel social do desenvolvedor e o tipo de mudança que ele promove no projeto. Hassan (2008) apresentou um conjunto de fontes de informações não-estruturadas e apontou quais as informações disponíveis nesses locais, destacando aquelas que eram passíveis de recuperação, a exemplo de decisões de design ou de links que podem ser estabelecidos entre essas diferentes fontes de dados. Haitzer, Navarro e Zdun (2017), por sua vez, indicam uma proposta onde seria possível impor a arquitetura projetada, evitando, dessa forma, a necessidade de recuperação da arquitetura ao longo da evolução do projeto e a existência de desvios arquiteturais. Já Graaf et al. (2014) afirmam que a adesão às ontologias podem incentivar a documentação da arquitetura de software. Segundo eles, a arquitetura é pouco documentada, uma vez que há vários tipos de desenvolvedores no projeto interessados nessa informação e sobre diferentes perspectivas. Além disso, o fato de não existir uma linguagem que consiga atender a todas essas perspectivas acentua essa ausência de registros.

Como se vê, a busca por informações arquiteturais, além da dimensão estrutural, tem sido um tópico recorrente, no entanto, além de não se dispor de informações sobre todas as dimensões, pouco se sabe sobre quais características marcam as mudanças na arquitetura ao longo do processo evolutivo de projetos de software. A literatura mostra que modificar a arquitetura inadvertidamente traz consequências como a dificuldade de manutenção e de inclusão de novas funcionalidades, provocando, assim, um decaimento estrutural acelerado, abreviando, portanto, o ciclo de vida dos projetos com essa característica.

Nesse sentido, pesquisando informações sobre as mudanças na arquitetura do projeto, ao longo de seu ciclo de vida (para além dos aspectos estruturais), foram conduzidos estudos empíricos que buscam identificar e caracterizar as mudanças ocorridas na arquitetura de projetos de software ao longo de sua evolução. Esses estudos pretendem identificar, além de uma nova fonte de documentação arquitetural (não construída com essa finalidade), caracterizar as informações encontradas, classificando o tópico arquitetural disponível, como é documentado, se há um padrão nesse tipo de documento, como o projeto é afetado em seus atributos de qualidade – usando métricas de tamanho, complexidade, coesão e acoplamento – além do alcance e aplicabilidade dessas informações para pesquisadores de engenharia de software e para desenvolvedores em geral. As questões de pesquisa apresentadas a seguir guiaram a condução desses estudos.

1.2 QUESTÕES DE PESQUISA

Consoante com o apresentado até aqui, esta pesquisa busca informações sobre mudanças arquiteturais a partir de novas fontes de dados para além do código-fonte. Os esforços empreendidos se concentraram em mensagens de *commit* e nas modificações realizadas no código. As hipóteses levantadas incluem:

1. **A disponibilidade de informações sobre mudanças arquiteturais em comentários de *commits* de código-fonte:** Espera-se que em meio a descrição das mudanças realizadas no projeto, os colaboradores destaquem as mudanças realizadas nas várias dimensões da arquitetura do projeto;

2. **O código-fonte é afetado de forma diferente quando as mudanças no projeto alteram a arquitetura do projeto em comparação com outros tipos de mudanças:** Espera-se que as modificações arquiteturais provoquem mudanças mais acentuadas no tamanho, na complexidade, no acoplamento e na coesão do projeto que outros tipos de modificações onde a arquitetura não foi alterada, e;

3. **Essas informações ajudam na compreensão do projeto tanto para desenvolvedores quanto para pesquisadores da engenharia de software:** Espera-se que as informações identificadas sejam úteis em processos de compreensão da arquitetura dos projetos, principalmente para projetos onde essa informação não se encontra disponível.

Para tanto, estruturaram-se três questões de pesquisa que visam investigar essas hipóteses. Essas questões de pesquisa estão relacionadas, individualmente, à cada hipótese levantada através de estudos empíricos descritos no decorrer desta tese. Assim, as questões de pesquisa principais são:

RQ1: Mensagens de *commit*, disponíveis em sistemas de controle de versão, contêm informações sobre mudanças arquiteturais ao longo da evolução de projetos?
Projetou-se um estudo empírico baseado num grande projeto de Software Livre, onde, através de mineração de repositórios, vasculharam-se mensagens de *commit* em busca de informações que remetessem às mudanças na arquitetura desse projeto. Aqui pretende-se encontrar mensagens de *commit* com conteúdo arquitetural e sinalizar as características dessas mensagens.

RQ2: *Commits* com mudanças arquiteturais afetam as medidas do código-fonte de um projeto de forma diferente em comparação com *commits* com mudanças onde a arquitetura não foi alterada?
Uma vez que a **RQ1** trouxe um conjunto de mensagens, onde foram declaradas modificações na arquitetura do projeto, a continuação desta pesquisa deu-se com a avaliação do impacto sobre código-fonte a partir dessas mudanças. Em virtude disso, foi conduzido um estudo empírico, analisando as alterações ocorridas em cada um dos módulos/arquivos presentes nos *commits* com mudanças na arquitetura, calculando valores de métricas de tamanho, complexidade, acoplamento e coesão. Da mesma forma, foram realizadas

análises dos módulos/arquivos da versão[2] anterior desses módulos/arquivos, antes das modificações declaradas. Ao se calcular as diferenças entre esses valores de métricas, obteve-se uma visão de como o projeto foi alterado. Analisou-se também um conjunto de versões sorteadas ao acaso no mesmo projeto e no período em que foram realizadas as mudanças arquiteturais. Nesses *commits*, no entanto, foram realizados outros tipos de mudanças (a arquitetura não foi alterada) e, dessa forma, adquiriu-se um conjunto de *commits* disponível para a comparação das versões do projeto com um conjunto de dados produzido em um período similar. Assim, pretende-se caracterizar o quanto são diferentes os impactos causados por mudanças no projeto onde ocorreram mudanças na arquitetura daqueles ocasionados por mudanças onde a arquitetura não foi modificada.

RQ3: Desenvolvedores e Pesquisadores da Engenharia de Software podem se beneficiar no processo de compreensão de software ao usar informações sobre mudanças arquiteturais obtidas através de mensagens de *commit*?
Por fim, para avaliar os achados desta pesquisa, foi realizado um estudo empírico envolvendo especialistas no assunto: pesquisadores de Engenharia de Software e desenvolvedores experientes. Nesse estudo, perguntou-se aos participantes qual a sua opinião sobre os achados desta pesquisa: Se as mensagens informando sobre mudanças na arquitetura colaboram na compreensão do software e se nos projetos onde eles atuam, as mudanças na arquitetura provocam os mesmos efeitos que foram observados neste estudo.

Essas questões de pesquisa foram detalhadas em questões de pesquisas específicas em cada estudo relacionado, os quais foram descritos nos capítulos destinados a cada um deles.

1.3 PUBLICAÇÕES

Um dos resultados obtidos por esta pesquisa de doutorado é a publicação de artigos contendo os resultados alcançados. Até o momento, o trabalho abaixo foi publicado no 32º Simpósio Brasileiro de Engenharia de Software (São Carlos, 2018) e foi premiado como o segundo melhor artigo na trilha principal desse evento. Ele está relacionado com a primeira questão de pesquisa, apresentando os resultados obtidos com a mineração de repositório de código-fonte:

MOTTA, Tiago Oliveira; GOMES E SOUZA, Rodrigo Rocha; SANT'ANNA, Claudio. Characterizing architectural information in commit messages: an exploratory study. In: Proceedings of the XXXII Brazilian Symposium on Software Engineering. 2018. p. 12-21.

A próxima seção apresenta as contribuições desta tese.

[2]No contexto desta tese, o termo versão foi trabalhado no sentido da versão do código-fonte do projeto disponível após cada *commit* realizado. Dessa forma, as versões não são necessariamente numeradas. As versões do projeto que possuem numeração própria, são tratadas como *releases*.

1.4 CONTRIBUIÇÕES DESTA TESE

Cada questão de pesquisa que estruturou esta tese trouxe contribuições para a pesquisa em Engenharia de Software. Destaque-se as seguintes:

- Identificação e caracterização de fonte de informações sobre mudanças arquiteturais;
- Proposição de taxonomia para tópicos arquiteturais baseados nas definições clássicas de arquitetura de software;
- Caracterização de mudanças arquiteturais em projetos de Software Livre quanto a:
  - Perfil de colaboradores que modificam a arquitetura de software;
  - Períodos que concentram mudanças na arquitetura;
  - Tópicos arquiteturais que mais sofrem modificações;
  - Abrangência (em número de arquivos/módulos) das modificações arquiteturais;
- Caracterização das mudanças provocadas sobre o código do projeto após mudanças arquiteturais a partir de:
  - Métricas de tamanho do código;
  - Métricas de complexidade do código;
  - Métricas de acoplamento do código, e;
  - Métricas de coesão do código.
- Pacote experimental para a análise de código-fonte pós mudanças arquiteturais;
- Pacote experimental para mineração de mensagens de *commits* com informação arquitetural;
- Listagem de dificuldades encontradas por desenvolvedores e pesquisadores no tocante à documentação arquitetural, e;
- Listagem de estratégias, métodos e ferramentas utilizadas por desenvolvedores e pesquisadores no processo de busca por informações arquiteturais.

1.5 ORGANIZAÇÃO DESTA TESE

Esta tese foi organizada de forma a apresentar os trabalhos que a compõem, e sua estrutura está resumida na 1.1. Dessa forma, os capítulos foram assim organizados: O Capítulo 1 apresenta as definições iniciais e o contexto deste trabalho, assim como o problema e as questões de pesquisa. No Capítulo 2, são apresentados os fundamentos teóricos relacionados a esta tese, nele são apresentados (i) conceitos de arquitetura de software, além de estudos sobre documentação arquitetural e recuperação de informações sobre arquitetura; (ii) conceitos sobre repositórios de software livre; (iii) conceitos e estudos sobre a mineração de repositórios e recuperação da informação em geral, e;

(iii) apresentação das métricas de código-fonte utilizadas durante essa pesquisa. Os três capítulos seguintes apresentam os estudos realizados ao longo desse processo de doutoramento. Em cada capítulo, foram apresentados os antecedentes, os trabalhos relacionados, o design do estudo, os resultados, as ameaças à validade e as conclusões de cada um.

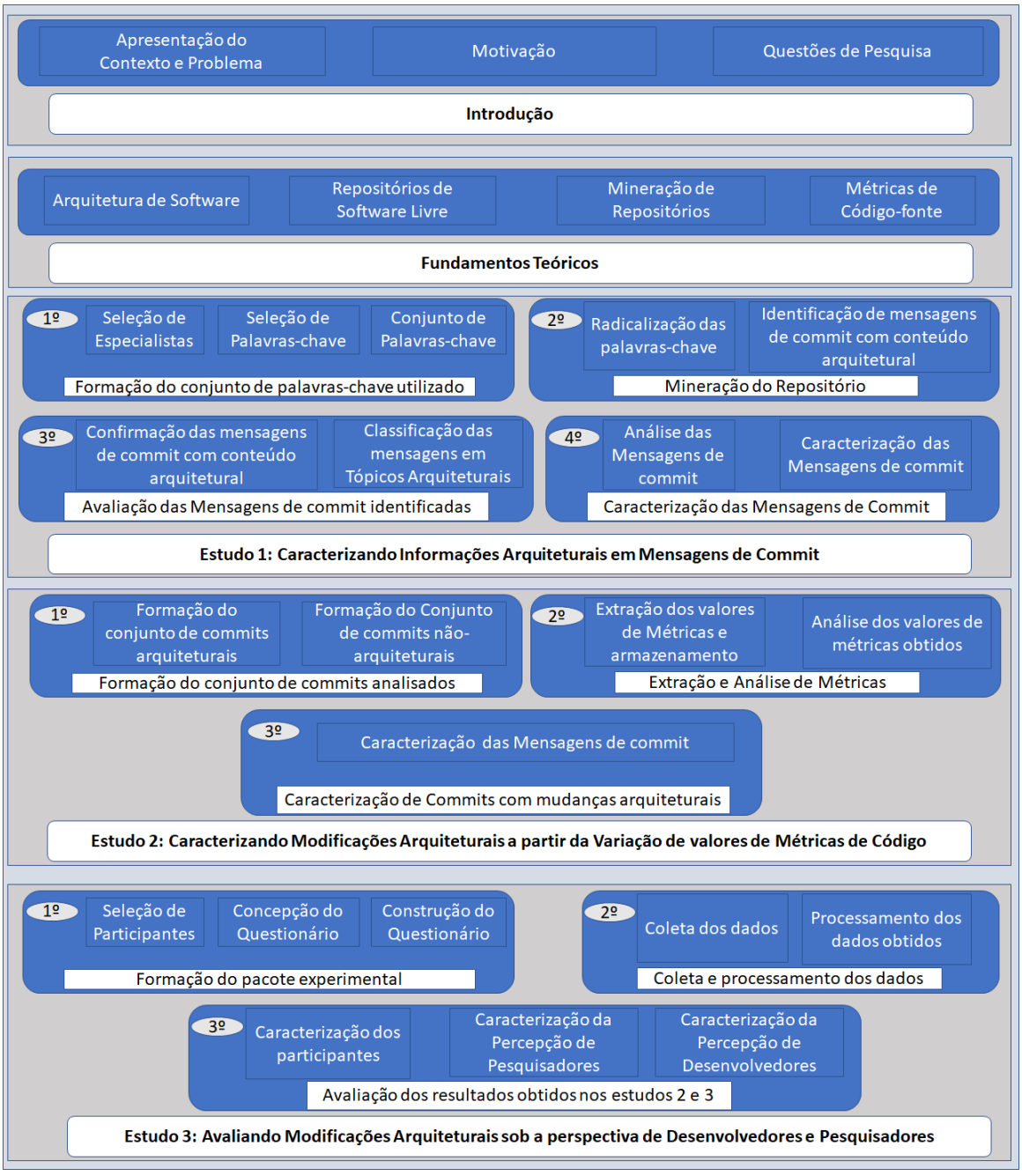


Figure 1.1: Organização desta tese.

Seguindo esse rito, no Capítulo 3, é apresentado o estudo realizado envolvendo a

mineração de repositórios, explorando comentários de *commits* do projeto KDELibs. Esse capítulo está relacionado à questão de Pesquisa **RQ1**. No Capítulo 4, é apresentado o estudo realizado, envolvendo os *commits* arquiteturais identificados durante a mineração do projeto KDELibs, buscando caracterizar as mudanças ocorridas no projeto, utilizando métricas de tamanho, complexidade, acoplamento e coesão de código. Esse capítulo está relacionado à **RQ2**. O Capítulo 5, por sua vez, apresenta o estudo empírico que objetivou apresentar os resultados desse estudo à pesquisadores e desenvolvedores e diante disso coletar suas impressões acerca da aplicabilidade e utilidade desses resultados em sua prática profissional. Ele está relacionado à **RQ3**. Por fim, o Capítulo 6 apresenta as considerações finais deste trabalho.

Chapter 2

FUNDAMENTOS TEÓRICOS

Os desenvolvedores têm percebido que a documentação dos softwares desenvolvidos é um item importante para o processo de compreensão e evolução do produto. Essa constatação ocorre porque sua evolução é nata e ocorre de acordo com a demanda de seus consumidores. Além disso, esse processo evolutivo, com frequência, inclui a alteração de diversas partes do projeto e o incremento de novas funcionalidades. No entanto, pesquisas em repositórios de projetos de Software Livre mostram que a documentação da arquitetura do projeto não tem obtido a mesma atenção (DING et al., 2014) que o código-fonte do produto. Nesse sentido, esse capítulo apresenta fundamentos teóricos relacionados aos tópicos tratados no âmbito desta pesquisa: (i) Arquitetura de software e sua documentação, (ii) Repositórios de Software Livre, (iii) Mineração de repositórios, e (iv) Métricas de código-fonte.

2.1 ARQUITETURA DE SOFTWARE

Arquitetura de software é uma abstração do projeto de software de alto nível que define, através de restrições e induções, os relacionamentos entre unidades de código-fonte de um projeto, a exemplo de classes, pacotes, componentes ou módulos. Há um conjunto de definições para arquitetura de software registradas na literatura. No escopo deste trabalho, serão consideradas as três definições clássicas: Perry e Wolf (1992), Garlan e Shaw (1993) e Bass, Clements e Kazman (1998).

Para Perry e Wolf (1992), arquitetura de software é o conjunto de elementos arquiteturais que possuem alguma organização. Esses autores classificaram os elementos arquiteturais de três formas: Elementos de Processamento, Elementos de Dados e Elementos de Conexão. Os elementos e a sua organização são definidos por decisões tomadas para satisfazer objetivos e restrições.

Já Garlan e Shaw (1993) justificam o seu trabalho sobre arquitetura de software, relatando que a mesma se torna necessária, conforme o tamanho e a complexidade dos sistemas de software crescem. Dessa forma, a construção de sistemas transcende a escolha dos algoritmos e estruturas de dados adequados ao problema. No planejamento, também

serão incluídas decisões sobre as estruturas que comporão o sistema, a estrutura global de controle, os protocolos de comunicação, os mecanismos de sincronização e acesso a dados, a atribuição de funcionalidade a elementos do sistema ou ainda sobre distribuição física dos elementos do sistema. Além disso, será necessário envolver decisões que se refletirão em atributos de qualidade como escalabilidade e desempenho, entre outros, (GARLAN; SHAW, 1993).

Bass, Clements e Kazman (1998), por sua vez, definem que a arquitetura de um programa, ou de sistemas computacionais, é a estrutura ou estruturas do sistema, as quais são compostas por elementos de software, as propriedades externamente visíveis desses elementos e os relacionamentos entre eles.

Dentre as definições mais utilizadas e conhecidas, o IEEE apresentou, através da norma ISO/IEC/IEEE 1471-2000, um quarto conceito, dentre os vários disponíveis, para arquitetura de software o qual define a arquitetura de software como a organização fundamental de um sistema incorporado, em seus componentes, seus relacionamentos com o ambiente e com os princípios norteadores de design e evolução (MAIER; EMERY; HILLIARD, 2001).

A partir das definições clássicas, apresentadas acima, é de se presumir que, como esses elementos guiam o desenvolvimento do projeto de software, a sua documentação é parte importante do processo de desenvolvimento. Como já afirmado, no contexto de projetos de Software Livre, Ding et al. (2014) apontaram que a documentação arquitetural desses projetos é um artefato escasso em seus repositórios. O que em si surpreende, pois, como é de conhecimento geral, a omissão desses artefatos pode culminar em projetos diferentes do projetado inicialmente, e trazer resultados indesejáveis aos seus desenvolvedores (GURP; BOSCH, 2002; FARID; AZAM; IQBAL, 2011; BEHNAMGHADER et al., 2017; KLEEBAUM et al., 2018). As questões relacionadas à documentação arquitetural são tratadas na próxima subseção.

2.1.1 Documentação Arquitetural

A documentação da arquitetura de um software ainda não possui uma linguagem unificada que seja capaz de abarcar todas as facetas necessárias e, dessa forma, incentivar os desenvolvedores e arquitetos a utilizarem-na. No entanto, a literatura aponta que ter acesso à evolução arquitetural de um projeto e a lógica, por trás dessa evolução, melhora a compreensão e a rastreabilidade do projeto (DUTOIT; PAECH, 2001; BACHMANN et al., 2005).

Alguns anos antes, Kogut e Clements (1995) apresentaram uma lista de questões a serem compreendidas para que se chegue a uma linguagem de documentação arquitetural unificada: Quais são os componentes? Eles são módulos que existem apenas em tempo de design, mas são compilados juntos antes de tempo de execução? Eles são tarefas ou processos dispersos em diferentes módulos, montados em tempo de compilação e formam unidades em tempo de execução? O que os componentes fazem? Como se comportam? De quais outros componentes eles dependem? O que as conexões significam? Será que elas significam “envia dados para”; “envia controle para”; “Chamadas”; “é uma parte de”; alguma combinação destes, ou algo mais? Quais são os mecanismos utilizados para

satisfazer essas relações? Em um trabalho posterior, Clements (1996) realizou um estudo buscando caracterizar Linguagens de Descrição Arquitetural (ADL) e construiu um mapa contendo as diferenças perceptíveis entre um conjunto de ADLs analisadas e o que deve conter em uma ADL[1]. Num esforço adicional, em prol da utilização de ADLs, Malavolta manteve o projeto *Architectural Languages Today*[2] contendo a lista das Linguagens de Descrição Arquitetural disponíveis até então, indicando o link para essas linguagens, o trabalho acadêmico que o descreve (quando há), se está em desenvolvimento ativo, se está disponível e se é comercial.

Em busca de convergência e construção de uma linguagem de descrição arquitetural, Graaf et al. (2014) indicou que a solução para a construção de uma ADL universal passa pela adesão de ontologias, pois isso incentivaria a documentação e tornaria possível documentar todas as facetas da arquitetura de software. Reforçando a necessidade de que a documentação arquitetural deve acompanhar a evolução do projeto, Díaz-Pace et al. (2016) propuseram um método de documentação arquitetural incremental e centrado nas necessidades dos envolvidos no projeto.

Uma vez superados os entraves para a documentação da arquitetura dos sistemas, espera-se que esse artefato passe a ser parte integrante dos projetos de software desenvolvidos e que o momento seguinte seja de inclusão de uma rotina de atualização dessa documentação que acompanhe o processo de evolução do sistema. Enquanto isso, iniciativas para a construção de métodos para a recuperação arquitetural se avolumam na literatura e os principais trabalhos nesse sentido são apresentados na subseção a seguir.

2.1.2 Recuperação de Informações sobre a Arquitetura

Diante da dificuldade relatada ao longo deste texto, acerca da disponibilidade de documentação arquitetural, pesquisadores da área de arquitetura de software têm proposto técnicas para recuperar informações sobre a arquitetura de um software e suas mudanças ao longo da evolução dos projetos.

A maior parte dessas técnicas é baseada na análise estática de código e traz a parte estática da arquitetura dos projetos. A Tabela B.1 mostra um conjunto dessas técnicas e os trabalhos publicados quando da sua apresentação à comunidade científica. Dentre as representações disponíveis encontram-se as Matrizes Estruturais do Projeto (HUYNH et al., 2008), os grafos de dependência (EBERT et al., 2002; MALTON, 2005) e os mapas conceituais (CLEARY; EXTON, 2005). Essas técnicas, no entanto, trazem aos seus usuários detalhes estruturais do projeto, mas não as decisões tomadas ao longo da evolução dos projetos, objetivos, restrições e motivações, mecanismos de sincronização, requisitos não-funcionais, mecanismos de acesso aos dados e outras facetas do projeto arquitetural.

Hesse et al. (2016) apontaram que ferramentas de *issue trackers* possuem informações sobre as decisões tomadas ao longo do projeto e, com elas, buscaram uma estratégia

[1] O resultado deste trabalho está resumido na Tabela B.1 do Apêndice B

[2] Esse projeto foi descontinuado em virtude do ao coordenador da pesquisa ter ido trabalhar em outra universidade, mas está disponível para acesso via Archive.org (última atualização em 29/01/2019). Disponível em: https://web.archive.org/web/20190129121836/http://www.di.univaq.it/malavolta/al/

dominante para a documentação desse tipo de informação. Nesse estudo, eles fizeram a interpretação de 260 *issues* do projeto Firefox e concluíram que lá estão disponíveis informações sobre a tomada de decisões no projeto e que a maior parte das issues está relacionada à documentação e solução de problemas reportados. Kleebaum et al. (2018) traçaram uma estratégia para recuperar as decisões tomadas ao longo da evolução de projetos de software baseada na organização de pacotes do projeto, nas ramificações e integrações registradas no sistema de controle de versões, além de atualizações nas tarefas pendentes a partir de *commits* de código e de *issue trackers*.

Rogers et al. (2015) aplicaram algoritmos de classificação sobre ferramentas de *bugs report* em busca dessas decisões ao longo do projeto, usando as ferramentas GATE (CUNNINGHAM, 2002) e Weka (HALL et al., 2009) sobre o repositório de bugs do projeto Chrome e capturaram informações sobre requisitos, decisões, alternativas, argumentos, premissas, dúvidas, respostas e procedimentos. Shahbazian et al. (2018), por sua vez, estudaram o impacto das decisões tomadas a nível de design sobre a arquitetura do projeto a partir da análise de repositórios, utilizando a ferramenta *RecovAr* sobre os projetos *Hadoop* e *Struts* e apresentam uma metodologia para recuperar decisões arquiteturais. Vale lembrar que Hassan (2008) já havia mapeado possíveis fontes de informação não-estruturadas, para que se procurem os registros de decisões tomadas ao longo da evolução dos projetos (além de outras informações sobre o projeto). Nesse trabalho, ele indica o tipo de informação disponível em repositórios de informações e aponta como fontes potenciais repositórios de projetos, em geral, (código-fonte, bugs e comunicação do projeto), logs de desenvolvimento e repositórios de projetos de Software Livre disponíveis. A próxima subseção trata da área de conhecimento arquitetural, algo que como se viu durante esta subseção, é relevante para a compreensão da arquitetura, mas ainda não está amadurecida nem na pesquisa nem no uso na indústria.

2.1.3 Conhecimento Arquitetural

A importância do conhecimento arquitetural tem ganhado destaque na pesquisa em Engenharia de Software nos últimos anos. Essa subárea de conhecimento tem estudado diversos aspectos que vão além das informações estruturais sobre a arquitetura. Weinreich e Groher (2016) apresentaram uma revisão sistemática de literatura que traz à tela abordagens desenvolvidas pela pesquisa em Engenharia de Software acerca do gerenciamento do conhecimento em Arquitetura de Software. Nesse trabalho, os autores apontaram que a maioria das abordagens é direcionada para o uso do conhecimento já adquirido, enquanto a aquisição desse conhecimento é um assunto ainda em aberto. Como conclusões, os autores destacaram que os maiores desafios percebidos nessa área são a aquisição e manutenção do conhecimento arquitetural adquirido ao longo da evolução dos projetos.

Em outro estudo sobre o assunto, Capilla et al. (2016) apontam como os principais desafios dessa área a utilização prática das informações recuperadas no trabalho da indústria, formação de um conhecimento amplo e interligado (desde à captura, passando pela gestão e utilização das informações arquiteturais) e, a validação do que foi proposto no contexto industrial. Para isso foram analisados estudos realizados nos dez anos anteriores à publicação desse estudo que estivessem relacionados à aquisição, uso, gerenciamento

e compartilhamento de conhecimentos arquiteturais.

Num outro sentido, Tang et al. (2017) apontam que os fatores humanos - como a cognição, os comportamentos e as interações em grupo - devem ser incluídos no processo de aquisição e gerenciamento de conhecimento arquitetural. Através de uma revisão de literatura sobre tomada de decisão de arquitetura de software, eles indicaram que essa área necessita priorizar a formulação de: heurísticas de tomada de decisão para lidar com a complexidade do projeto, mecanismos para a contenção de viéses cognitivos e ferramentas para visualização de insights que ajudem os projetistas a superarem as próprias limitações cognitivas.

A próxima seção trata dos conceitos que cercam os repositórios de projetos de Software Livre, apresentando uma visão geral e em seguida descrevendo o projeto utilizado nesta tese.

2.2 REPOSITÓRIOS DE SOFTWARE LIVRE

Os repositórios de projetos de Software Livre são uma importante fonte de informação para o estudo de projetos de software e sua evolução. Balieiro et al. (2007) já havia noticiado que os projetos de Software Livre e seus repositórios são alvos de estudos em diversas áreas da computação. Assim, dada a disponibilidade de informações em projetos de Software Livre, o estudo de mudanças ao longo de sua evolução torna-se possível em diversas dimensões e abstrações, a exemplo da Arquitetura de Software.

2.2.1 Visão Geral

Um repositório de Software Livre é um espaço virtual onde são reunidos artefatos de um projeto de Software. Como artefatos, estão incluídos todos os subprodutos do projeto de software, aí incluídos desde o código-fonte, passando pela documentação (como diagramas de UML ou *wikis* para desenvolvedores e usuários do projeto) até códigos de testes. Kagdi, Collard e Maletic (2007) classificaram diferentes tipos de repositórios de software, indicando quais teriam maior potencial de mineração para o estudo da evolução desses. Estão aí incluídos os repositórios de código-fonte. Esses repositórios além de conter toda a evolução do produto, armazenam outros artefatos anexos, tão ricos de informação quanto o próprio código-fonte. Um exemplo desses artefatos anexos são os comentários de *commit*.

Para cada nova modificação inserida no repositório de código-fonte, por parte dos colaboradores do projeto, são incluídas informações sobre o que foi modificado na versão *commitada* e muitas vezes as justificativas para essas modificações. Isso permite que se explore tanto o código-fonte (e se perceba os impactos das modificações) quanto as justificativas para as modificações. Essas possibilidades estão atreladas à existência de um Sistema de Controle de Versões (SCV), um software que gerencia as mudanças ocorridas em cada versão depositada no espaço virtual (repositório). Exemplos de SCVs são o Git[3],

[3]https://git-scm.com/

o CVS[4] ou o Subversion[5].

Esses repositórios possuem uma política de utilização definida por cada projeto. Essa política inclui o lapso temporal para o lançamento de novas *releases* do projeto, os critérios para a numeração de cada *release* liberada à comunidade, o tipo de modificação que cada *release* deve conter, quem pode inserir modificações diretamente no repositório, quais colaboradores devem verificar as contribuições e adicioná-las no fluxo principal do projeto, dentre outras. A próxima subseção apresenta o projeto, e seu respectivo repositório, utilizado no escopo desta pesquisa de doutorado: O projeto KDELibs.

2.2.2 O Projeto KDELibs

O projeto KDELibs é um projeto de Software Livre, que por 21 anos foi a biblioteca central do projeto KDE. O KDE[6] nasceu como um ambiente de trabalho em 1996 por iniciativa de Matthias Ettrich[7], e se tornou um hub de desenvolvimento que fornece recursos e ferramentas que permitem o trabalho colaborativo. Ele inclui desde uma interface gráfica, passando por ambientes de desenvolvimento e aplicações simples (como editor de texto e calculadora), até ambientes de trabalho para escritório (como o KOffice). Trata-se de um ecossistema de Software (NEU et al., 2011) que está disponível no formato de Software Livre. Para o participar do ecossistema KDE, é preciso aderir ao Manifesto KDE[8] e seguir as regras gerais de participação (como ter uma documentação voltada ao usuário, hospedar o projeto no servidores do projeto, etc).

Como parte do ecossistema KDE, a biblioteca KDELibs tem disponível seu processo evolutivo desde 1997 (o *commit* inicial data de 14/04/1997) e, até a disponibilização do framework KF5[9] (um conjunto de aproximadamente 80 bibliotecas oriundas da KDELibs, mas com um novo projeto arquitetural) como sua substituta, foi a concentradora das principais funções e interfaces que compunham as aplicações do KDE. O projeto ainda registra alterações, pois ainda há usuários que utilizam versões anteriores desse ecossistema. Esse projeto (KDELibs) possui um repositório próprio de código-fonte que utiliza como VCS o Git. Durante o período de análise do repositório desse projeto, cerca de 700 colaboradores haviam registrado *commits*, realizado revisões e feito contribuições sobre o mesmo.

Essa biblioteca teve 5 *releases* lançadas entre 14/08/1998 e 11/01/2008. Cada uma dessas *releases* contém identificação em três níveis. Como exemplo, suponha uma *release* numerada como X.Y.Z. A *release* com variação em "Z" é a de correção de bugs, a *release* com variação em "Y" reflete pequenas variações arquiteturais e acréscimo de novas *features* do projeto, já a *release* com variação em "X" reflete grandes alterações no projeto, incluido aí a arquitetura do projeto. As *releases* com as correções de bugs devem ter a periodicidade mensal, as que contém acréscimo de novas funcionalidades quadrimensal e as *releases* com grandes alterações no projeto (no exemplo, as versões "X") atendem a um

[4]https://www.nongnu.org/cvs/
[5]https://subversion.apache.org/
[6]https://kde.org/community/whatiskde/
[7]https://groups.google.com/g/comp.os.linux.misc/c/SDbiV3Iat_s/m/zv_D_2ctS8sJ
[8]https://manifesto.kde.org/
[9]https://kde.org/announcements/kde-frameworks-5.0/

calendário definido no Akademy, um evento anual que reúne a comunidade mundial do KDE. Na programação estão incluídos uma conferência, um período para a codificação do projeto e a reunião de membros para a discussão sobre o projeto[10]. É importante destacar que o ecossistema KDE conta com um sistema de integração contínua[11].

Ante a organização do projeto, é possível assumir que os repositórios do KDELibs são alvos em potencial na busca de informações não disponíveis nas páginas do projeto. A mineração desse repositório é uma técnica que pode ser utilizada nesse sentido. Dessa forma, são apresentados os fundamentos sobre mineração de repositórios na próxima subseção.

2.3 MINERAÇÃO DE REPOSITÓRIOS

2.3.1 Visão Geral

Durante o ciclo de desenvolvimento de software, muitas informações são geradas, mas a diversidade e riqueza dessas informações são pouco exploradas. Dados sobre os ajustes necessários ao produto, motivação para alterações, implementação de novas funcionalidades e decisões sobre o padrão de desenvolvimento são exemplos de informações geradas nesse processo que não são utilizadas em geral. A busca por essas informações envolve tanto o reconhecimento de padrões quanto o conhecimento do que se deseja resgatar dentro dessas fontes. Repositórios de código-fonte e de *bugs*, comentários de códigos-fontes, listas de discussão de projetos, registros de canais de discussão de projetos como *Internet Relay Chat* (IRC) e *issues trackers* são potenciais fontes para que se resgate o histórico do projeto e com ele decisões tomadas ao longo do desenvolvimento.

Dessa forma, a pesquisa da Engenharia de Software tem investido na busca dessas informações através da aplicação de técnicas de mineração e se concentrado na aquisição através de fontes de dados não-estruturados a partir de artefatos com função inicialmente limitada, mas com dados com potencial exploratório. Nesse sentido, técnicas para a construção de conhecimento, baseadas nessas informações, têm sido desenvolvidas e aprimoradas, tais como métodos para a separação de interesses com foco na concentração de palavras-chave em artefatos desenvolvidos por Ohba e Gondow (2005), estudos que relacionam a presença de bugs e mudanças a partir da análise do histórico de versões, conforme pesquisado por Śliwerski, Zimmermann e Zeller (2005), métodos que conseguem estabelecer associações através da análise das informações e aplicação de métodos probabilísticos como o Latent Dirichlet Allocation - LDA - desenvolvido por Blei, Ng e Jordan (2003) ou estudos para análise de fontes de informação baseada em abordagens derivadas de filtros de Spam como a pesquisa desenvolvida por Mizuno et al. (2007).

É importante citar outras associações estudadas, até então, usando a mineração de dados, como a sucessão de mudanças em artefatos de um projeto proposta por Pattison, Bird e Devanbu (2008), estudos sobre a participação de desenvolvedores em projetos de Software Livre como os de Shihab, Zhen e Hassan (2009) ou o questionamento sobre a qualidade da descrição do trabalho realizado como relatado no estudo de Maalej e Happel

[10] https://ev.kde.org/akademy/

[11] https://community.kde.org/Infrastructure/Continuous_Integration_System

(2010). A literatura ainda apresenta estudos com análise de *bugs trackers* em busca de predizer defeitos no software produzido como em Kim et al. (2007), Moser, Pedrycz e Succi (2008) e Eaddy et al. (2008). Esses estudos consideram além do código-fonte atual de cada projeto, o histórico evolutivo dos projetos em estudo. Outro tipo de mineração, já realizado, é a busca por decisões tomadas ao longo do projeto, como nos trabalhos de Hesse et al. (2016), Kleebaum et al. (2018) e Alkadhi et al. (2017).

2.3.2 Busca de elementos em fontes de documentação não-estruturada

A pesquisa de informações em fontes de dados não-estruturados apresenta alguns desafios ainda não superados pela Engenharia de Software. Atualmente, os pesquisadores têm buscado nessas fontes informações dados não disponíveis na documentação dos projetos desenvolvidos.

As estratégias para busca têm variado ao longo do tempo, abaixo é relacionado um conjunto de técnicas e métodos utilizado para fins de recuperação da informação:

- **Extração de Comentários de Códigos**: Hirata e Mizuno (2011) estudaram os comentários de códigos-fonte e relacionaram a qualidade dos comentários com propensão a *bugs*. A técnica de qualificação dos comentários está na geração de *tokens*, as quais correspondem às palavras presentes nos comentários, separadas por caracteres especiais como espaço, vírgulas, chaves e outros.

- **Separação de dados estruturados de dados não-estruturados**: Merten, Mager e Paech (2014) propuseram um método para a separação de dados estruturados (de linguagens de programação, por exemplo) de dados não-estruturados em fontes, em que esses são encontrados de forma conjunta. O método desenvolvido envolve a indicação de palavras-chave utilizadas nas informações técnicas (como palavras reservadas de dez linguagens de programação), a busca de linhas parecidas, usando lógica difusa, a busca de patches parecidos com uso de lógica difusa e a detecção de ocorrências com o uso de expressões regulares.

- **Framework de Análise de IRC**: Nessa técnica, Shihab, Zhen e Hassan (2009) se propõem a análise de um conjunto de reuniões realizadas entre a equipe de um projeto de Software Livre via *Internet Relay Chat* (IRC), e a partir dessa análise, inferir questões relativas à participação no projeto analisado. Essa proposta prevê a coleta dos dados, uma inspeção manual para classificação dos tipos de dados obtidos, a identificação dos dados e o armazenamento das informações para comparação posterior.

- **Recuperação de descrição de trabalho**: Nessa proposta, Maalej e Happel (2010) discutem se é possível recuperar descrições de trabalho explorando textos disponíveis em *commits*, ordens de serviço, logs de uso de IDEs, comentários de código. Para tanto, foi utilizado um método para classificar as palavras conforme os seus tipos gramaticais (substantivos, verbos, advérbios), a fim de terem a sua frequência analisada e, a partir disso, serem submetidas a uma ontologia que identifica quais registros se referem às descrições de tarefas.

- **Extração em Repositório de Código-fonte**: Mizuno et al. (2007) analisaram códigos-fonte defeituosos utilizando técnicas para a definição de filtros de spam. Assim eles elaboraram uma técnica que aprende padrões de código mais suscetíveis a defeitos. Eles utilizaram a ferramenta de detecção de SPAM CRM114 e compararam o resultado obtido, utilizando três técnicas disponíveis: Sparse Binary Polynomial Hash Markov model (SBPH), Orthogonal Sparse Bigrams Markov model (OSB) e Simple Bayesian model (BAYES) para atestar a funcionalidade e eficiência da ferramenta desenvolvida.

As técnicas apresentadas acima são utilizadas de diversas formas e sobre diversas fontes de dados. Elas são empregadas, principalmente, sobre fontes de dados, onde se registram informações de forma não-estruturada e, às vezes, de modo informal e buscam informações não-documentadas ao longo da evolução dos projetos. É comum o uso dessas ferramentas nas tentativas de associar determinado fenômeno (como a ocorrência de *bugs*) com ações e registros despretensiosos nessas fontes de dados.

Como se vê, várias abordagens estão disponíveis e, em sua essência, encontra-se a necessidade de processar informações sobre as quais se sabe pouco e, por isso, a construção dessas associações e estratégias de associações faz-se necessária. Outro tipo de informação que ajuda a caracterizar um projeto são as informações sobre seu tamanho, acoplamento, coesão e complexidade. Essas informações são passíveis de extração a partir de métricas de código-fonte. A próxima seção apresenta uma breve apresentação sobre essa temática.

2.4 MÉTRICAS DE CÓDIGO-FONTE

Métrica de software é uma medida numérica que pode ser extraída de um produto de software (INCE; SHEPPARD, 1988). Sommerville (2011) afirma que "(...) a medição de software preocupa-se com a derivação de um valor numérico ou o perfil para um atributo de um componente de software, sistema ou processo". Assim sendo, as métricas permitem a comparação desses valores entre si e com outros projetos de mesmo domínio e, assim, ser capaz de tirar conclusões sobre a qualidade do software ou avaliar a eficácia dos métodos, das ferramentas e dos processos de software (SOMMERVILLE, 2011). Dadas essas características, as métricas possuem potencial para expandir a caracterização de diferentes tipos de mudanças em projetos de software. A seguir, serão apresentadas um conjunto de métricas a serem utilizadas no escopo deste estudo, classificadas em 4 tipos: métricas de tamanho, métricas de complexidade, métricas de acoplamento e métricas de coesão.

2.4.1 Métricas de Tamanho

Métricas de tamanho são aquelas que quantificam o tamanho de um software ou de partes dele. Elas se fazem necessárias desde o início da computação, pois, nos primórdios da computação, o software ocupava espaço físico (cartões perfurados) sendo, portanto, necessário ter-se uma estimativa dessa natureza. As métricas de tamanho também estão relacionadas aos padrões de qualidade e processos de certificação de processos de desenvolvimento (LINDERS, 2011). Nas próximas subseções, são apresentadas as

métricas de tamanho coletadas nessa pesquisa.

Média de Linhas de Código por Método (AMLoc)
A métrica Média de Linhas de Código por Método (AMLoc) representa o tamanho de uma classe/módulo a partir da média de linhas de código para cada função/método presentes no módulo/arquivo analisado (LORENZ; KIDD, 1994). A Figura 2.1 apresenta uma avaliação do tamanho do módulo "`kconf_update`" do projeto KDELibs, usando a métrica AMLoc.

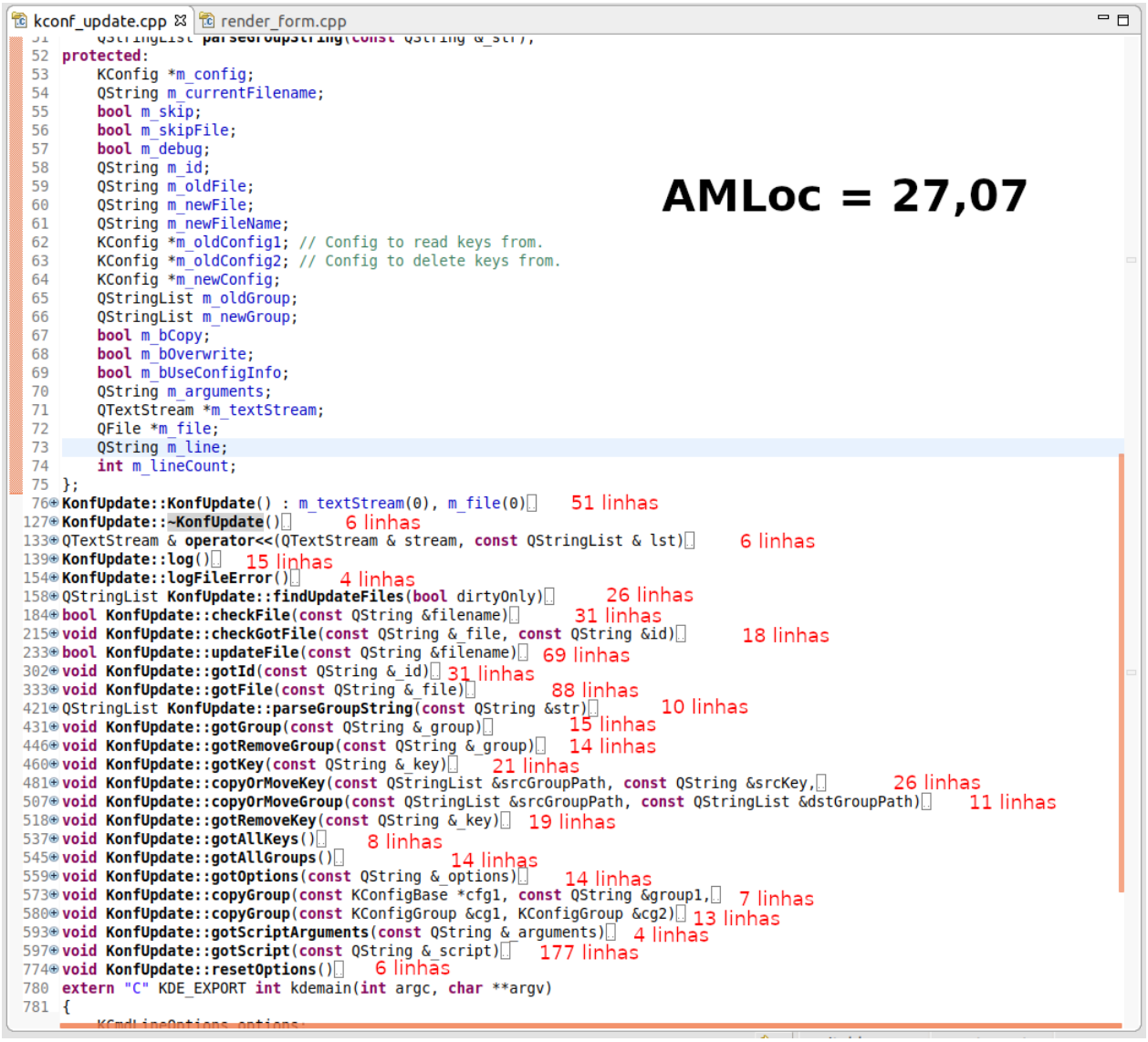


Figure 2.1: Exemplo de cálculo do valor da métrica AMLoc.

A equação que define o valor da métrica AMLoc é: A razão entre os valores V (quantidade de linhas de código) dos métodos M presentes na classe C e a quantidade

de métodos (n) dessa classe (C): $\forall M \in C$,

$$AMLoc = \frac{\sum_{i=1}^{n} V(M_i)}{n}$$

Número Médio de Parâmetros (ANPM)
A métrica ANPM é uma medida de tamanho que considera a quantidade de parâmetros presentes nas assinaturas dos métodos/funções das classes e módulos analisados. Ela reflete a razão entre a soma da quantidade de parâmetros em cada método/função de uma classe/módulo e a quantidade de métodos/funções desse módulo/classe (BASILI; ROMBACH, 1987). A Figura 2.2 apresenta uma avaliação do código-fonte do módulo "`kconf_update`" do projeto KDELibs para a métrica ANPM. Seu valor é definido pela seguinte equação: Seja V o valor (quantidade de parâmetros) de um método/função M presente na classe/módulo C que contém n métodos, tem-se: $\forall M \in C$,

$$ANPM = \frac{\sum_{i=1}^{n} V(M_i)}{n}$$

Linhas de Código (LOC)
A métrica Linhas de código é a mais antiga e mais utilizada métrica de código, e representa o tamanho de um módulo/classe através do número de linhas de código de um módulo/classe (BOEHM, 1984). Ela contém variações, como a SLOC, LLC e ELOC, dentre outras. A métrica SLOC contabiliza apenas as linhas efetivas de instruções (excluindo linhas em branco e de comentários). Outra variação é a métrica Linhas Físicas de Código (LFC), que consideram todas as linhas num arquivo de código-fonte. Já a variação Linhas Lógicas de Código (LLC) considera apenas as instruções efetivas (linhas muito grandes que foram divididas em duas para ajudar na legibilidade do código são consideradas como apenas uma). Por fim, a variação Linhas Efetivas de Código (ELOC), desconsidera linhas contendo apenas delimitadores (como as chaves), as linhas em branco e linhas com apenas comentários (SHEPPERD, 1990). A Figura 2.3 apresenta um exemplo de definição de SLOC usando o módulo "kconf_update" do projeto KDELibs.

Número de Atributos (NOA)
Outra métrica de tamanho disponível é o Número de Atributos. Ela é valorada a partir da quantidade de atributos em uma classe/módulo (HENDERSON-SELLERS, 1995). A Figura 2.4 apresenta um exemplo de cálculo da métrica NOA, usando o módulo "`kconf_update`" do projeto KDELibs.

Número de Métodos (NOM)

A métrica Número de Métodos também valora o tamanho de uma classe/módulo e tem seu valor mensurado a partir da quantidade de métodos que compõem uma classe/módulo (HENDERSON-SELLERS, 1995). A Figura 2.5 apresenta um exemplo de cálculo da métrica NOM, usando o módulo "`kconf_update`" do projeto KDELibs.

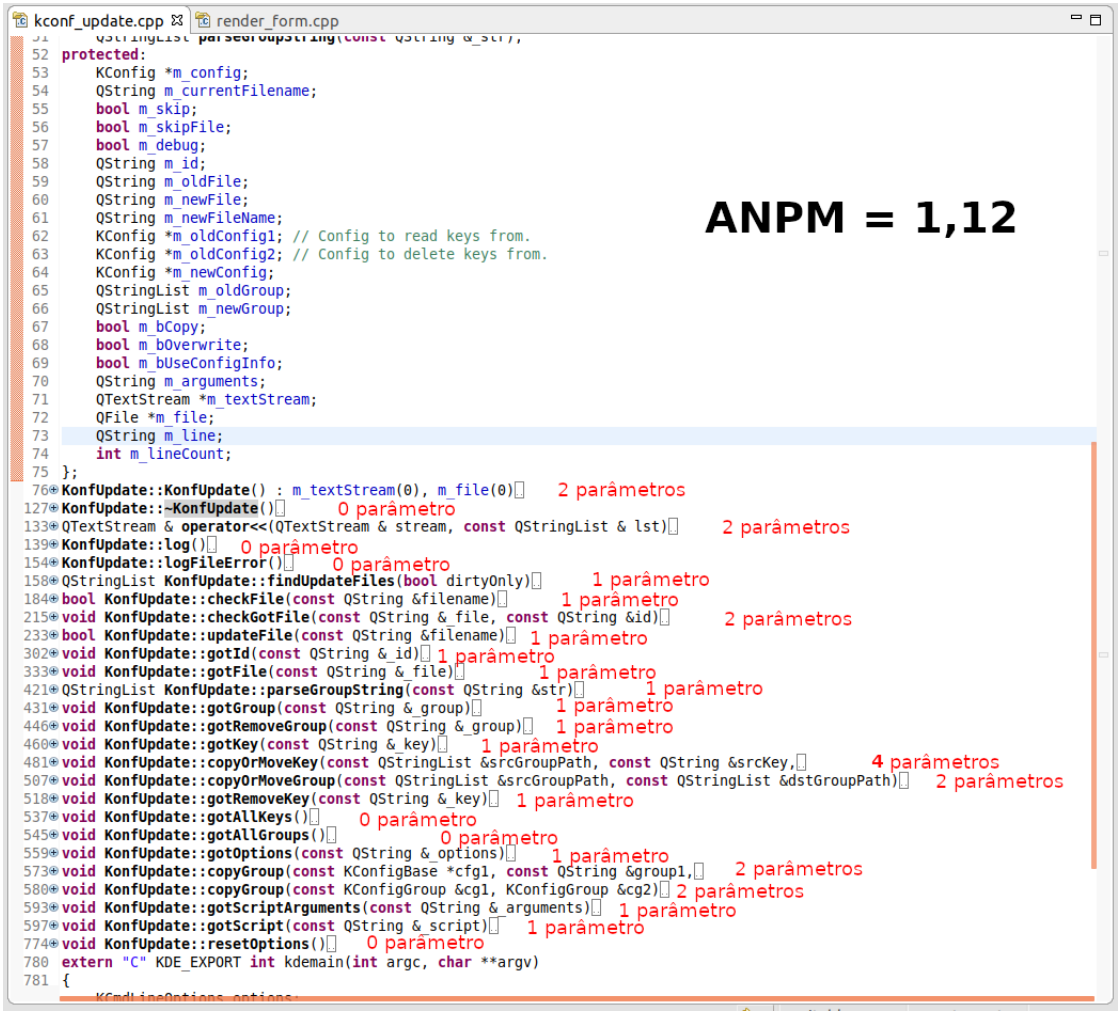


Figure 2.2: Exemplo de cálculo do valor da métrica ANPM.

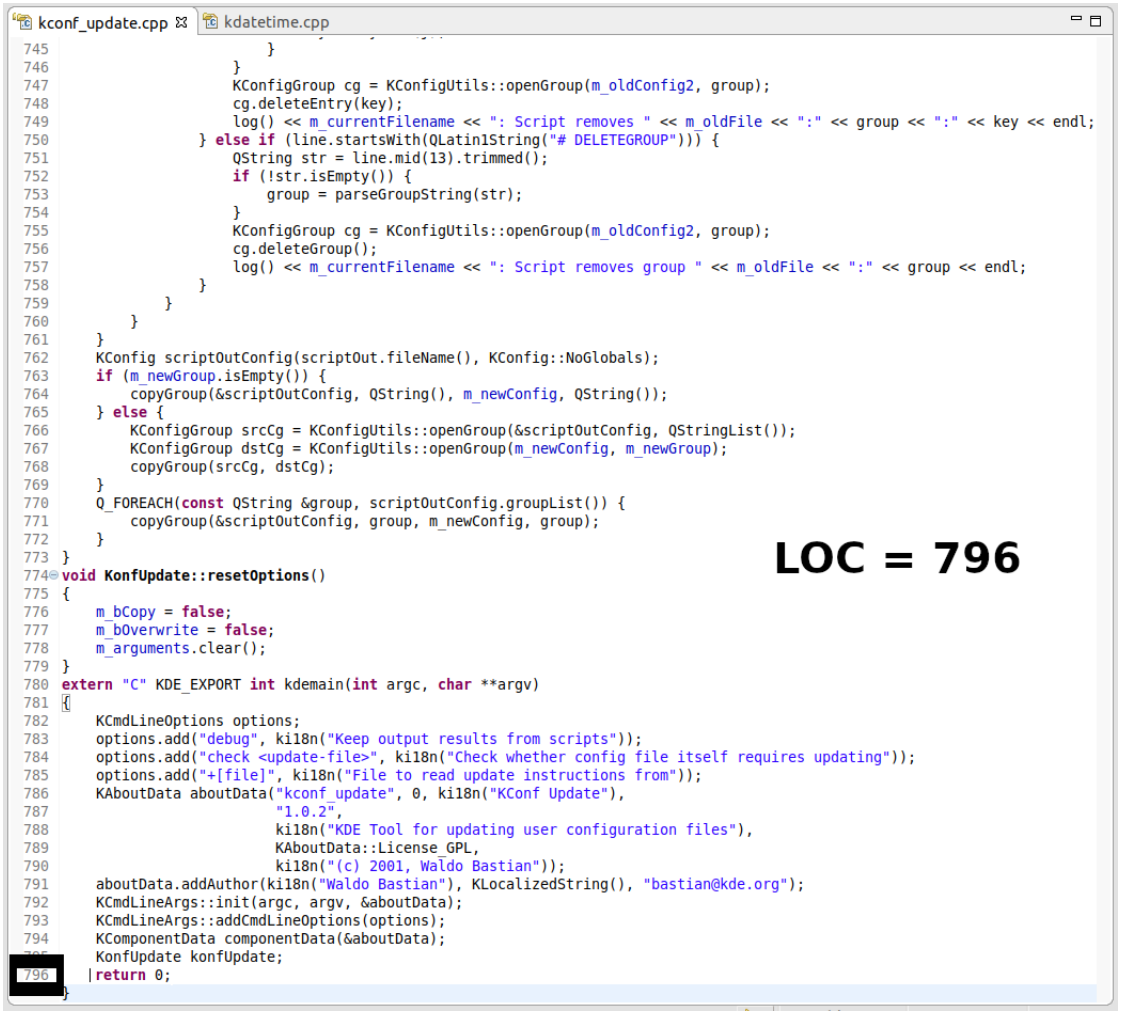


Figure 2.3: Exemplo de definição do valor da métrica LOC.

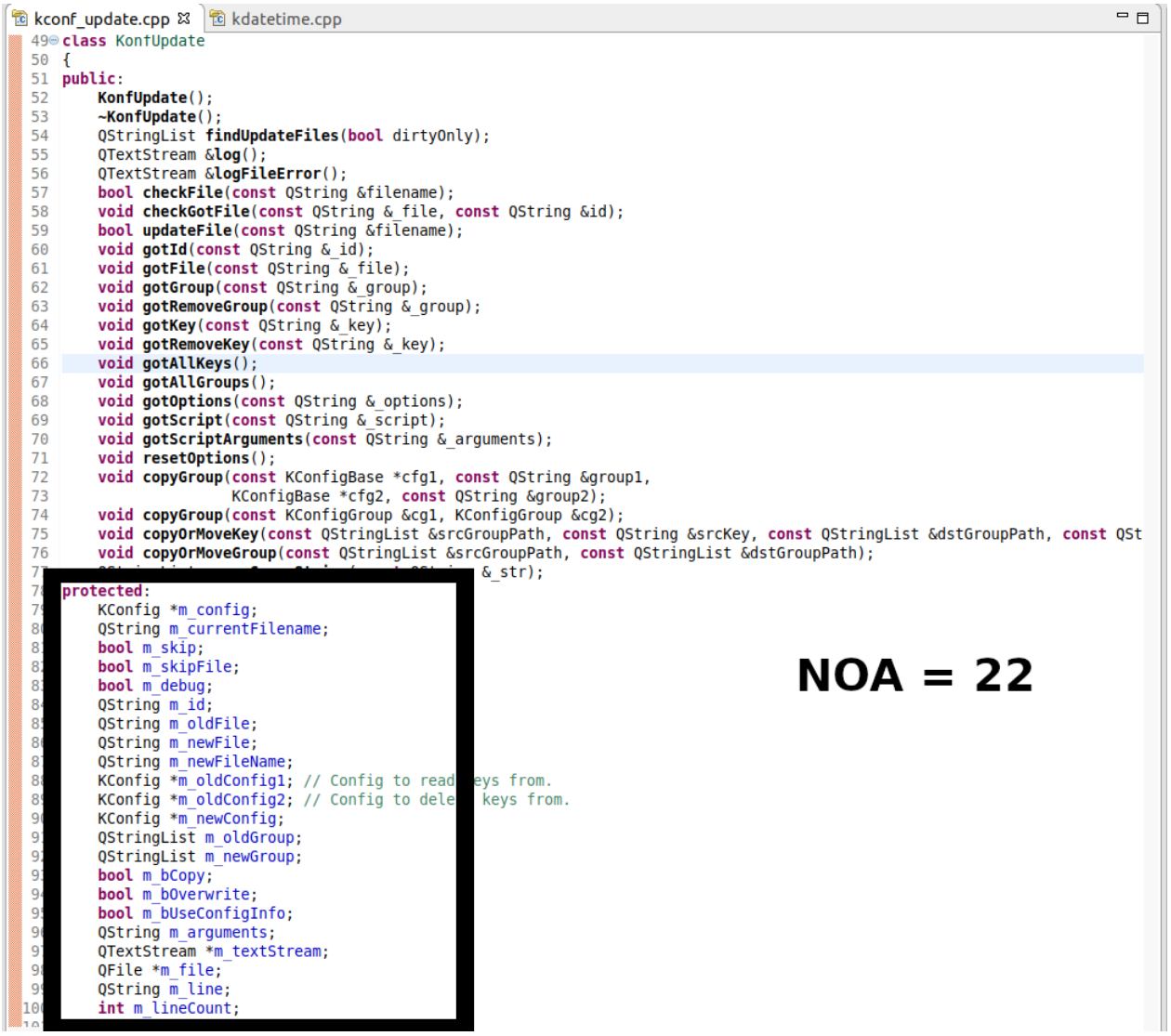


Figure 2.4: Exemplo de definição do valor da métrica NOA.

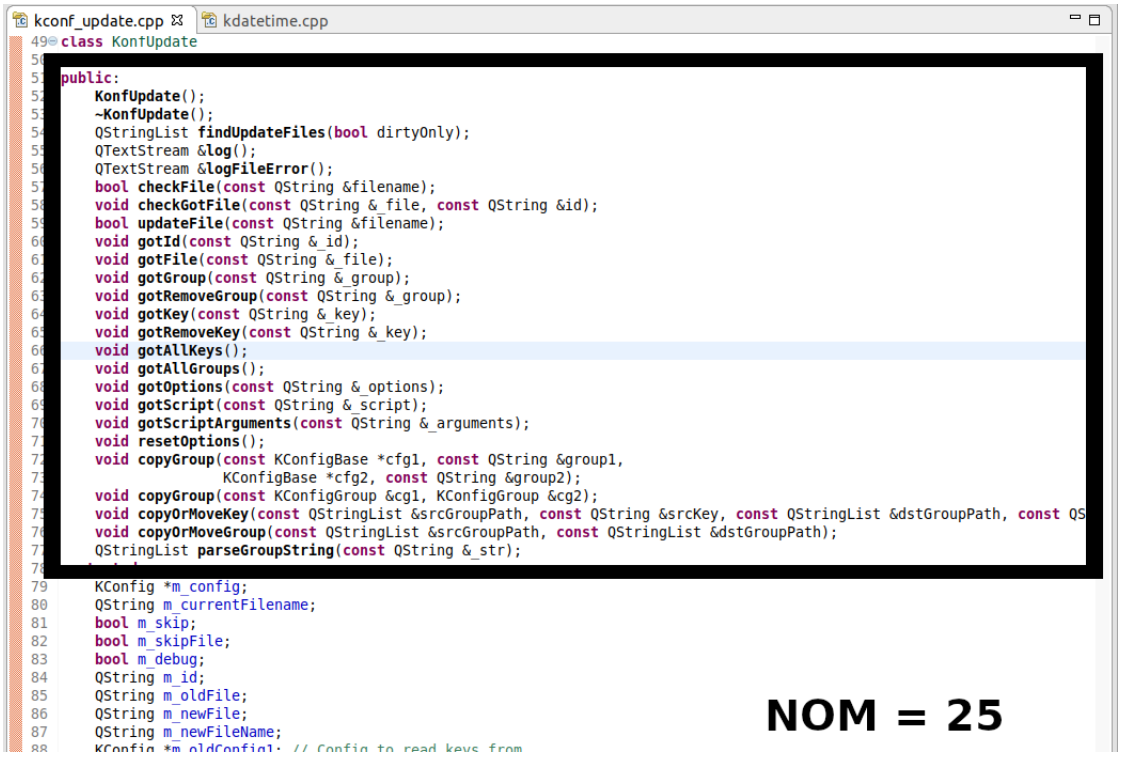


Figure 2.5: Exemplo de definição do valor da métrica NOM.

Tamanho do Maior Método (MMLOC)
A métrica Tamanho do Maior Método é uma variação da métrica Linhas de código. Ela representa o valor do maior método em uma classe/módulo (TERCEIRO et al., 2010). A Figura 2.6 apresenta um exemplo da métrica MMLOC, usando o módulo "`kconf_update`" do projeto KDELibs. Simbolicamente, pode-se definir o valor da métrica MMLOC é: Seja V o valor (quantidade de linhas de código) de um método/função M, presente na classe/módulo C, que contém n métodos/funções, tem-se: $\forall M \in C$, $MMLOC = MAX_{i=1}^{n} V(M_i)$

2.4.2 Métricas de Complexidade

Segundo Lehman et al. (1997), métricas de complexidade estão fortemente relacionadas à manutenibilidade, pois quanto mais interações, maior a chance de mudanças em uma parte do código afetar outras que estejam relacionadas. Nas subseções abaixo, são apresentadas as métricas de complexidade aplicadas nessa pesquisa.

Complexidade Ciclomática por Método (ACCM)
A Complexidade Ciclomática por Método (ACCM) é uma métrica que mede a complexidade de um método/função a partir da contagem do número de desvios de fluxos (caminhos independentes) que ele pode executar até o seu final (MCCABE, 1976). Para uma classe ou módulo, o valor dessa métrica é calculado considerando a média dos valores obtidos

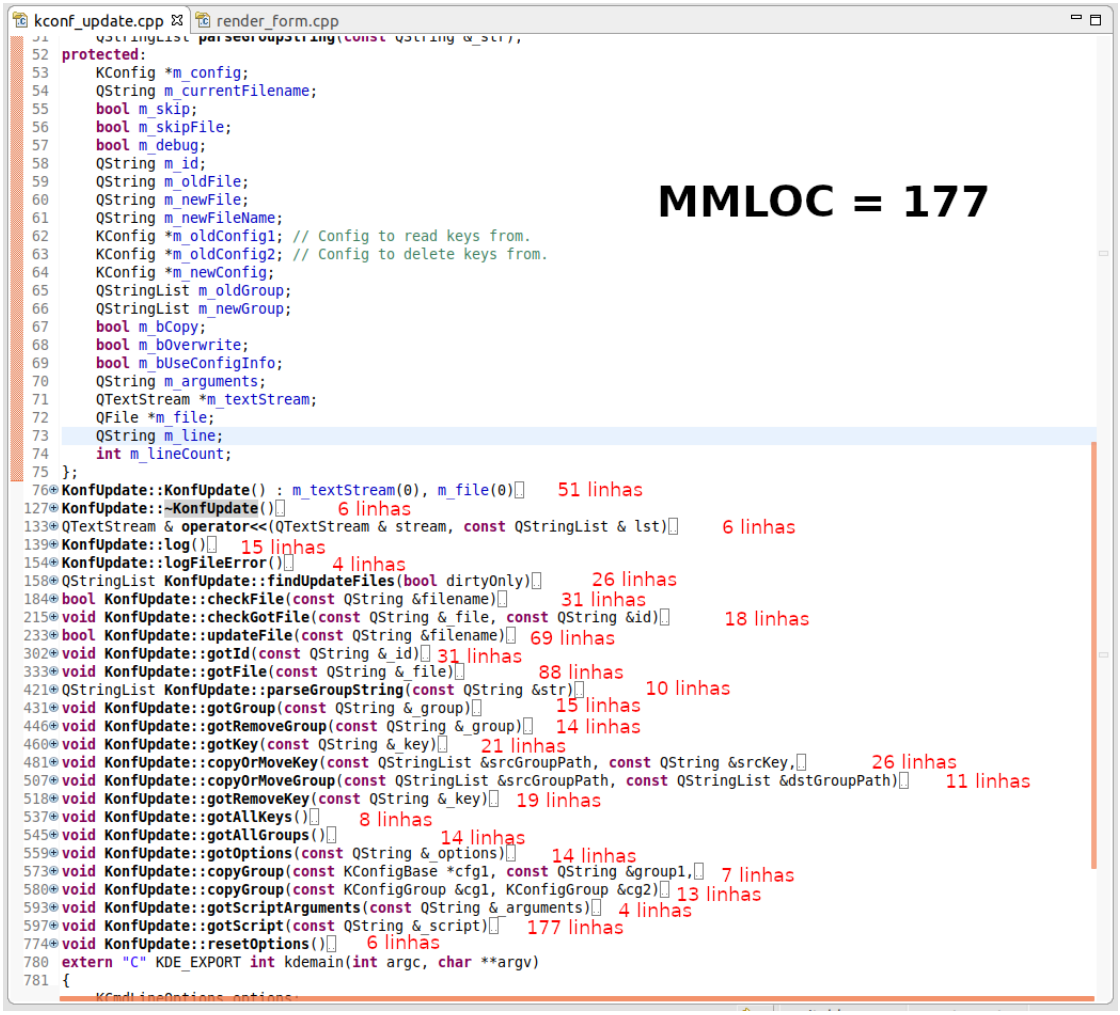
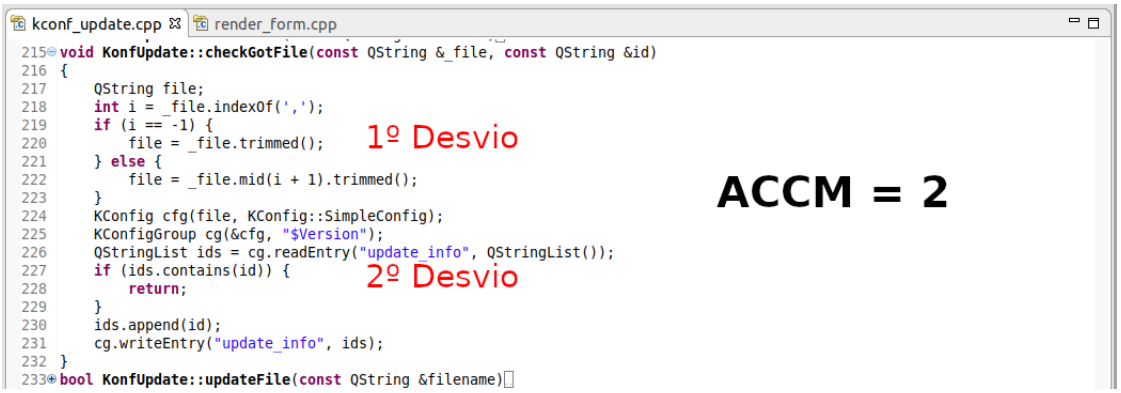


Figure 2.6: Exemplo de definição do valor da métrica MMLOC.

de ACCM para cada método/função. A Figura 2.7 apresenta um exemplo da métrica ACCM, usando a função "void KonfUpdate::checkGotFile(const QString, const QString)" do módulo "`kconf_update`" do projeto KDELibs.




```
215 void KonfUpdate::checkGotFile(const QString &_file, const QString &id)
216 {
217     QString file;
218     int i = _file.indexOf(',');
219     if (i == -1) {
220         file = _file.trimmed();
221     } else {
222         file = _file.mid(i + 1).trimmed();
223     }
224     KConfig cfg(file, KConfig::SimpleConfig);
225     KConfigGroup cg(&cfg, "$Version");
226     QStringList ids = cg.readEntry("update_info", QStringList());
227     if (ids.contains(id)) {
228         return;
229     }
230     ids.append(id);
231     cg.writeEntry("update_info", ids);
232 }
233 bool KonfUpdate::updateFile(const QString &filename)
```

Figure 2.7: Exemplo de definição do valor da métrica ACCM.

Número de Métodos Públicos (NPM)
A métrica de complexidade Número de Métodos Públicos é valorada a partir da contagem de métodos públicos que compõem uma classe/módulo (LORENZ; KIDD, 1994). A Figura 2.8 apresenta um exemplo de cálculo da métrica NPM, usando o módulo "`kconf_update`" do projeto KDELibs.

Número de Atributos Públicos (NPA)
A métrica Número de Atributos Públicos é valorada contando os atributos públicos que compõem uma classe/módulo (LORENZ; KIDD, 1994). A Figura 2.9 apresenta um exemplo de cálculo da métrica NPA, usando o módulo "`kconf_update`" do projeto KDELibs. Observa-se que o valor é zero, pois apesar de contar com 22 atributos, essa classe não atribui a nenhum deles o acesso público.

2.4.3 Métricas de Acoplamento

Métricas de acoplamento medem o grau de interdependência entre os módulos em uma estrutura de software (PRESSMAN; MAXIM, 2016). Quanto maior o valor de acoplamento para uma classe, maior dificuldade de se efetuar mudanças. Assim, o desejável é que haja conectividade simples entre os módulos, favorecendo a compreensão e diminuindo reações em cadeia em caso de bugs e modificações realizadas (STEVENS; MYERS; CONSTANTINE, 1979). Nas próximas subseções, são apresentadas as métricas de acoplamento consideradas nessa pesquisa.

Conexões Aferentes por Classe (ACC)
A métrica Conexões Aferentes por Classe mensura o grau de acoplamento do arquivo/módulo analisado através da contagem de arquivos/módulos que dependem do arquivo/módulo

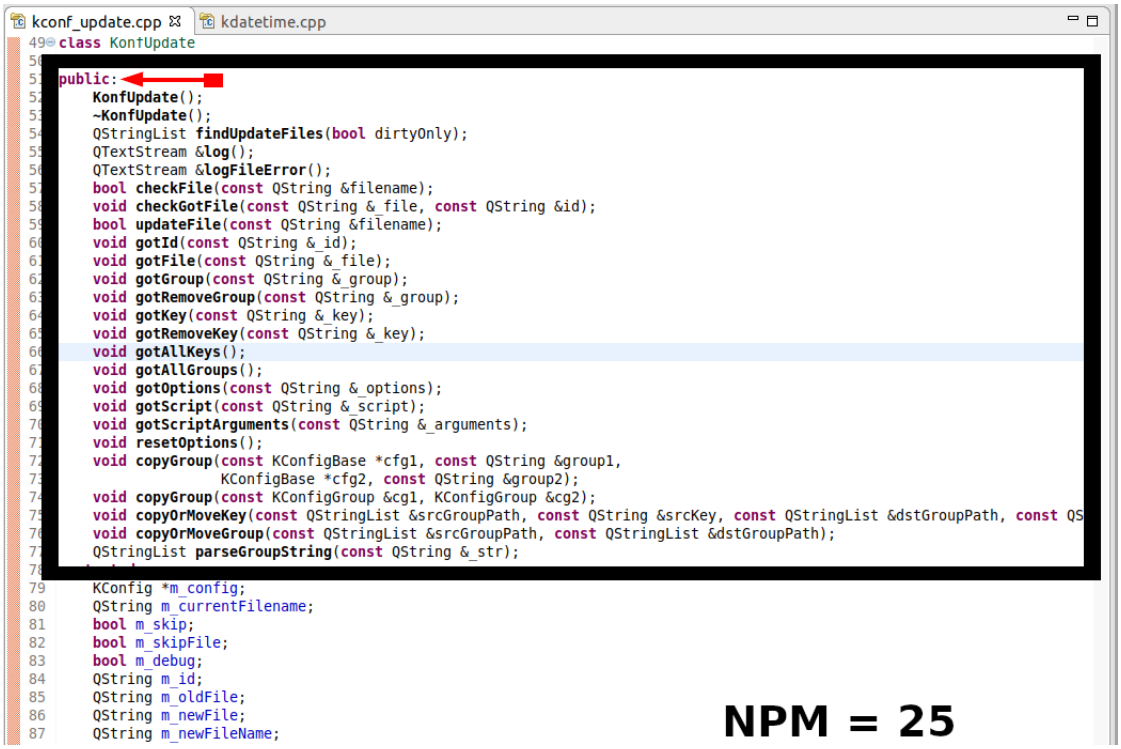


Figure 2.8: Exemplo de definição do valor da métrica NPM.

avaliado (MARTIN, 1994). A Figura 2.10 mostra um diagrama de classes desenvolvido em UML, simplificado, que representa algumas classes do projeto KDELibs. Ela apresenta o valor de ACC para a classe "`QString`" é 4, pois as classes "`KonfUpdate`", "`QTextStream`", "`QStringList`" e "`KConfig`" possuem relação de dependência estabelecidas com ela. Nesse mesmo exemplo, porém, observa-se que a classe "`KonfUpdate`" possui valor de ACC = 0, pois nenhuma das classes ali expressas possui relação de dependência estabelecida com a mesma.

Acoplamento entre Objetos (CBO)

A métrica Acoplamento entre Objetos mensura acoplamento do arquivo/módulo analisado através da contagem de arquivos/módulos que dependem do arquivo/módulo (conexão eferente) avaliado e dos arquivos/módulos dos quais ele depende (conexão aferente) (CHIDAMBER; KEMERER, 1994). A Figura 2.11 mostra um diagrama de classes desenvolvido em UML, simplificado, que representa algumas classes do projeto KDELibs. Ela mostra que, para as classes ali expressas, o valor da métrica CBO para a classe "`QString`" é 4, pois as classes "`KonfUpdate`", "`QTextStream`", "`QStringList`" e "`KConfig`" possuem relação de dependência estabelecidas com ela. Nesse mesmo exemplo, porém, observa-se que a classe "`KonfUpdate`" possui valor de CBO = 4, pois, apesar nenhuma das classes ali expressas possuir relação de dependência estabelecida com a mesma, ela depende de 4 outras classes.

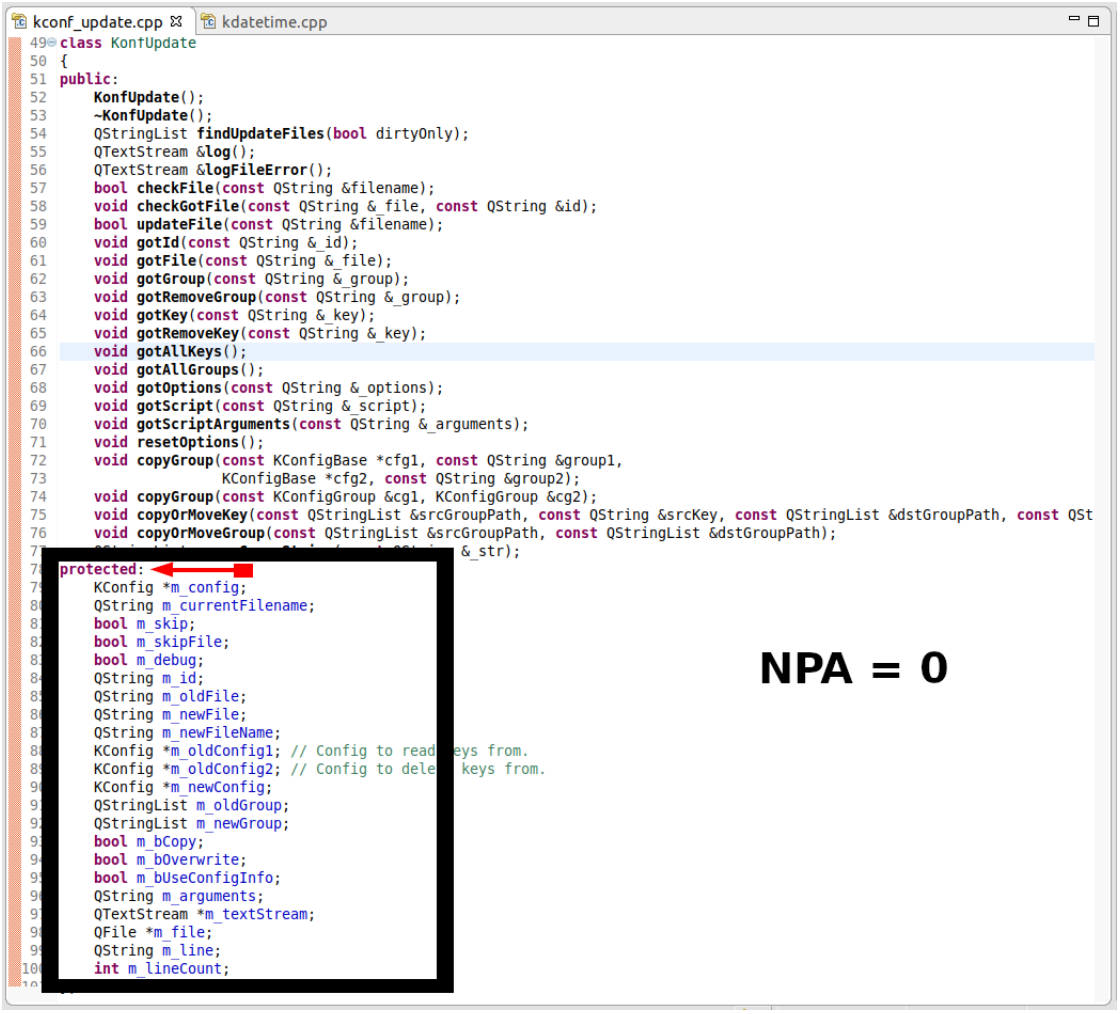


Figure 2.9: Exemplo de definição do valor da métrica NPA.

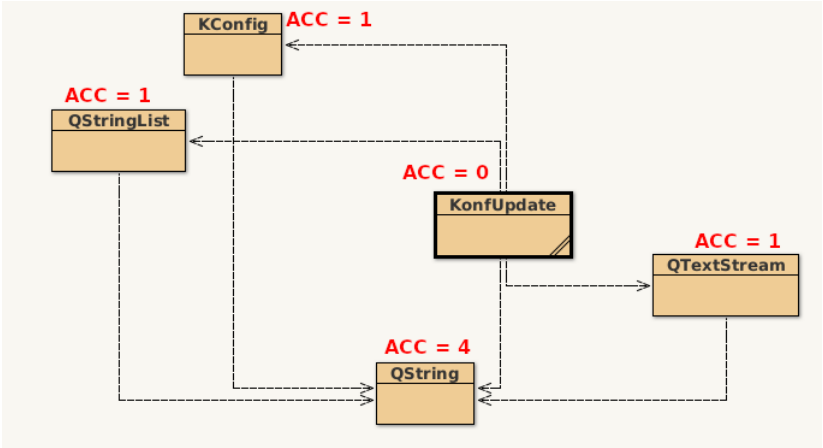


Figure 2.10: Exemplo de definição do valor da métrica ACC.

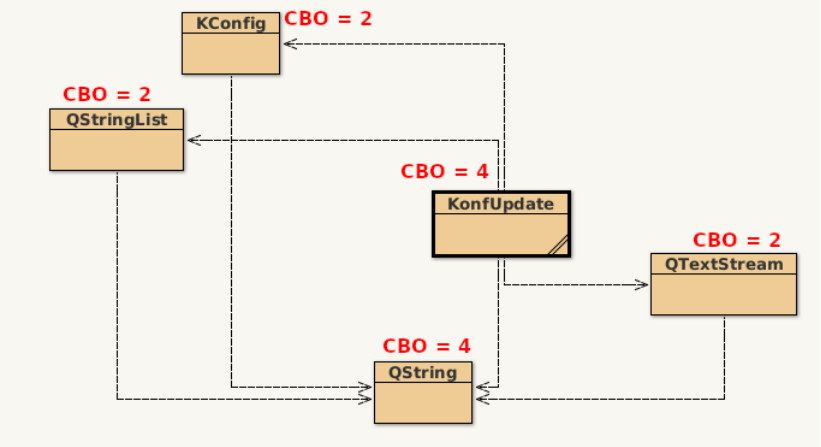


Figure 2.11: Exemplo de definição do valor da métrica CBO.

Profundidade da Árvore de Herança
A métrica Profundidade da Árvore de Herança (DiT) calcula o acoplamento de uma classe/módulo através da maior distância da classe até a raiz na árvore de herança. Esse conceito considera a possibilidade dessa classe/módulo implementar o conceito de herança múltipla (CHIDAMBER; KEMERER, 1994). A Figura 2.12 apresenta um diagrama de classes desenvolvido em UML, simplificado, que representa algumas classes do projeto KDELibs. Ela exemplifica, para as classes ali expressas, o valor da métrica DiT para as classes "`QCommandLineButton`" e "`PushButtonWidget`" é 4, pois elas são filhas da classe "`QPushButton`" que, por sua vez, é filha da classe "`QAbstractButton`" que é filha da classe "`QWidget`" que é filha da classe "`QObject`", percorrendo, portanto, 4 níveis de herança até a classe "`QObject`" que, no contexto desse diagrama, é a raiz da árvore de herança.

Número de Classes Filhas (NOC)
A métrica Número de Classes Filhas (NOC), por sua vez, calcula o acoplamento de uma classe/módulo através da quantidade de filhos (subclasses, por exemplo) desse módulo/classe (CHIDAMBER; KEMERER, 1994). A Figura 2.13 apresenta um diagrama de classes desenvolvido em UML, simplificado, que representa algumas classes do projeto KDELibs. Ela mostra que, para as classes ali expressas, o valor da métrica NOC para a classe "`QCommandAbstractButton`" é 4, pois ela é mãe das classes "`QPushButton`", "`QRadioButton`", "`QToolButton`" e "`QCheckBox`". Por outro lado, as classes "`QToolButton`" e "`QCheckBox`" possuem o valor de NOC igual a zero, pois, nesse contexto, não tem classes filhas.

Resposta por Classe (RFC)
A métrica Resposta por Classe (RFC) mede o acoplamento de uma classe/módulo através da contabilidade de métodos/funções dessa classe/módulo acrescido do número de métodos/funções que ela invoca (de outras classes ou módulos). Em resumo, a métrica RFC reflete o número de métodos/funções que podem ser executados como resultado do envio de

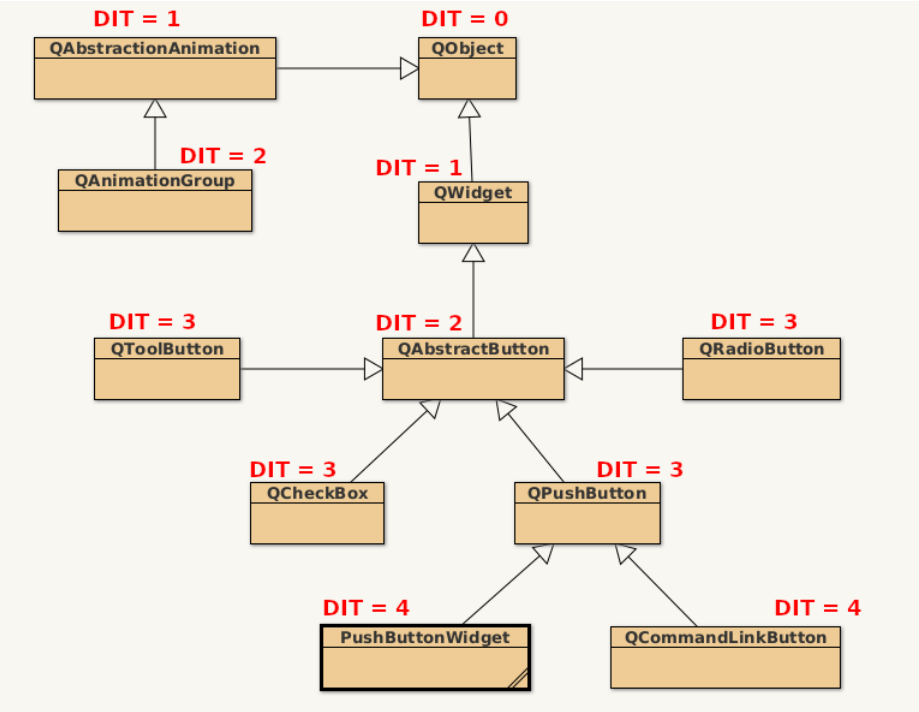


Figure 2.12: Exemplo de definição do valor da métrica DiT.

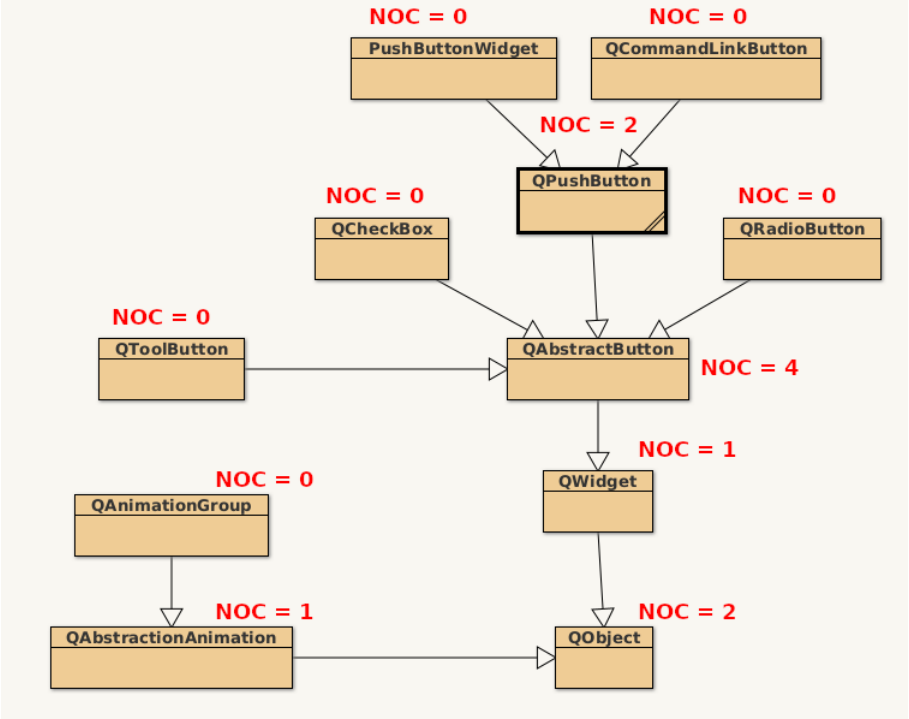


Figure 2.13: Exemplo de definição do valor da métrica NOC.

uma mensagem ao módulo/classe (CHIDAMBER; KEMERER, 1994). A Figura 2.14 apresenta os cálculos realizados a partir do código da classe "`kconf_update`" para a definição do valor da métrica RFC para a mesma. Inicialmente, foi contabilizada a quantidade de métodos, que foram somados à quantidade de métodos externos invocados por cada método dessa classe e, por fim, subtraída a quantidade de métodos externos invocados em mais de método dessa classe.

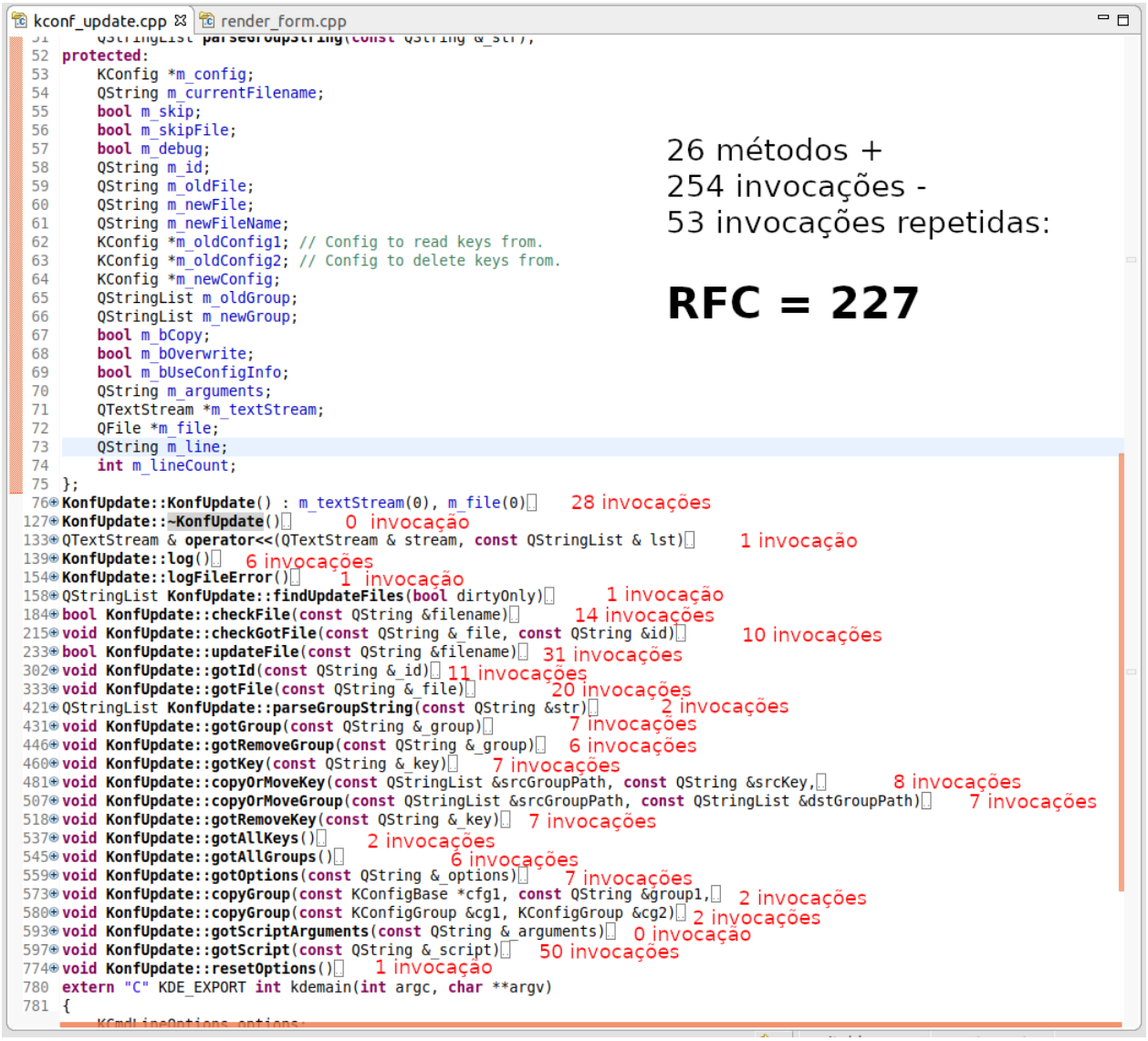


Figure 2.14: Exemplo de definição do valor da métrica RFC.

2.4.4 Métricas de Coesão

Métricas de coesão mensuram o grau de responsabilidade que uma unidade de software está relacionada (FENTON; BIEMAN, 2014). Uma unidade de software que participa de tarefas não-relacionadas ou executa muitas tarefas tem como consequência baixo índice

de coesão (ALSHAYEB, 2009). Segundo Larman (2012), uma unidade de software coesa é de fácil compreensão, reutilização e manutenção, portanto, desejável. A seguir, são apresentadas as métricas de coesão utilizadas nessa pesquisa.

Falta de Coesão dos Métodos (LCOM4)
A métrica Falta de Coesão dos Métodos (LCOM4) mede a coesão de uma classe/módulo a partir do grau de compartilhamento de atributos/variáveis entre os métodos/funções dessa classe/módulo (CHIDAMBER; KEMERER, 1994). A Figura 2.15 exemplifica o cálculo dessa métrica. Seja x, y, w e z, atributos da classe ExemploLCOM4 e "`public int getX()`" (aqui chamado de A), "`public int getY()`" (chamado de B), "`public int getW()`" (chamado de C), "`public int getZ()`" (chamado de D), "`public int somaWZ()`"
(chamado de E) e "`public int somaXY()`" (chamado de F) os métodos dessa classe. Deve-se produzir um conjunto P, formado por subconjuntos de pares de métodos onde a intersecção de atributos utilizados por esse método é vazia. Deve-se também produzir um conjunto Q formado por subconjuntos de pares de métodos, onde a intersecção de atributos utilizados por esse método não é vazia.

O conjunto P será formado por:

$$A \cap B, A \cap C, A \cap D, A \cap E, B \cap C, B \cap D, B \cap E, C \cap D, C \cap F, D \cap F,$$

enquanto o conjunto Q será formado por: $A \cap F, B \cap F, C \cap E, D \cap E$.

Dessa forma o valor de LCOM4 será definido pela subtração entre a quantidade de subconjuntos do conjunto P e a quantidade de subconjuntos do conjunto Q. Dessa forma, $LCOM4_{ExemploLCOM4} = 10-4$. Assim o valor de **LCOM4** para a classe ExemploLCOM4 é 6.

Complexidade Estrutural (SC)
A métrica Complexidade Estrutural (SC) mede a coesão de uma classe/módulo a partir do produto entre o acoplamento dessa classe/módulo medida pela métrica CBO e a coesão medida pela métrica LCOM4 (DARCY et al., 2005). Esses autores mostraram que essa métrica é positivamente correlacionada com o esforço dispendido em operações de manutenção e evolução de produtos de software, isto é, quanto maior a complexidade estrutural, maior o esforço dispendido entre essas tarefas.

A subseção a seguir apresenta as considerações finais sobre os fundamentos teóricos apresentados neste capítulo.

2.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os fundamentos teóricos que dão sustentação à este trabalho. Foram apresentados conceitos relacionados à Arquitetura de Software, sua documentação e os trabalhos a ela relacionados. Também foram apresentados conceitos relacionados à repositórios de Projetos de Software Livre, onde além dos conceitos relacionados, foi apresentado o projeto KDELibs, principal fonte de dados deste estudo de doutorado. Em

```
class ExemploLCOM4
{
    private int x;
    private int y;
    private int w;
    private int z;
    public ExemploLCOM4()
    {
        x = 0;
        y = 0;
        w = 0;
        z = 0;
    }
    public int getX(){
        return x;
    }
    public int getY(){
        return y;
    }
    public int getW(){
        return w;
    }
    public int getZ(){
        return z;
    }
    public int somaWZ(){
        return w+z;
    }
    public int somaXY(){
        return x+y;
    }
}
```

Figure 2.15: Exemplo de definição do valor da métrica LCOM4.

seguida, foram apresentados conceitos e técnicas de mineração de repositórios, além de métodos para buscar informações em fontes de dados não-estruturadas.

Por fim, as seções seguintes se encarregaram de apresentar as métricas utilizadas para caracterizar as mudanças arquiteturais no tocante ao impacto causado no código-fonte. O próximo capítulo apresenta o primeiro estudo desta pesquisa de doutorado. Nele, o repositório do KDELibs foi explorado, a partir de mensagens de *commit*, com auxílio de palavras-chaves relacionadas à arquitetura de software, em busca de informações que identificassem mudanças no projeto que ocasionaram alterações na arquitetura do projeto.

Chapter 3

CARACTERIZANDO INFORMAÇÕES ARQUITETURAIS EM MENSAGENS DE COMMIT

3.1 INTRODUÇÃO

Os desenvolvedores tomam várias decisões durante o desenvolvimento de software, sejam eles relacionados ao design (e, especificamente, à arquitetura), implementação, requisitos seja por outras questões. Essas decisões podem mudar durante a evolução do projeto. O conhecimento dessas mudanças é de grande importância durante a manutenção e evolução do software, pois fazer alterações, sem o devido cuidado, pode resultar em resultados indesejáveis (GURP; BOSCH, 2002; FARID; AZAM; IQBAL, 2011).

Os sistemas de controle de versão (VCS) acompanham as mudanças em um projeto. Em um VCS, os desenvolvedores enviam alterações ao código-fonte de um projeto e podem registrar a motivação por trás dessas alterações. À medida que o software evolui, além de ter a documentação inicial, acompanhar a evolução do projeto também é desejável para desenvolvedores e pesquisadores da área (OREIZY; MEDVIDOVIC; TAYLOR, 1998).

Capturar informações arquiteturais é desafiador, no contexto de Projetos de Software Livre, pois os colaboradores, geralmente, estão distribuídos geograficamente e nenhuma documentação clara do projeto pode ser detectada (BRUNET et al., 2014). Os projetos de Software Livre dependem quase, exclusivamente, de comunicação por escrito, como mensagens de *commit*, comentários de código, canais IRC (*Internet Relay Chat*), listas de discussão e wikis, para colaboração e coordenação de atividades de desenvolvimento. A comunicação que ocorre por esses meios contém muitas informações sobre o sistema de software, a exemplo do histórico de desenvolvimento, razões para mudanças (STOREY et al., 2014; ALKADHI et al., 2017) e decisões de design (KO; DELINE; VENOLIA, 2007; HASSAN, 2008).

Em um trabalho anterior, Brunet et al. (2014) analisaram 77 projetos populares de Software Livre disponíveis no GitHub para entender se o design de software é discutido nos comentários de *commits*, nas solicitações de mudanças ou em mensagens de *issue trackers*. Eles consideraram como uma discussão sobre design um conjunto de comentários contendo ao menos um comentário referente às características estruturais do design de

software, a exemplo da ocorrência de dependência entre classes. Como resultado, eles descobriram que, em média, 25% das discussões em um projeto mencionam algum aspecto do design. No entanto, eles não diferenciaram discussões sobre detalhes de design ou design arquitetural. Dada à evidência de que as mensagens de *commit* podem conter informações de design, é importante investigar em que medida se pode obter informações arquiteturais nessa fonte de informação.

Para isso, foi realizado um estudo com o objetivo de investigar se, e como, as alterações arquiteturais são relatadas nas mensagens de *commit*. O objetivo é fornecer aos pesquisadores e profissionais uma primeira visão sobre em que grau eles são capazes de explorar mensagens de *commit* como fonte de informações arquiteturais no futuro. Em particular, foi investigado se as informações arquiteturais estão disponíveis nas mensagens de *commit* do projeto KDELibs[1], o qual contém, em seu VCS, *commits* dos últimos 20 anos de seu desenvolvimento.

Esse trabalho foi guiado pela questão de pesquisa **RQ1** que questiona a disponibilidade de informações sobre mudanças na arquitetura em mensagens de *commit* disponíveis em sistemas de controle de versão. Ela foi desmembrada em sete subquestões de pesquisa que além de identificar tais informações, as caracterizam a partir de mensagens de *commit* do projeto KDELibs. A seguir, as subquestões relacionadas a este trabalho são explicitadas.

RQ1.1: Os colaboradores do projeto informam mudanças arquiteturais nas mensagens de *commit*?
Nesta subquestão, analisou-se a presença de *commits* com mensagens que mencionam algum aspecto arquitetural. Para isso, foi construído um conjunto de palavras-chave relacionadas à arquitetura, criado a partir de um *survey* realizado junto à especialistas. Tendo encontrado um número razoável de *commits* com informações arquiteturais, pode-se prosseguir com as próximas subquestões de pesquisa, que objetivam caracterizar essas mensagens de *commit* com informações sobre mudanças arquiteturais sobre vários aspectos. Foi considerada como mensagem arquitetural uma mensagem de *commit* que possui ao menos uma palavra-chave relacionada à arquitetura de software, detectada por uma abordagem automatizada e validada pelos pesquisadores participantes deste estudo.

RQ1.2: Há períodos que concentram *commits* com alterações na arquitetura?
Nesta subquestão, analisou-se a distribuição de *commits* arquiteturais ao longo do tempo em que o projeto evoluiu. O objetivo foi entender se há pontos ao longo da evolução do software que concentram mudanças na arquitetura.

RQ1.3: Quais são os tópicos arquiteturais mais citados nas mensagens de *commit* sobre mudanças na arquitetura?
Ao definir a arquitetura do software, a literatura menciona diferentes aspectos do software que podem ser considerados como arquiteturais. Esses aspectos foram listados como tópicos (Seção 3.3.4.3) e utilizados, nesta subquestão de pesquisa, para caracterizar os aspectos de software envolvidos nas mudanças arquiteturais. Investigou-se quais tópicos ocorrem com maior frequência e quando ocorrem ao longo da evolução do projeto. Além disso, registrou-se quais palavras-chave foram úteis para encontrar cada tópico.

RQ1.4: Qual é o perfil dos colaboradores que realizam alterações na arquitetura?
Nesta subquestão, caracterizou-se o perfil dos colaboradores que realizam mudanças

[1]https://github.com/KDE/kdelibs

arquiteturais, com o objetivo de compreender se os colaboradores arquiteturais também são os mais experientes em termos de número de mudanças que fazem ao longo da evolução do projeto.

RQ1.5: As mensagens de *commit* arquiteturais são mais longas do que outros tipos de mensagens de *commit*?
Nesta subquestão de pesquisa, analisou-se o tamanho do texto das mensagens de *commit* arquiteturais em comparação com mensagens sem informações arquiteturais (não-arquiteturais). Espera-se que as mensagens de *commit* arquiteturais sejam mais longas, pois as mudanças na arquitetura, provavelmente, terão um impacto crítico e, assim, exigirão mais explicações. Caso essa expectativa se confirme, os engenheiros de software, que procuram informações arquiteturais, podem restringir a sua pesquisa a *commits* com mensagens maiores.

RQ1.6: *Commits* arquiteturais alteram mais arquivos do que *commits* não-arquiteturais?
Esta subquestão de pesquisa visa caracterizar o impacto dos *commits* arquiteturais. Espera-se que as mudanças na arquitetura sejam granulares e tenham um impacto mais amplo, afetando diferentes partes do software. Nesse caso, os engenheiros de software, que procuram informações arquiteturais, podem restringir sua pesquisa a *commits* que alteram um grande número de arquivos/módulos.

RQ1.7: *Commits* arquiteturais alteram o mesmo conjunto de arquivos?
A hipótese por trás desta pergunta é que podem haver elementos de um sistema que são essenciais para sua arquitetura, de modo que as alterações arquiteturais, frequentemente, os afetam. Portanto, essa subquestão de pesquisa tem como objetivo analisar se diferentes *commits* arquiteturais tendem a afetar conjuntos semelhantes de arquivos de código-fonte com frequência.

Ao responder a essas perguntas, apresentar-se-ão as contribuições deste trabalho com base no projeto estudado: Indicam-se quais são as características das discussões sobre a arquitetura de um projeto durante sua evolução. Será revelado que existem colaboradores que focam as discussões arquiteturais do software – 10 colaboradores registram 54% das mensagens com informações arquiteturais – e que essas discussões também se concentram nos períodos de lançamento de novas *releases*. Há seis vezes mais mensagens com discussão arquitetural nesse período do que em meses fora dele. Criou-se uma taxonomia para informações arquiteturais. No projeto estudado, foram encontrados todos os tópicos arquiteturais definidos nessa taxonomia. Também se descobriu que *commits* com mensagens contendo informações arquiteturais têm, em média, três vezes mais caracteres do que em *commits* com outros tipos de alterações. Finalmente, mostrou-se que *commits* com mensagens com informações arquiteturais modificam mais arquivos do que *commits* com outros tipos de alterações (24% a mais) e que eles modificam os mesmos arquivos com mais frequência do que outros tipos de alterações (12% a mais). Essas contribuições podem ser úteis para vários propósitos. Tome-se como exemplo que, essas informações podem ajudar a orientar novos colaboradores do projeto sobre a quem consultar se tiverem dúvidas sobre a arquitetura do projeto ou em que períodos do projeto deve-se procurar essas informações ao longo de sua evolução.

O restante deste capítulo está organizado da seguinte forma. A Seção 3.2 discute trabalhos relacionados. A Seção 3.3 descreve o design do estudo. A Seção 3.4 discute

os resultados do estudo. A Seção 3.5 apresenta implicações dos resultados obtidos para pesquisa, prática e educação. A Seção 3.6 discute as ameaças à validade deste estudo. Por fim, a Seção 3.7 apresenta as considerações finais sobre esse estudo.

3.2 TRABALHOS RELACIONADOS

Nessa seção, discutem-se, brevemente, os trabalhos relacionados a este trabalho. Eles envolvem (i) mineração de informações de design de fontes não-estruturadas e (ii) recuperação de informações de fontes de dados não-estruturados. Realizou-se uma revisão não sistemática da literatura, pesquisando artigos relacionados a esses tópicos no Google Scholar.

3.2.1 Mineração de informações de design em fontes não-estruturadas

Como já foi afirmado na Seção 3.1, além de estar relacionado a esse estudo, o trabalho desenvolvido por Brunet et al. (2014) também serviu de motivação, pois eles encontraram discussões sobre design em mensagens de *commit* em projetos de Software Livre. De fato, eles descobriram que uma quantidade razoável de mensagens de *commit*, solicitações mudanças e problemas reportados mencionam aspectos de design. No entanto, eles não diferenciam informações arquiteturais e informações de detalhes de design. Além disso, ao contrário desse trabalho, eles não usaram palavras-chave para pesquisar mensagens relacionadas ao design. Em vez disso, eles marcaram, manualmente, 1.000 discussões (conjunto de comentários discutindo uma *commit*, solicitação de mudança ou problema reportado) como relacionadas ao design ou não. Em seguida, eles usaram esse conjunto de discussões para treinar um classificador de aprendizado de máquina. Por fim, eles usaram o classificador para extrair discussões sobre 77 projetos de Software Livre.

Hesse et al. (2016) investigaram relatórios de problemas obtidos em *issue trackers* dos projetos Firefox e Linux Kernel. Eles procuraram a existência de estratégias dominantes para documentar e discutir questões e estudaram se havia uma relação entre a estratégia de documentação e as decisões tomadas para abordar o problema, incluindo decisões de design e arquitetura. Diferente deste estudo, eles consideravam qualquer tipo de decisão, não apenas decisões de design e arquitetura. Além disso, eles pesquisaram sobre essas informações em relatórios de erros ao invés de mensagens de *commit*.

Bhat et al. (2017) apresentaram uma abordagem para extrair decisões de design registradas em *issue trackers*, usando o aprendizado de máquina. Eles produziram um modelo para classificar as decisões de design de acordo com um conjunto de categorias predefinidas. No entanto, os autores analisaram apenas dois projetos, dificultando o uso dos dados com outros projetos. Mais uma vez, o foco foi analisar *issue trackers* ao invés de mensagens de *commit*.

Recentemente, Alkadhi et al. (2018) trouxeram um estudo que apresentou a possibilidade de recuperação da lógica por trás das decisões durante o desenvolvimento de software, incluindo decisões de design. Eles descobriram que um quarto de todas as mensagens nos canais de IRC de projetos de Software Livre contém a lógica para as mudanças e que os *commiters* de código contribuíram com a maior parte dessas explicações. Os autores, no entanto, não forneceram nenhuma caracterização adicional dessas justificativas e focaram

nas mensagens do IRC.

3.2.2 Recuperação de informações de fontes não-estruturadas

Estudos anteriores também consideraram a recuperação de informações de software (além de informações de design ou arquitetura) através de artefatos não-estruturados, porém buscando outros tipos de informações. Ying, Wright e Abrams (2005), por exemplo, analisaram comentários no código-fonte do projeto Eclipse. Por meio da verificação manual, eles identificaram que os comentários não se limitavam a apoiar o entendimento do programa, mas também a relatar tarefas futuras, partes de tarefas inacabadas, partes do código-fonte que requerem atenção, discussão de soluções a serem implementadas e objetivos a serem alcançados através de alterações de pedidos ou indicação de preocupações.

Hindle, German e Holt (2008) compararam técnicas para recuperar informações de software a partir de artefatos não-estruturados. Eles verificaram que essas técnicas não eram eficazes porque não tinham sensibilidade ao contexto. Em seguida, propuseram uma ferramenta para explorar os logs de *commits* do VCS para recuperar informações de manutenção de software.

Outro exemplo de trabalho usando mineração sobre fontes de dados não-estruturados é o estudo realizado por Hirata e Mizuno (2011). Eles exploraram os comentários disponíveis no código-fonte. Foi utilizada uma abordagem baseada em filtragem de texto para previsão de módulos propensos às falhas e uma abordagem para medir a distância entre código e comentários. A partir dessa métrica, eles calcularam as probabilidades de falha para cada módulo.

Shihab, Zhen e Hassan (2009) examinaram os logs de reuniões realizadas via IRC usados pelos principais colaboradores e mantenedores do GNOME GTK+. Para isso, foram utilizadas três métricas diferentes: o volume de discussão da reunião, os níveis de participação na reunião e o nível de contribuição dos participantes da reunião. Foi realizada uma inspeção nos logs da reunião e identificados cinco tipos diferentes de linhas de mensagem do IRC. Por fim, desenvolveram uma estrutura que usa expressões regulares para lidar com os diferentes tipos de mensagens e classificá-los.

Como se percebe, a busca por informações não-documentadas em fontes de dados não-estruturadas não é uma novidade em si, mas até aqui, elas não haviam sido utilizadas na busca por informações sobre mudanças arquiteturais.

3.3 DESIGN DO ESTUDO

Nesta seção, encontra-se descrito o design deste estudo. Inicialmente, é apresentada uma visão geral dele. Em seguida, descreve-se como foi construído o conjunto de palavras-chave usado para pesquisar os *commits* arquiteturais. Por fim, são descritos os procedimentos executados para extrair e caracterizar informações arquiteturais das mensagens dos *commits*.

3.3.1 Visão geral

Para extrair informações arquiteturais das mensagens de *commit*, seguiu-se um processo de mineração para buscar **Traços Arquiteturais**. Foram definidas como **Traço Arquitetural**

as *informações sobre a arquitetura do software que são descritas de forma incompleta e que não são comparáveis às instruções escritas em uma linguagem de descrição arquitetural e não estejam disponíveis em uma documentação da arquitetura do software, mas em meio a outros artefatos de documentação*, tais como mensagens de commit ou comentários de código. Não era esperado a identificação de descrições explícitas e completas de restrições ou conexões arquiteturais, mas apenas traços (vestígios) desse tipo de informação. Um exemplo de mensagem de *commit* com traço arquitetural pode ser encontrado na Figura 3.1. Essa mensagem foi considerada como sendo arquitetural, porque trata do fato do relacionamento entre componentes de software que deverá ocorrer por meio de interfaces públicas e não de elementos internos.

```
only make the interfaces public, or we will use the internal stuff again everywhere...
svn path=/trunk/kde/kdelibs/; revision=661367
```

Figure 3.1: Exemplo de mensagem de *commit* com conteúdo arquitetural.

O processo de mineração de informações arquiteturais adotado consistiu nas seguintes etapas: (i) construção um conjunto de palavras-chave relacionadas à arquitetura de software, (ii) filtragem automática de mensagens de *commit* do projeto KDELibs, usando as palavras-chave definidas na etapa anterior, (iii) inspeção manual de cada mensagem de *commit* para a confirmação do conteúdo e (iv) classificação das informações arquiteturais de cada mensagem de acordo com os tópicos arquiteturais definidos na taxonomia seguida. Na Seção 3.3.4.4, foram fornecidos mais detalhes sobre os procedimentos adotados para analisar os dados e, assim, responder cada subquestão de pesquisa proposta.

Para construir um conjunto de palavras-chave relacionadas à arquitetura de software (etapa i), foi realizado um *survey* com cinco pesquisadores especialistas em arquitetura de software. Depois, o nível de concordância entre os conjuntos de palavras-chave produzidos pelos especialistas foi avaliado. Para isso, usou-se o índice *kappa* (COHEN, 1968). Para selecionar o grupo de especialistas, foi adotada a técnica de amostragem por conveniência, procurando especialistas em arquitetura de software nos grupos de pesquisa com os quais se teve contato.

Para filtrar, automaticamente, as mensagens de *commit* (etapa ii), foi extraído o log de mensagens do VCS do projeto KDELibs e o mesmo inserido como entrada para uma ferramenta desenvolvida para esse fim, chamada **Architecture Traces Mining** (ATM). A ferramenta extrai o radical das palavras-chave, a fim de identificar todas as variações de cada uma das palavras.

Tendo o conjunto de mensagens de *commit* candidatas a conter informações arquiteturais, a próxima etapa foi filtrar, manualmente, as mensagens de *commit* não relacionadas à arquitetura (etapa iii). Para isso, todas as mensagens de *commit* selecionadas pela ferramenta ATM fora lidas. Para aceitar uma mensagem de *commit* como arquitetural, considerou-se: (i) o conhecimento pessoal sobre arquitetura de software, (ii) a experiência com desenvolvimento de software e (iii) diferentes definições da arquitetura de software encontradas na literatura.

Finalmente, na etapa iv, classificou-se cada mensagem de *commit* considerada como arquitetural na etapa iii, de acordo com uma lista de tópicos arquiteturais que foram

derivados de diferentes definições de arquitetura de software encontradas na literatura (PERRY; WOLF, 1992; GARLAN; SHAW, 1993; BASS; CLEMENTS; KAZMAN, 1998) (e apresentada na Seção 3.3.4.3). Nesse processo de classificação, cada mensagem de *commit* foi lida novamente, buscando identificar traços arquiteturais relacionados aos tópicos arquiteturais considerados. Classificou-se o *commit* mostrado na Figura 3.1, por exemplo, como "Relacionamento entre Elementos". Para mais detalhes sobre a lista de tópicos arquiteturais, consulte a Seção 3.3.4.3. Nas próximas seções, mais detalhes de cada etapa deste estudo serão apresentados.

3.3.2 Construindo um conjunto de palavras-chave arquiteturais

Conforme afirmado anteriormente, foi utilizado um conjunto de palavras-chave relacionadas à arquitetura para executar uma primeira seleção automática de mensagens de *commit* que descrevem mudanças na arquitetura. Para construir esse conjunto de palavras-chave, realizou-se um *survey* com pesquisadores especialistas em arquitetura de software.

Decidiu-se usar palavras-chave como um dos pilares do processo de mineração aqui descrito, porque elas estão, frequentemente, em informações relevantes (GUAN; WONG, 1999). Algumas pesquisas usam o aprendizado de máquina para definir palavras-chave, principalmente usando, LDA (MASKERI; SARKAR; HEAFIELD, 2008), LSA (KUHN; DUCASSE; GÍRBA, 2007) e LSI (MARCUS; MALETIC, 2003; ANTONIOL et al., 2002), mas essas técnicas ainda dependem de dados específicos para treinamento. Além disso, de acordo com Maskeri, Sarkar e Heafield (2008), ainda são necessários especialistas no domínio estudado para certificar a qualidade das palavras-chave selecionadas, ao usar os métodos mencionados acima. Por esse motivo, optou-se por fazer uma pesquisa com vários especialistas em arquitetura de software para criar um conjunto de palavras-chave, pois se acredita que eles estão familiarizados com esse tipo de discussão e porque são pesquisadores desse tema. De acordo com Guan e Wong (1999), o método de procurar informações, usando palavras-chave é eficiente em comparação com outras técnicas, embora possa não extrair todas as informações disponíveis. Assim, embora se acredite na qualidade do conjunto de palavras-chave produzido (atestado pelas estatísticas aplicadas nesse trabalho), é possível que, para melhorar a pesquisa de alguns tópicos específicos da arquitetura, seria necessária uma revisão mais extensa e até a adição de outras palavras-chave a esse conjunto.

3.3.2.1 Participantes do *Survey*

Foi enviado um convite para dez pesquisadores de grupos de pesquisa com os quais se tem contato. Apenas cinco deles aceitaram o convite e participaram da pesquisa. Todos eles têm doutorado em Ciência da Computação e são professores de universidades no Brasil ou nos EUA. Esses pesquisadores têm quinze anos de experiência em desenvolvimento de software, em média. Ao longo de suas carreiras, eles trabalharam com programação, arquitetura de software, modelagem e análise de sistemas. O participante mais experiente tem trinta anos de experiência e o menos experiente oito anos. Atualmente, todos eles são pesquisadores de arquitetura de software e de outros tópicos relacionados à engenharia

de software.

3.3.2.2 Coleta de dados

Para coletar as palavras-chave de cada participante, forneceu-se a cada um deles uma lista de palavras-chave sugeridas, agrupadas em quatro categorias: (i) Linguagens de Descrição Arquitetural (ADL) - palavras-chave extraídas das cinco ADLs mais citadas no Google Scholar; (ii) Requisitos Não-Funcionais - os sete principais tópicos emergentes sobre requisitos não-funcionais no Google Scholar; (iii) Estilos Arquiteturais - estilos arquiteturais mencionados nos dois livros de arquitetura de software mais citados no Google Scholar; (iv) Restrições Arquiteturais e termos relacionados - verbetes constantes no índice remissivo dos livros citados acima. A lista[2] contém 368 palavras-chave como sugestões para os participantes da pesquisa.

Foi solicitado a cada participante que selecionasse entre as 368 palavras-chave sugeridas, aquelas que considerou relacionadas à arquitetura de software. Também foi solicitado que fosse informada qualquer outra palavra-chave não sugerida na lista, mas que eles considerassem como relacionada à arquitetura de software.

3.3.2.3 Análise de concordância entre os conjuntos

Para avaliar o grau de concordância entre os conjuntos de palavras-chave obtidas pelos participantes recorreu-se ao índice *kappa* (COHEN, 1968). Esse índice (*kappa*) avalia o nível de concordância de um conjunto de avaliações realizadas por diferentes juízes (no caso em particular os participantes do *survey* que tiveram o propósito de selecionar palavras-chave dentre o conjunto apresentado). O índice considera rejeições e aceitações. No caso em tela, consideraram-se todas as palavras-chave e todas as avaliações quanto a estarem ou não relacionadas à arquitetura de software. Para definir o conjunto de palavras-chave usadas neste estudo, foram consideradas àquelas apontadas por pelo menos 80% dos participantes (ou seja, por pelo menos quatro participantes). Adotou-se esse limiar porque a pontuação obtida no índice *kappa* foi de 0,271 (um *fair agreement* de acordo com a escala de Landis e Kock (1977)).

3.3.3 O conjunto de palavras-chave

A Tabela 3.1 mostra a quantidade de palavras-chave selecionadas por cada especialista. Pode-se observar na Tabela 3.2 que, das 368 palavras-chave sugeridas aos especialistas no formulário, 176 não foram selecionadas por nenhum deles. Por outro lado, 192 palavras-chave foram selecionadas por ao menos um especialista. A escolha deste estudo foi considerar as palavras-chave selecionadas por pelo menos quatro especialistas, e assim, o conjunto de palavras-chave construído incluiu 32 palavras-chave (26 selecionadas por quatro especialistas e 6 selecionadas pelos cinco especialistas).

Como uma amostra de cinco participantes é à princípio pequena, replicou-se o *survey*

[2]O Apêndice C apresenta o conjunto completo de palavras-chaves utilizadas neste estudo.

Table 3.1: Quantidade de palavras-chave indicada por cada especialista.

Especialista	**Número de palavras-chave selecionadas**
Especialista 1	16
Especialista 2	70
Especialista 3	54
Especialista 4	149
Especialista 5	151

Table 3.2: Quantidade de palavras-chave indicada por número de especialistas.

Número de indicações	**Quantidade de palavras-chave**
Nenhum especialista	176
1 especialista	44
2 especialistas	86
3 especialistas	30
4 especialistas	26
5 especialistas	6

com nove estudantes de pós-graduação em Ciência da Computação e que pesquisam na área de Engenharia de Software. Esses estudantes têm alguns anos de experiência em desenvolvimento de software (nove anos em média), trabalhando com design, arquitetura ou desenvolvimento de software. Eles foram escolhidos porque eram de fácil acesso e tinham experiência com pesquisa em arquitetura de software e com desenvolvimento de software.

Para a construção desse conjunto, foram consideradas as palavras-chave selecionadas por pelo menos sete estudantes. Assim, o conjunto de palavras-chave obtidas pelos estudantes incluiu 39 palavras-chave, sendo: 22 selecionadas por sete estudantes, 13 selecionadas por oito e quatro selecionadas por todos eles. Em seguida, foi estabelecida uma comparação entre o conjunto de palavras-chave obtidas junto aos especialistas (primeira linha da Tabela 3.3) e o conjunto de palavras-chave obtidas dos alunos de pós-graduação (segunda linha da Tabela 3.3) e verificou-se que são similares. Na Tabela 3.3, estão destacadas em negrito as palavras-chave selecionadas pelos dois grupos de participantes. Para confirmar a similaridade, submeteu-se os dois conjuntos (palavras-chave de especialistas e palavras-chave de estudantes) ao índice de Jaccard (REAL; VARGAS, 1996) de similaridade e obteve-se um índice de 0,58. Esse índice indica que os dois conjuntos de palavras-chave são 60% semelhantes. Posto isso, decidiu-se usar apenas o conjunto de palavras-chave selecionadas pelos especialistas para as próximas etapas deste estudo.

É importante destacar que optou-se não solicitar aos colaboradores do KDELibs que validassem as palavras-chave, pois no passado, tem-se registro de dificuldades na obtenção de feedback dos colaboradores de outros projetos de Software Livre. De fato, Capra et al. (2009) relatam uma taxa de resposta muito baixa por parte de colaboradores de projetos de Software Livre nesse tipo de estudo: apenas 8,5% responderam frente à 8.780 convites.

Conforme descrito na Seção 3.3.1, para alimentar o processo de filtragem dos *commits*, foram extraídos os logs de mensagens do VCS do projeto KDELibs e o resultado dessa

Table 3.3: Palavras-chave indicadas por especialistas e estudantes de pós-graduação.

Tipo de participante	Conjunto de Palavras-chave
Especialistas	**adaptability "architecture style" architecture** changeability **client-server components connector constraints extends extensibility facade interface** iterator **layering** localizability "low coupling" maintainability modifiability modularity modularization module **package pipe-and-filter portability** provides **publish-subscribe reliability requires scalability "separation of concerns"** stability **subcomponents testability**
Estudantes	abstraction adaptability "architecture style" architecture availability client-server compatibility components configurability connections connector constraints "coupling and cohesion" encapsulation extends facade "failure tolerance" interface interoperability layering maintainability modifiability modularity modularization module package performance portability relationship reliability requires reusability scalability security "separation of concerns" subcomponents subprograms traceability variability

extração foi oferecido como entrada para a ferramenta ATM (Architecture Traces Mining). Essa ferramenta analisa cada mensagem de *commit* em busca das palavras-chave do conjunto descrito na Subseção 3.3.3. Antes dessa comparação, a ATM faz a extração do radical de cada palavra-chave para obter sua palavra primitiva, usando o algoritmo de Porter (1980).

Para mostrar a vantagem de executar o *stemming* (radicalização das palavras-chave), estabeleceu-se a comparação do número de mensagens de *commit* encontradas usando as palavras-chave originais e suas versões primitivas. A Tabela 3.4 mostra os resultados dessa comparação. Conforme informado na Seção 3.3.1, primeiro pesquisou-se automaticamente por mensagens de *commit* contendo as palavras-chave selecionadas e, posteriormente, realizou-se a filtragem manual dessas mensagens. Portanto, a Tabela 3.4 mostra a comparação de palavras-chave originais (OK) e das palavras-chave primitivas (SK) em termos de número de mensagens de *commit* encontrados automaticamente (AFC) e número de mensagens de *commit* confirmadas manualmente (MCC).

Pode-se observar na Tabela 3.4 como o uso dos radicais foi útil. Em todos os casos, as palavras-chave derivadas detectaram mais mensagens de *commit* do que as palavras-chave originais correspondentes. Somente para a palavra-chave *testability*, a palavra-chave derivada foi menos eficiente. O uso da técnica radicalização (*stemming*) resultou na identificação de 114 *commits* arquiteturais que não seriam identificados com o uso das palavras-chave originais.

Em alguns casos, uma mensagem de *commit* contém mais de uma palavra-chave do conjunto utilizado. A Tabela 3.5 mostra a relação entre o número de palavras-chave e o número de mensagens de *commit* encontradas automaticamente (AFC) e o número de mensagens de *commit* confirmadas manualmente (MCC).

Table 3.4: Comparação entre as mensagens detectadas pela ferramenta ATM usando as palavras-chave originais e suas respectivas palavras-chave primitivas.

		OK		SK	
OK	**SK**	**AFC**	**MCC**	**AFC**	**MCC**
adaptability	adapt	0	0	205	5
architectural style	architectur style	0	0	9	4
architecture	architectur	8	3	9	4
changeability	changeabl	0	0	3	0
client-server	client-server	0	0	0	0
components	compon	47	7	195	28
connector	connector	1	0	1	0
constraints	constraint	51	3	72	4
extends	extend	6	0	277	5
extensibility	extens	0	0	156	8
facade	facade	0	0	0	0
interface	interfac	464	34	467	35
iterator	iter	123	3	294	10
layering	layer	1	0	96	5
localizability	localiz	0	0	52	2
“low coupling”	“low coupl”	0	0	0	0
maintainability	maintain	2	0	84	5
modifiability	modifi	0	0	197	12
modularity	modular	1	0	7	1
modularization	modular	1	0	7	1
module	modul	430	32	442	33
package	packag	317	26	331	26
pipe-and-filter	pipe-and-filt	0	0	0	0
portability	portabl	5	1	34	1
provides	provid	62	6	534	41
publish-subscribe	publish-subscrib	0	0	0	0
reliability	reliabl	3	0	40	4
requires	requir	83	10	470	45
scalability	scalabl	1	0	352	0
“separation of concerns”	“separation of concerns”	1	0	1	0
stability	stabil	9	1	15	1
subcomponents	compon	0	0	195	28
testability	testabl	1	0	0	0
TOTAL		**1576**	**118**	**3919**[1]	**232**[2]

[1] O número total de *commits* encontrados automaticamente foi de 3919. Como em alguns casos essas mensagens continham mais de uma palavra-chave, o resultado não é uma soma direta das linhas da tabela.

[2] O total de número de *commits* confirmados manualmente foi 232, pois em alguns casos uma mensagem de *commit* continha mais de uma palavra-chave, portanto, o resultado não é uma soma direta das linhas da tabela.

3.3.4 Pesquisando informações arquiteturais

Uma vez definido o conjunto de palavras-chave relacionadas à arquitetura, esse conjunto foi utilizado para extrair as mensagens de *commit* do projeto KDELibs tendo em vista a identificação de *commits* relacionados a alterações arquiteturais.

3.3.4.1 O projeto KDELibs

Table 3.5: Mensagens de *commit* identificadas automaticamente (AFC) e Número de mensagens de *commit* confirmadas manualmente (MCC) em relação às palavras-chave.

Número de palavras-chave	AFC	MCC
1	3554	197
2	321	27
3	38	7
4	4	1
5	2	0

O projeto KDELibs foi selecionado para esse estudo por algumas razões: (i) é um projeto grande em termos de tamanho do código-fonte, (ii) é central no ecossistema do KDE, (iii) está ativo há muito tempo (pouco mais de vinte anos) e (iv) possui vários colaboradores. É desenvolvido majoritariamente em C/C++. Na última versão analisada, ele possui cerca de 630 KLOC em C++ e 144 KLOC em C. O período analisado tem pouco mais que dez anos, entre 8 de maio de 2006 e 5 de dezembro de 2017. Durante esse período, o projeto KDELibs continha: entre 820 KLOC (*commit* c63b20) e 968 KLOC (*commit* 2073eb), (ii) 652 colaboradores que registraram *commits* e (iii) entre 3.077 (*commit* c63b20) e 4.238 arquivos de código-fonte (*commit* 2073eb).

3.3.4.2 A ferramenta Architecture Traces Mining

A ferramenta ATM (Architecture Traces Mining) foi desenvolvida para apoiar o processo de mineração de repositórios. Ela foi implementada em Java e tem como entrada um conjunto de palavras-chave e as mensagens de *commit* exportadas de um VCS em formato de arquivo texto (`.txt`). Durante o processo de execução, no passo seguinte, a ferramenta executa a radicalização de cada palavra-chave. Por fim, ela usa o conjunto de palavras-chave derivadas para pesquisar as mensagens de *commit*.

Como saída, a ferramenta ATM oferece um conjunto de mensagens de *commit* que contém as palavras-chave do conjunto de entrada e, portanto, contêm informações arquiteturais segundo as suposições deste trabalho. A saída é mostrada em uma interface gráfica ao usuário que permite ao mesmo informar se concorda que a mensagem de *commit* apresentada contém informações arquiteturais ou não. Após serem selecionadas, essas informações podem ser salvas em formato de arquivo texto ou conteúdo separado por vírgulas (`.csv`).

3.3.4.3 Tópicos de Informações Arquiteturais

Para ajudar a responder à **RQ1.3** (descrita na Seção 3.1), foi definida uma taxonomia de tópicos arquiteturais para classificar os diferentes aspectos das informações arquiteturais. Para criar esta lista, buscou-se classificar esses aspectos a partir de três definições clássicas de arquitetura de software: uma definição focada em aspectos estruturais (PERRY; WOLF, 1992), uma definição mais abrangente e detalhada (GARLAN; SHAW, 1993) e

uma definição que complementa as outras duas (BASS; CLEMENTS; KAZMAN, 1998).

Para Perry e Wolf (1992), arquitetura de software é um conjunto de elementos de design de arquitetura classificados como elementos de processamento, elementos de dados e elementos de conexão. Eles também consideram como tópicos arquiteturais os objetivos, restrições e motivações para decisões arquiteturais. Esses aspectos, em geral, persistem na maioria das outras definições e abordagens.

Garlan e Shaw (1993) definem arquitetura de software como a estrutura geral de um sistema. Questões estruturais incluem organização geral e estrutura de controle global; protocolos de comunicação, mecanismos de sincronização e acesso a dados; atribuição de funcionalidades; distribuição física de elementos; escalabilidade e desempenho; e seleção entre alternativas de design.

Finalmente, também se considerou a definição clássica de Bass, Clements e Kazman (1998) que salienta que o design arquitetural de um sistema pode ser descrito por (pelo menos) três perspectivas: particionamento funcional de seu domínio de interesse, sua estrutura e a alocação da função de domínio para essa estrutura.

Como resultado, foram identificados quinze tópicos que direcionam a classificação das mensagens de *commit*. A partir da definição de Perry e Wolf (1992), foram definidos os seguintes tópicos arquiteturais:

(I) **Elementos Arquiteturais de Processamento**: Mensagens de *commit* sobre componentes ou agrupamentos de classes que processam entradas ou informações. Um exemplo desse tópico é a mensagem de *commit*: "Trabalho do módulo `kspread` `kross` portado feito na *akademy* na ramificação 1.6"[3]. `Kross` é uma estrutura de *script*.
Esta é uma mensagem arquitetural classificada como Elemento Arquitetural de Processamento, porque descreve a movimentação do módulo responsável pelo processamento de todos os *scripts* no componente `Kspread`.

(II) **Elementos Arquiteturais de Dados**: Mensagens sobre componentes/dados a serem transformados ou processados. Um exemplo desse tópico é a mensagem de *commit*: "Não consigo criar um `kcomponentdata` aqui, tenho que usar `s_instance` que não está registrado como o componentdata principal".
Esta é uma mensagem arquitetural classificada como Elemento Arquitetural de Dados porque descreve uma alteração do `KcomponentData` (um componente genérico dos dados do aplicativo associado).

(III) **Elementos Arquiteturais de Conexão**: Mensagens sobre componentes/interfaces que conectam componentes do sistema. Um exemplo desse tópico é a mensagem de *commit*: "Porta para a alteração do `kjob`, permitindo gerenciar várias unidades". O tipo `KJob` representa uma operação que pode ser executada de forma assíncrona.
Esta é uma mensagem arquitetural classificada como Elemento Arquitetural de Conexão, porque descreve uma alteração do `Kjob` (uma interface da biblioteca

[3]Todas as mensagens utilizadas como exemplo de tópicos arquiteturais originalmente encontravam-se em inglês e foram traduzidas para o português no escopo deste trabalho

KDELibs que define como deve ser o tipo de classe de tarefa nos aplicativos associados) por precisar expandir esse componente.

(IV) **Motivações, Objetivos e Restrições**: Mensagens sobre a lógica do projeto, implementação, links, objetivos e restrições aplicadas aos componentes do sistema. Um exemplo desse tópico é a mensagem de *commit*: "Quaisquer exceções usadas nunca cruzam a API. Ou seja, o código do cliente deve ter certeza de que o código da biblioteca subjacente não lançará nenhuma exceção quando chamado pela API pública".
Esta é uma mensagem arquitetural classificada como Motivação, Objetivos e Restrições, pois descreve uma restrição de uso da API do KDELibs no uso de exceções.

A partir da definição de Garlan e Shaw (1993), formaram-se os seguintes tópicos:

(V) **Estruturas Globais de Controle**: Mensagens sobre a estrutura global de controle do sistema. Um exemplo desse tópico é a seguinte mensagem de *commit*: "O conceito de proxies foi completamente removido. `resourcedata` agora mantém uma lista de recursos que contêm `*resourcedata::determinuri` foi modificado significativamente".
Esta é uma mensagem arquitetural classificada como Estrutura Global de Controle porque descreve a remoção de um conceito de fluxo de informações da biblioteca KDELibs, neste caso, o conceito de proxy.

(VI) **Estruturas do Sistema**: Mensagens sobre estruturas gerais do sistema. Aqui, estão classificadas as mensagens que referem-se aos componentes abstraídos no nível superior. Como exemplo, considere uma mensagem de *commit*: "(...) código usando o projeto `iso-codes` como fonte de validação e tradução está sendo desenvolvido no `kdepimlibs` e no `kdebase` e será movido para o KDELibs no 4.7 uma vez que os requisitos e a dependência da API implicações são melhores compreendidas".
Esta é uma mensagem arquitetural classificada como Estruturas do Sistema, porque descreve a alteração da funcionalidade que implica em todo o projeto e foi migrada para uma nova estrutura geral.

(VII) **Protocolos de Comunicação**: Mensagens sobre Protocolos de Comunicação no sistema. Um exemplo desse tópico é a mensagem de *commit*: "Além dos protocolos locais, não procure informações de proxy se um URL não tiver componente de host, por exemplo o protocolo de dados".
Esta é uma mensagem arquitetural classificada como Protocolos de Comunicação, porque descreve uma alteração que implica o uso do protocolo de dados do KDELibs.

(VIII) **Mecanismos de Sincronização**: Mensagens que mencionam a sincronização de elementos ou dispositivos representados como um módulo. Um exemplo desse tópico é a mensagem de *commit*: "Garantir que o `slotchanged` seja chamado em todas as instâncias, mova a chamada `slotchanged` para o *slot* `broadcastdone` para que seja chamado em todas as instâncias do dispositivo; de fato, `broadcastdone` é chamado pelo `dbus`. Isso garante que o estado do dispositivo seja atualizado

corretamente quando modificado por alguma rotina fora de processo, por exemplo, quando montada por meio de ações `kde-runtime/soliduiserver`".
Esta é uma mensagem arquitetural classificada como Mecanismo de Sincronização, porque descreve uma alteração que implica na integridade (atualização para todos em tempo real) de todas as instâncias que usam o objeto compartilhado (no caso objeto com alteração de *slot*).

(IX) **Mecanismos de Acesso a Dados**: Mensagens que mencionam mecanismos para acessar dados de componentes. Um exemplo desse tópico é a mensagem de *commit*: "Embora o `kfilemetadataprovider` garanta que `kloadmetafilemetadatathread::load()` ou `kloadmetafilemetadatathread::data()` não seja chamado durante a execução do encadeamento, é necessário proteger os dados compartilhados".
Esta é uma mensagem arquitetural classificada como Mecanismo de Acesso a Dados, porque discute a necessidade de proteger os dados compartilhados presentes nos objetos citados.

(X) **Atribuições de Recursos**: Mensagens que tratam das atribuições de recursos a componentes. Um exemplo desse tópico é a mensagem de *commit*: "Funcionalidade `guimanager` movida para o módulo `scriptmanager`".
Esta é uma mensagem arquitetural classificada como Atribuições de Recursos, porque alerta que a funcionalidade "`guimanager`" foi movida para outro módulo (`scriptmanager`) do projeto KDELibs.

A partir da definição de Bass, Clements e Kazman (1998) foram produzidos os seguintes tópicos:

(XI) **Relação entre Elementos**: Mensagens que mencionam a dependência entre módulos ou bibliotecas. A mensagem de *commit* "O `kio` agora usa o `nepomuk` em algum lugar, portanto, é necessário incluir o `soprano` por extensão". É um exemplo desse tópico.
Esta é uma mensagem arquitetural classificada como "Relação entre Elementos", porque notifica que o módulo Kio precisa do componente "`soprano`" pelo uso da interface `Nepomuk`, explicitando, assim, a relação entre esses componentes.

(XII) **Organização Fundamental**: Mensagens sobre a Organização Fundamental dos elementos de um sistema. Nesse tópico, estão classificadas as mensagens que se referem à organização lógica dos componentes em geral. Como exemplo, considere a seguinte mensagem de *commit* extraída pelo KDELibs: "Adequação da estrutura do pacote para pacotes dinâmicos".
Essa é uma mensagem arquitetural classificada como Organização Fundamental porque justifica a alteração com uma nova diretiva na qual os pacotes do KDELibs devem ser dinâmicos.

Por fim, foram adicionados outros três tópicos identificados na literatura sobre arquitetura de software (BUSCHMANN et al., 1996; CLEMENTS et al., 2010):

(XIII) **Requisitos Não-Funcionais**: Mensagens sobre Requisitos Não-Funcionais, como segurança, concorrência, tempo de resposta e assim por diante. Como um exemplo, considere a seguinte mensagem de *commit* extraída do projeto KDELibs: "Use o algoritmo `fisher-yates` para randomizar listas, isso garante que o tempo necessário para o embaralhamento dependa linearmente do comprimento da lista, e não quadraticamente. Além disso, evita alocações de memória durante o embaralhamento e economiza tempo, mesmo em listas muito curtas".
Esta é uma mensagem arquitetural classificada como um requisito não-funcional, porque justifica a alteração pela necessidade de economizar tempo e melhora a experiência do usuário.

(XIV) **Estilos Arquiteturais**: Mensagens que se referem aos Estilos Arquiteturais possivelmente adotados no projeto. Como um exemplo, é oportuno considerar a mensagem de *commit* extraída do projeto KDELibs: "Alterne o sistema `kcalendarsystem` para usar uma arquitetura compartilhada".
Esta é uma mensagem arquitetural classificada como Estilos Arquiteturais, porque justifica a alteração do componente "`Kcalendarsystem`" para arquitetura compartilhada.

(XV) **Outros elementos relacionados à arquitetura**: Mensagens sobre arquitetura de software, mas que não se enquadram nos tópicos anteriores. Por exemplo, ao considerar a seguinte mensagem de *commit* extraída do projeto KDELibs: "Adicione uma variável `cmake disablealloptionalsubdirectories` que pode ser configurada como `true` para desativar todos os subdiretórios por padrão, para facilitar o projeto de construção de maneira modular".
Mensagem arquitetural classificada como Outros elementos relacionados à Arquitetura, porque trata-se das mudanças que visam facilitar a modularização do sistema.

A definição dessa taxonomia colabora com a delimitação de quais informações da arquitetura de um projeto estão sendo priorizadas, uma vez que, conforme já dissertado nesse trabalho, trata-se de um assunto amplo e sem um consenso do que faz parte do projeto arquitetural e do que contempla apenas aspectos de design, mais próximos do que é materializado no código-fonte. Ainda sobre esses tópicos, é preciso destacar a independência entre eles, uma vez que por serem oriundos de diferentes definições de arquitetura, esse sombreamento poderia ocorrer. Tome-se como exemplo os tópicos "Estruturas do Sistema" e "Organização Fundamental". Enquanto os traços arquiteturais identificados como relacionados ao tópico "Estruturas do Sistema" estão relacionados aos componentes com alto nível de abstração, os traços arquiteturais relacionados ao tópico "Organização Fundamental" descrevem a lógica de organização dos componentes no projeto.

3.3.4.4 Metodologia para responder às perguntas da pesquisa

Para analisar as mensagens selecionadas pela ferramenta ATM, cada mensagem de *commit*, apontada por essa ferramenta, foi inspecionada manualmente, a fim de determinar se ela descreve uma mudança arquitetural ou não. Em caso afirmativo, classificou-se em

um tópico arquitetural específico entre os definidos na subseção anterior. A Figura 3.2 mostra um exemplo de mensagem de *commit* que foi classificada como "Mecanismos de Sincronização". Ela fala sobre a sincronização entre um dispositivo e as rotinas. A Figura 3.3 ilustra uma mensagem de *commit* que foi classificada no tópico "Relação entre Elementos". Ela menciona a necessidade de uma biblioteca. A última etapa deste estudo foi analisar o conjunto de *commits* obtido para responder às perguntas da pesquisa (apresentadas na Seção 3.1).

```
Ensure slotChanged is called in all instances

Move slotChanged call to the broadcastDone slot
so that it is called in all instances of the device; in fact broadcastDone
is called by dbus.  This makes sure that the device state is correctly updated
when modified by some out-of-process routine, e.g. when mounted via
kde-runtime/soliduiserver actions.

This *should* fix bug 268020, please check
CCBUG: 268020
```

Figure 3.2: Exemplo de mensagem de *commit* com conteúdo arquitetural - Tópico: Mecanismo de Sincronização.

```
Building Solid UPnP backend by default now. HUPnP library required.

svn path=/trunk/KDE/kdelibs/; revision=1168594
```

Figure 3.3: Exemplo de mensagem de *commit* com conteúdo arquitetural - Tópico: Relação entre Elementos.

A seguir, é apresentado como foi realizada a análise de cada subquestão de pesquisa:

- A **RQ1.1** pergunta se desenvolvedores/colaboradores escrevem sobre mudanças arquiteturais nas mensagens de *commit*. Para responder a essa pergunta, a ferramenta ATM foi executada, tendo como entrada os logs do repositório escolhido e, em seguida, foram analisadas todas as mensagens de *commit* identificadas, preliminarmente, selecionando aquelas que realmente contêm informações sobre arquitetura de software.

- A **RQ1.2** pergunta se há períodos que concentram *commits* cujas mensagens contêm informações arquiteturais. Analisou-se quando ocorreram *commits* relacionados à arquitetura e se eles ocorreram em período próximo ao lançamento de novas *releases* do projeto. Para isso, comparou-se o número médio de mensagens de *commit*, contendo informações arquiteturais e o número médio de mensagens de *commit* sem informações arquiteturais, por mês, tanto no período de novas liberações quanto nos demais períodos. Em seguida, foi testada a significância estatística desses dados usando o teste U de Mann-Whitney-Wilcoxon (MANN; WHITNEY, 1947; WILCOXON, 1992).

- A **RQ1.3** pergunta sobre os tópicos arquiteturais mais citados nas mensagens de *commit*. Esta subquestão de pesquisa também analisa quais tópicos arquiteturais

ocorrem no mesmo período de lançamento de novas *releases*, o intervalo entre as alterações registradas dos tópicos arquiteturais e quais palavras-chave detectam quais tópicos. Para definir o tópico arquitetural mais citado, contou-se o número de mensagens para cada ocorrência. Para apontar se diferentes tópicos ocorreram, ao mesmo tempo, observou-se a sua ocorrência na mesma janela de tempo. Definiu-se a janela de avaliação como três meses. Essa escolha se deu, porque se percebeu que os meses de lançamento de novas *releases* do projeto, bem como os meses de lançamento anteriores e subsequentes, tiveram mais *commits* do que a média mensal ao longo do projeto. O mesmo método foi usado para definir se há registros para cada tópico da arquitetura no momento do lançamento de novas *releases*. Por fim, para relacionar as palavras-chave com os tópicos arquiteturais disponíveis, contou-se a presença de cada palavra-chave nas mensagens identificadas para cada tópico.

- A **RQ1.4** pergunta sobre o perfil dos colaboradores que, de acordo com as mensagens de *commit*, executam alterações na arquitetura. Ela tenta distinguir dois grupos (colaboradores não-específicos e colaboradores arquiteturais), analisando atributos como a duração da participação no projeto, o número de *commits* gerais *versus* *commits* arquiteturais e o tempo médio entre alterações arquiteturais e alterações gerais para cada colaborador. Além disso, esta subquestão de pesquisa analisou o hiato temporal entre as contribuições de cada colaborador arquitetural. Para responder a **RQ1.4**, observou-se, numericamente, a quantidade de *commits* do autor da mensagem de *commit* com informações arquiteturais detectadas e comparadas com a participação geral dos autores. Também se mediu o período entre *commits* arquiteturais consecutivos realizados por cada responsável pela arquitetura e esses foram comparados ao período onde ocorreram os *commits* de uso geral.

- A **RQ1.5** pergunta se as mensagens de *commit* arquiteturais são descritas em mais detalhes do que outros tipos de mensagens de *commit*. Para responder **RQ1.5**, analisou-se o tamanho (em caracteres) das mensagens de *commit* classificadas como arquiteturais e estabeleceu-se uma comparação com o tamanho das outras mensagens de *commit*. A análise foi realizada duas vezes, uma delas removendo palavras irrelevantes antes de medir o tamanho da mensagem e a outra considerando todo o texto da mensagem original.

- A **RQ1.6** pergunta se os *commits* arquiteturais alteram mais arquivos do que os *commits* não-arquiteturais. Para responder a essa subquestão, observou-se o número de arquivos alterados por cada *commit* arquitetural e comparou os números obtidos com o número de arquivos alterados por *commits* sem modificações arquiteturais. Somente arquivos de código-fonte foram considerados.

- Finalmente, a **RQ1.7** pergunta se os *commits* arquiteturais tendem a alterar os mesmos arquivos repetidamente. Para responder a **RQ1.7**, contou-se quantos arquivos foram modificados repetidamente por *commits* arquiteturais e *commits* não-arquiteturais diferentes. Nessa comparação, considerou-se apenas arquivos de código-fonte, a exemplo da lógica utilizada para a análise da **RQ1.6**.

A significância estatística foi avaliada usando o teste de Mann-Whitney-Wilcoxon e, para definir o tamanho do efeito, foi usado o Delta de Cliff, interpretando $|Delta| < 0{,}33$ como efeito pequeno, $|Delta| < 0{,}474$ como efeito mediano e $|Delta| > 0{,}474$ como grande efeito (ROMANO et al., 2006).

3.4 RESULTADOS

Esta seção apresenta os resultados do estudo guiados pelas subquestões de pesquisa propostas. Neste estudo, foi considerado apenas o período entre 08/05/2006 e 05/12/2017 do desenvolvimento do KDELibs. Nesse período, o projeto KDELibs lançou cinco *releases* de *patches* na série 3.5.x (de 3.5.6 a 3.5.10) e cinco *releases* secundárias na série 4.x (de 4.0 a 4.4). Também se analisou cada uma das mensagens de *commit* para identificar o tópico arquitetural correspondente (de acordo com os tópicos apresentados na seção de design deste estudo), o perfil de seus autores, o tamanho das mensagens e o tamanho das alterações realizadas (em quantidade de caracteres, número de arquivos alterados).

A partir das próximas subseções, são apresentados os resultados obtidos pelo estudo empírico descrito na Seção 3.3. A apresentação dos resultados foi agrupada por cada subquestão de pesquisa através de subseções relacionadas.

3.4.1 Os desenvolvedores/colaboradores do projeto informam mudanças arquiteturais nas mensagens de commit?

A **RQ1.1** pergunta se os desenvolvedores/colaboradores do projeto informam alterações arquiteturais nas mensagens de *commit*. Neste estudo, detectou-se um conjunto de mensagens de *commit* com informações sobre mudanças na arquitetura. Com base no conjunto de palavras-chave usadas, a ferramenta ATM retornou *commits* potencialmente relacionados às alterações arquiteturais (3.919 de um total de 42.117 no período analisado). No entanto, após a inspeção manual, apenas 232 mensagens de *commit* foram confirmadas como relacionadas à arquitetura, o que representa apenas 0,6% do total minerado. As dúvidas relacionadas à existência de informações sobre a arquitetura do software foram superadas. Elas existiam, porque o objetivo das mensagens de *commit* é descrever as alterações no código do projeto. Os resultados obtidos identificaram mensagens de *commit*, contendo informações arquiteturais. Assim, confirmaram-se as mensagens de *commit* como fonte de informações arquiteturais.

> O projeto KDELibs possui *commits* com mensagens, nas quais os colaboradores informam sobre mudanças na arquitetura. Foram identificadas 232 mensagens de *commit* com essa característica.

No início desta subseção, foram apresentados os resultados obtidos na busca por comentários arquiteturais nas mensagens de *commit*. Esses resultados mostraram que apenas um pequeno número de mensagens possuía informações arquiteturais. Assim, ter mais elementos para ajudar a identificar esse tipo de mensagem diminuiria os falsos positivos a serem analisados ao procurar essas informações. A Figura 3.4 nos leva a crer que há períodos que concentram as mensagens de *commit* com informações arquiteturais. Esses períodos coincidem com os lançamentos de novas *releases*?

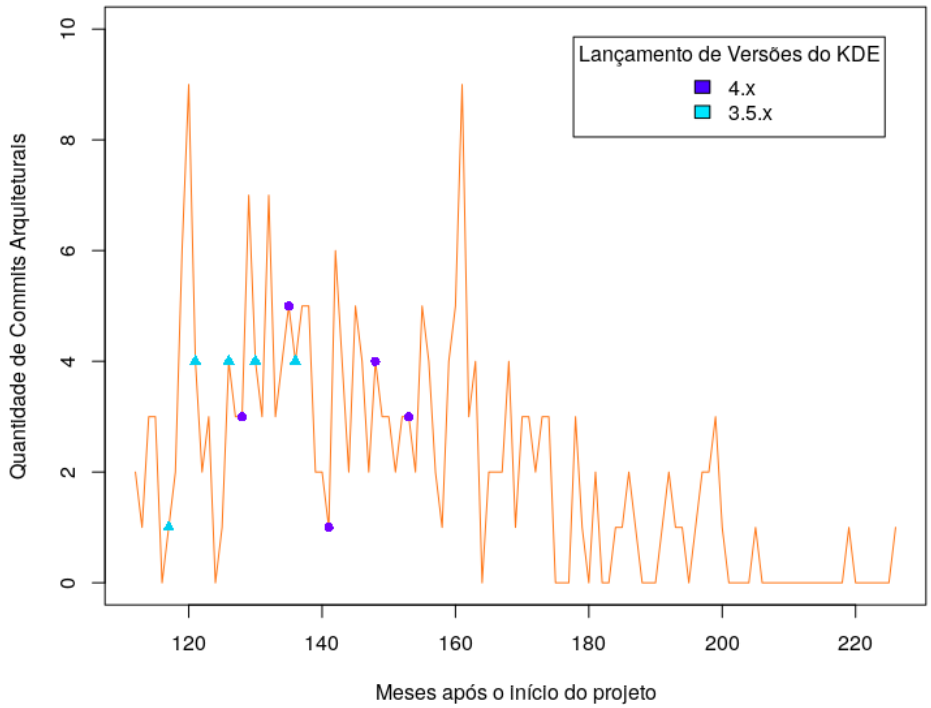


Figure 3.4: Número de *commits* arquiteturais identificados ao longo do projeto KDELibs.

3.4.2 Há períodos que concentram commits com alterações na arquitetura?

A **RQ1.2** pergunta se há períodos específicos durante a evolução do software que concentram *commits* relacionadas à alterações na arquitetura. A Figura 3.4 mostra o número de *commits* arquiteturais ao longo da evolução do projeto KDELibs. A Figura 3.5 mostra o acumulado de *commits* arquiteturais durante a evolução do projeto KDELibs. Os círculos azuis representam o lançamento das novas *releases* 4.x e os triângulos cianos representam o lançamento das novas *releases* 3.5.x. Quando se compara o lançamento de novas *releases* do KDELibs com picos de novas mensagens de *commit*, contendo informações arquiteturais, vê-se alguns picos que ocorreram perto de novos lançamentos (Figura 3.4).

Em alguns casos, os picos de *commits* ocorreram um mês antes de um novo lançamento. Em outros casos, esses picos ocorreram um mês após o lançamento ou precisamente no mesmo mês. As *releases* 4.1 e 4.3 (meses 135 e 146) são exemplos de *releases* com picos de *commits* arquiteturais próximas aos lançamentos. No entanto, a Figura 3.4 também mostra casos em que uma nova *release* foi lançada, mas nenhum ou poucos *commits* arquiteturais foram detectados no mesmo período. As *releases* 3.5.6 e 4.2 (meses 117 e 141) são exemplos dessa percepção. Finalmente, a Figura 3.4 também mostra picos de *commits* arquiteturais em períodos sem lançamento de novas *releases*.

A resposta para **RQ1.2**, no entanto, vem da aplicação e análise do teste U de Mann-Whitney-Wilcoxon. A comparação da média de mensagens de *commit* que ocorreram nos meses da janela de novos lançamentos com os meses de outros períodos revela que, em meses dentro do período de liberação, há, em média, pouco mais que o dobro de

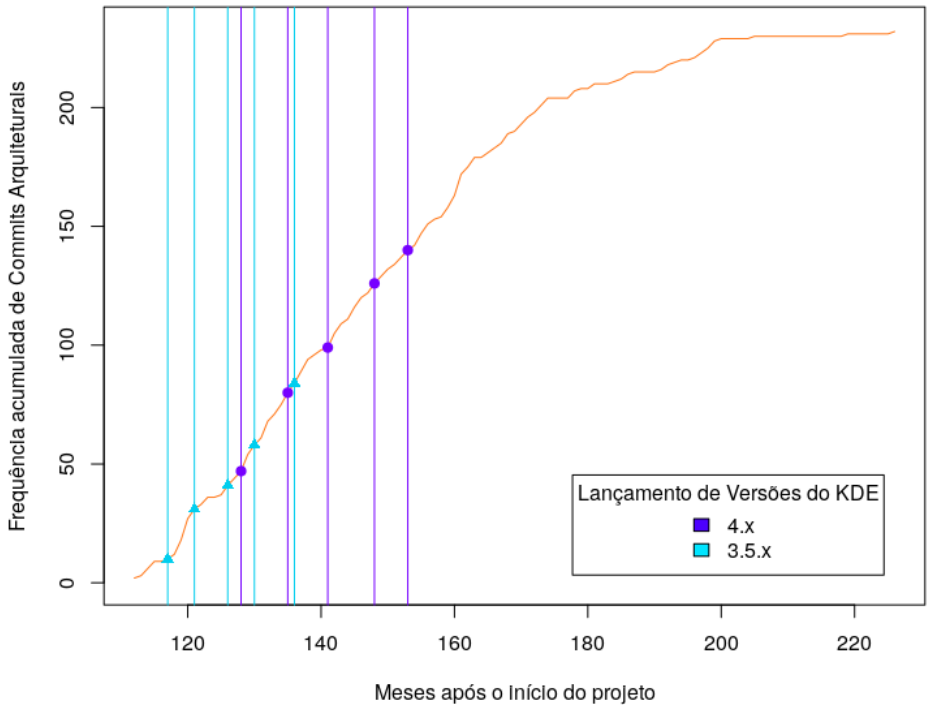


Figure 3.5: Frequência acumulada de *commits* arquiteturais identificados ao longo da evolução do projeto KDELibs.

commits com mensagens arquiteturais do que em outros meses. O teste de Mann-Whitney-Wilcoxon indicou um *p-value* de $4{,}452e^{-5}$ e um grande efeito (ΔCliff = 1,08). A Figura 3.6 ilustra a comparação das mensagens de *commit* arquiteturais nos meses dos períodos de liberação e nos outros meses. Porém, é importante destacar que a maioria das mensagens de *commit* detectadas durante o projeto ocorreu fora do período de liberação (144 mensagens de *commit* em 110 meses versus 88 mensagens de *commit* em 30 meses do período de liberação).

Também observou-se que, após o lançamento da última *release* do KDELibs, o número de mudanças na arquitetura diminuiu. Essa tendência descendente de novos *commits* arquiteturais pode ser explicada pela decisão da equipe do KDE de criar um novo projeto, chamado KF5, que compreende uma reescrita completa do KDELibs, o que, no entanto, não representou um abandono imediato do projeto. Como consequência, nenhuma nova *release* do KDELibs foi lançada após o 153º mês do projeto, embora o projeto ainda receba correções de bugs.

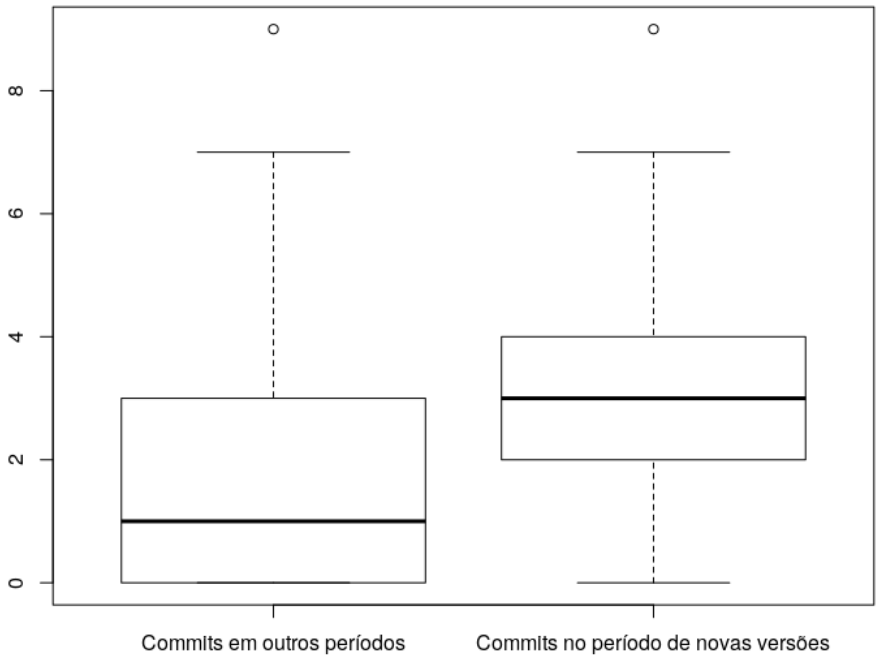


Figure 3.6: Comparação da quantidade de *commits* arquiteturais identificados em meses no período de lançamento versus *commits* arquiteturais em outros períodos.

Assim, a resposta para a **RQ1.2** é:

No projeto KDELibs, os períodos de lançamento de novas *releases* concentram mensagens de *commit* com informações arquiteturais.

Os meses dentro do período de liberação têm, em média, pouco mais que o dobro de *commits* arquiteturais que os demais meses do projeto. Esse resultado mostra que os colaboradores discutem a arquitetura do projeto antes do lançamento de novos lançamentos - e isso é explicado pelo planejamento de lançamentos futuros – ou após os lançamentos – e isso pode estar relacionado ao feedback recebido após o uso dessas atualizações no projeto. Não foram buscados esses motivos. Os resultados, no entanto, também mostram que a maioria dos *commits* com informações arquiteturais estão fora do período de lançamento de novas *releases*. Foram 144 mensagens distribuídas em 110 meses, em comparação com 88 mensagens em 30 meses nos períodos de lançamento de novas *releases*.

Em busca de mais informações sobre os *commits* com informações arquiteturais identificados e confirmada a hipótese de que o lançamento de novas *releases* é crucial para o registro de informações arquiteturais nessas mensagens, questiona-se: quais tópicos arquiteturais são mais discutidos? É provável que algum tópico arquitetural seja mais frequente próximo ao lançamento de novas *releases*?

3.4.3 Quais são os tópicos arquiteturais mais citados nas mensagens de commit sobre mudanças na arquitetura?

A **RQ1.3** pergunta quais tópicos arquiteturais são mais citados nas mensagens de *commit* relacionadas às mudanças na arquitetura. Como declarado anteriormente, cada mensagem confirmada como arquitetural foi classificada manualmente quanto ao tópico mencionado. A Tabela 3.6 mostra a quantidade de mensagens de *commit* por tópico. Percebe-se a prevalência dos tópicos "Relação entre Elementos" (60 mensagens de *commit*) e "Mecanismos de Sincronização" (43 mensagens de *commit*) em comparação aos demais tópicos. Por outro lado, "Elementos Arquiteturais de Dados" e "Estilos Arquiteturais" são os tópicos menos citados.

Table 3.6: Quantidade de *commits* arquiteturais encontrados no projeto KDELibs para cada tópico.

Tópico Arquitetural	**Número de *commits***
Relação entre Elementos	60
Mecanismos de Sincronização	43
Estruturas do Sistema	23
Requisitos Não-Funcionais	20
Protocolos de Comunicação	19
Elementos Arquiteturais de Conexão	13
Atribuições de Recursos	12
Mecanismos de Acesso a Dados	11
Motivações, Objetivos e Restrições	7
Outros elementos relacionados à arquitetura	7
Elementos Arquiteturais de Processamento	6
Estruturas Globais de Controle	6
Organização Fundamental	3
Elementos Arquiteturais de Dados	1
Estilos Arquiteturais	1

Para capturar as associações entre os diferentes tópicos arquiteturais, considerando a co-ocorrência entre eles, usou-se o algoritmo A-Priori com o software Weka[4] (HALL et al., 2009). O ponto de corte utilizado para as associações foi um nível de confiança de 0,6 e um máximo de 19.000 regras de associação. A maioria das associações relatadas pelo algoritmo foram regras de não co-ocorrência. Foram separados alguns exemplos de não co-ocorrência, como o existente entre o tópico "Elementos Arquiteturais de Dados", implicando na não co-ocorrência do tópico "Organização Fundamental" e vice-versa (confiança de 0,97). Por outro lado, foram encontradas 113 regras de associação para a ocorrência concomitante de tópicos arquiteturais. A maioria deles está associada aos tópicos "Mecanismos de Sincronização" (73) e "Relação entre Elementos" (37).

Por fim, foram analisadas outras associações considerando os mesmos dados e com uma janela deslizante que considerou o mês anterior ao lançamento do *commit*, o mês subsequente e o próprio mês. Estabeleceu-se o ponto de corte de metade das ocorrências de cada tópico como o limite mínimo a ser considerado. Assim, observa-se que:

[4]https://www.cs.waikato.ac.nz/ml/weka/

- (i) metade das ocorrências do tópico "Requisitos Não-Funcionais" precedem a ocorrência do tópico "Relação entre Elementos";

- (ii) 2/3 das ocorrências do tópico "Estruturas Globais de Controle" sucedem ocorrências do tópico "Relação entre Elementos";

- (iii) metade das ocorrências do tópico "Atribuições de Recursos" sucedem os tópicos "Relação entre Elementos" e "Mecanismos de Sincronização", e;

- (iv) 71% de ocorrências do tópico "Outros Elementos Relacionados à Arquitetura" sucedem ocorrências do tópico "Relação entre Elementos".

Em suma, ao ser examinado se havia algum relacionamento de co-ocorrência entre os diferentes tópicos da arquitetura, observa-se que os tópicos "Mecanismos de Sincronização" e "Relação entre Elementos" co-ocorrem com a maioria dos outros tópicos, sendo, frequentemente, seguidos pelos tópicos "Estruturas Globais de Controle", "Atribuições de recursos" e "Outros elementos relacionados à arquitetura", bem como precedidos pelo tópico "Requisitos não-funcionais".

Além disso, a Tabela 3.7 mostra a quantidade de mensagens obtidas por cada tópico registradas no período de lançamento de novas *releases* do projeto KDELibs. Como se pode observar nessa tabela, os tópicos mais registrados no período de lançamento de novas *releases* são: "Relação entre Elementos", "Mecanismos de Sincronização", "Estruturas do Sistema" e "Mecanismos de Acesso a Dados". É preciso um destaque para o tópico "Mecanismos de Acesso a Dados", pois todas as ocorrências a ele relacionadas foram registradas no mesmo período de lançamento de novas *releases*. Também destaque-se o tópico "Requisitos Não-Funcionais", pois 2/3 de seus registros ocorreram fora desses períodos. A Tabela 3.8 complementa os dados dos tópicos arquiteturais ocorridos no lançamento de novas *releases* do projeto (coluna CCLNV). Ela apresenta a porcentagem de mensagens localizadas no período de lançamento de novas *releases* por tópico.

Também foi analisada a frequência com que cada tópico ocorre. A Tabela 3.8 mostra a periodicidade média em que os tópicos arquiteturais aparecem ao longo da evolução do projeto (coluna ATOAC). Nesse aspecto, percebe-se que a periodicidade pode ser sumarizada em três grupos: mudanças frequentes (de um a 4 meses em média), mudanças ocasionais (de 5 a 20 meses em média) e mudanças esporádicas (acima de 20 meses em média). Destaque-se o tópico "Relação entre Elementos" como um tópico frequentemente alterado (uma alteração mensal em média). Por outro lado, observou-se que o tópico "Atribuição de Recursos" é um exemplo de tópico com alterações ocasionais (uma alteração a cada 15 meses). Por fim, o tópico "Estruturas Globais de Controle" é um exemplo de mudanças esporádicas, o que já era esperado, uma vez que é importante produzir Estruturas Globais de Controle estáveis.

Table 3.7: Quantidade de *commits* arquiteturais sumarizado por tópico no período de lançamento de novas *releases* do projeto KDELibs.

Tópico Arquitetural	***Release***	**Quantidade**
Relação entre Elementos[6]	4.x	15
	3.5.x	17
Mecanismos de Sincronização[2]	4.x	12
	3.5.x	5
Estruturas do Sistema[3]	4.x	6
	3.5.x	6
Mecanismos de Acesso a Dados[4]	4.x	4
	3.5.x	7
Protocolos de Comunicação[1]	4.x	2
	3.5.x	4
Requisitos Não-funcionais	4.x	3
	3.5.x	3
Elementos Arquiteturais de Conexão[1]	4.x	3
	3.5.x	2
Motivações, Objetivos e Restrições[1]	4.x	1
	3.5.x	2
Elementos Arquiteturais de Processamento	4.x	1
	3.5.x	1
Atribuição de Recursos	4.x	1
	3.5.x	2
Estilos Arquiteturais	4.x	1
Organização Fundamental	3.5.x	1
Estrutura Global de Controle	4.x	1
	3.5.x	1
Elementos Arquiteturais de Dados	4.x	1
Outros elementos relacionados à Arquitetura[1]	4.x	1
	3.5.x	3

[1]Uma mensagem de *commit* registrada na interseção entre os períodos de lançamento.
[2]Duas mensagens de *commit* registrada na interseção entre os períodos de lançamento.
[3]Três mensagens de *commit* registrada na interseção entre os períodos de lançamento.
[4]Quatro mensagens de *commit* registrada na interseção entre os períodos de lançamento.
[6]Seis mensagens de *commit* registrada na interseção entre os períodos de lançamento.

A Tabela 3.8 também mostra os tópicos arquiteturais citados em *commits* ocorridos no período de lançamento de novas *releases* do projeto (coluna CCLNV). Ao analisar essa tabela, é possível identificar dois tópicos que foram detectados apenas uma vez ao longo de todo o projeto e que coincidiu com o período de lançamentos de novas *releases* do projeto: "Elementos Arquiteturais de Dados" e "Estilos Arquiteturais". Além disso, o tópico "Mecanismos de acesso a dados", um tópico classificado como alteração esporádica, concentrou todas as suas ocorrências no período de lançamento de novas *releases*. No entanto, o tamanho da amostra não permite fazer uma inferência estatística entre eles. Entre os tópicos arquiteturais que sofreram alterações frequentes, somente o tópico "Relação entre Elementos" registrou mais da metade de suas ocorrências no período de lançamento de novas *releases* (53%). Os tópicos arquiteturais com alterações ocasionais não tiveram *commits* registrados no período de lançamento de novas *releases*.

Ainda em relação à **RQ1.3**, buscou-se estabelecer a relação entre as palavras-chave contidas nas mensagens de *commit* e os tópicos arquiteturais associados. Descobriu-se que algumas palavras-chave selecionaram apenas algumas mensagens de *commit*, mas com uma taxa de aceitação superior à média verificada (5,9% de mensagens filtradas por

mensagens confirmadas). A Tabela 3.9 apresenta a relação dessas palavras-chave. Por outro lado, um conjunto de palavras-chave conseguiu indicar uma quantidade maior de mensagens de *commit*. Tendo em vista que esse estudo buscou o máximo de informações arquiteturais disponíveis no projeto, é importante destacar quais palavras-chave foram responsáveis por detectar mais mensagens de *commit* com conteúdo arquitetural. Elas são exibidas na Tabela 3.10. Vale ressaltar que as palavras-chave destacadas na Tabela 3.10 apresentaram as maiores porcentagens de mensagens de *commit* arquiteturais confirmadas (exceto a palavra-chave "`component`" e suas variações) ao considerar as mensagens de *commit* manualmente confirmadas como arquiteturais em comparação com aquelas não-confirmadas.

Table 3.8: Tempo médio entre ocorrências (em meses) de cada tópico arquitetural (ATOAC), porcentagem de *commits* ocorridos no período de lançamento de novas *releases* do projeto (CCLNV) para cada tópico arquitetural e total de ocorrências de *commits* encontrados para cada tópico arquitetural.

Tópico Arquitetural	**ATOAC**	**CCLNV (%)**	**Total**
Elementos Arquiteturais de Processamento	16	33.3	6
Elementos Arquiteturais de Dados	-	100	1
Elementos Arquiteturais de Conexão	15	38.5	13
Motivações, Objetivos e Restrições	9	42.9	7
Estrutura Global de Controle	44	33.3	6
Estruturas do Sistema	4	52.2	23
Protocolos de Comunicação	4	31.6	19
Mecanismos de Sincronização	2	39.5	43
Mecanismos de Acesso a Dados	22	100	11
Atribuição de Recursos	15	25.0	12
Relação entre Elementos	1	53.3	60
Organização Fundamental	-	33.3	3
Requisitos Não-funcionais	6	30.0	20
Estilos Arquiteturais	-	100	1
Outros elementos relacionados à Arquitetura	5	57.1	7

Foram identificados três cenários no relacionamento entre palavras-chave e tópicos arquiteturais identificados em mensagens de *commit*. No primeiro, observa-se que a maioria das palavras-chave não possui uma identificação direta com nenhum tópico arquitetural. Para essa análise, consideram-se apenas as palavras-chave com pelo menos quatro mensagens de *commit* relacionadas. Como exemplos de palavras-chave não relacionadas a nenhum tópico, tem-se: *"adaptability"*, *"layering"* e *"extends"* (5 ocorrências espalhadas em 4 tópicos diferentes), *"constraints"* (4 ocorrências espalhadas em 3 tópicos diferentes), *"extensibility"* (8 ocorrências em 6 tópicos diferentes), *"iterator"* (9 ocorrências em 6 tópicos diferentes), *"provides"* (com 41 ocorrências espalhadas em 11 tópicos diferentes) e *"interface"* (31 ocorrências em 9 tópicos diferentes). No segundo cenário, algumas palavras-chave selecionaram uma alta porcentagem de mensagens de *commit* relacionadas a um tópico específico. Por exemplo, a palavra-chave *"modifiability"* (com 13 ocorrências e 54% das ocorrências concentradas no tópico "Mecanismos de Sincronização") *"requires"* (com 45 ocorrências e 54% delas concentradas nos tópicos "Relação entre Elementos" e "Mecanismos de sincronização") e *"module"* (33 ocorrências e 63% das ocorrências

concentradas nos tópicos “Relação entre Elementos” e “Estruturas de sistema”).

Table 3.9: Palavras-chave com maior taxa de aceitação perante os *commits* filtrados, Número de *commits* filtrados (NCF), Percentual de *commits* confirmados por palavra-chave (PCCK).

Palavra-chave	**NCF**	**PCCK(%)**
architecture	4	44.44%
component	28	14.36%
modularity	33	12.50%
reliability	4	10.00%
requires	45	9.57%

Table 3.10: Palavras-chave que mais detectaram *commits* arquiteturais, Número de *commits* filtrados (NCF), Número de mensagens de *commit* confirmados por palavra-chave(NCC) e Percentual de *commits* confirmados por palavra-chave (PCCK).

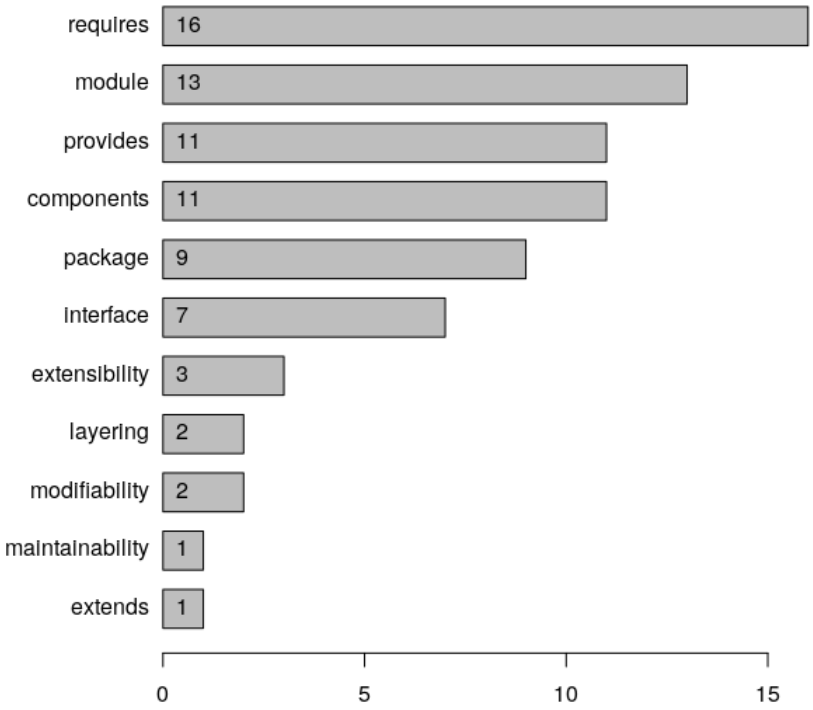
Palavra-chave	**NCF**	**NCC**	**PCCK(%)**
requires	470	45	9.57%
provides	534	41	7.68%
interface	467	35	7.49%
module	442	33	7.47%
components	195	28	14.36%

Um caso digno de destaque foi a palavra-chave “`package`”, encontrada em 17 mensagens de *commit*. Ela está relacionada a 10 tópicos diferentes, com 54% para os tópicos “Relação entre Elementos” ou “Estruturas do Sistema”.

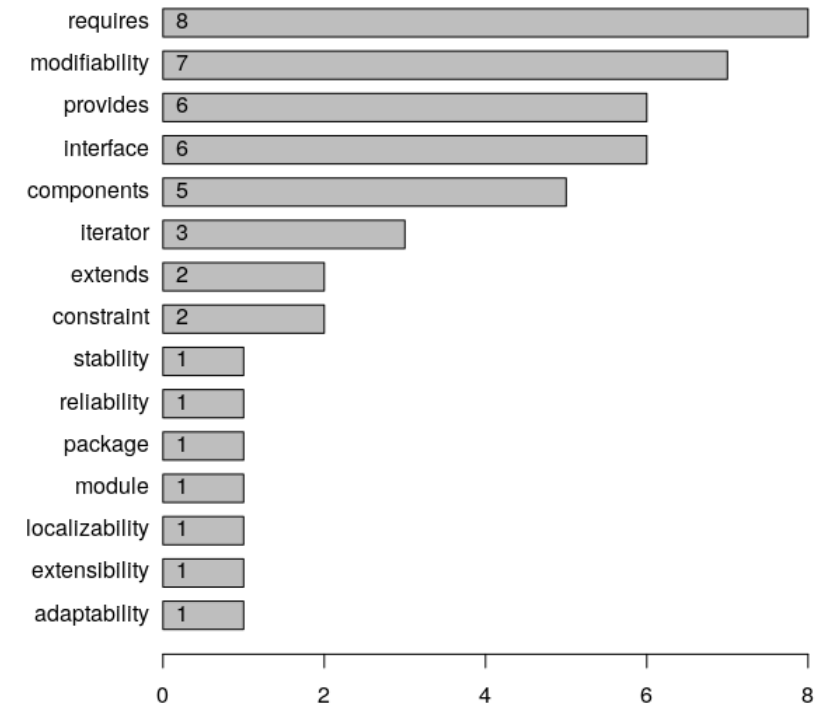
Utilizou-se o teste U de Mann-Whitney-Wilcoxon para analisar a distribuição e diferenciar as amostras. Para esse conjunto, obteve-se um *p-value* de $2{,}764e^{-5}$. As Figuras 3.7, 3.8 e 3.9 apresentam os gráficos de barras contendo as palavras-chave associadas aos cinco tópicos arquiteturais mais encontradas no projeto KDELibs.

Apesar de um conjunto amplo e que encontrou pelo menos uma ocorrência para cada tópico arquitetural definido na taxonomia utilizada, não se pode descartar que as palavras-chave do conjunto sejam suficientes para apontar todas as mensagens de *commit* com informações arquiteturais presentes no projeto.

A distribuição das mensagens de *commit* denotou um aspecto esperado das mudanças na arquitetura. Para um projeto de longo prazo, como o KDELibs, espera-se que os “Elementos Arquiteturais de Dados” e o “Estilo Arquitetural” do projeto mudem pouco. Vale ressaltar que, quando a equipe de colaboradores do projeto decidiu alterar seu estilo arquitetural (de monolítico para em camadas), o projeto KDELibs foi interrompido. Para substituir essa biblioteca, a equipe de colaboradores do KDE criou o framework KF5.

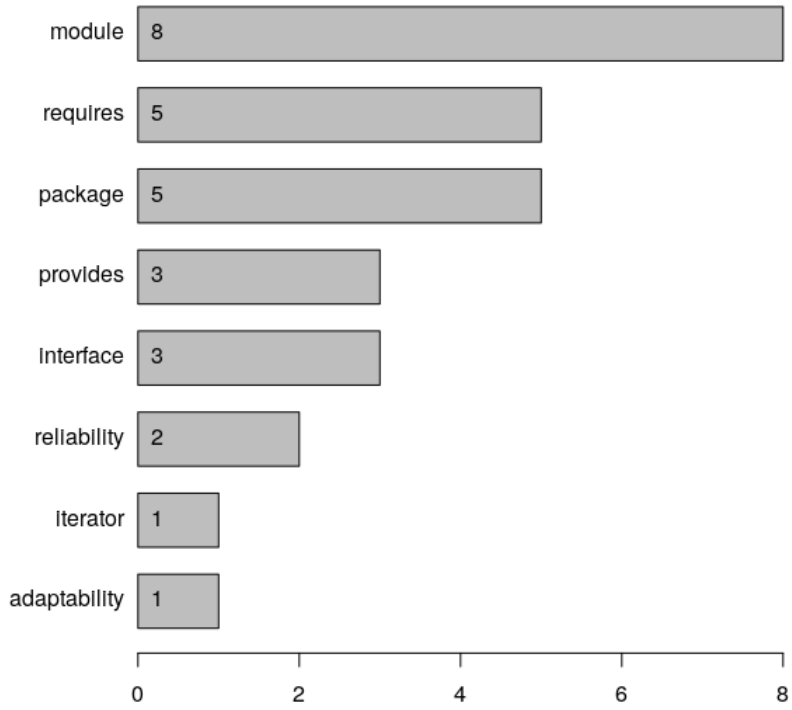


(a) Palavras-chave que apontaram *commits* arquiteturais - Tópico: **Relação entre Elementos**

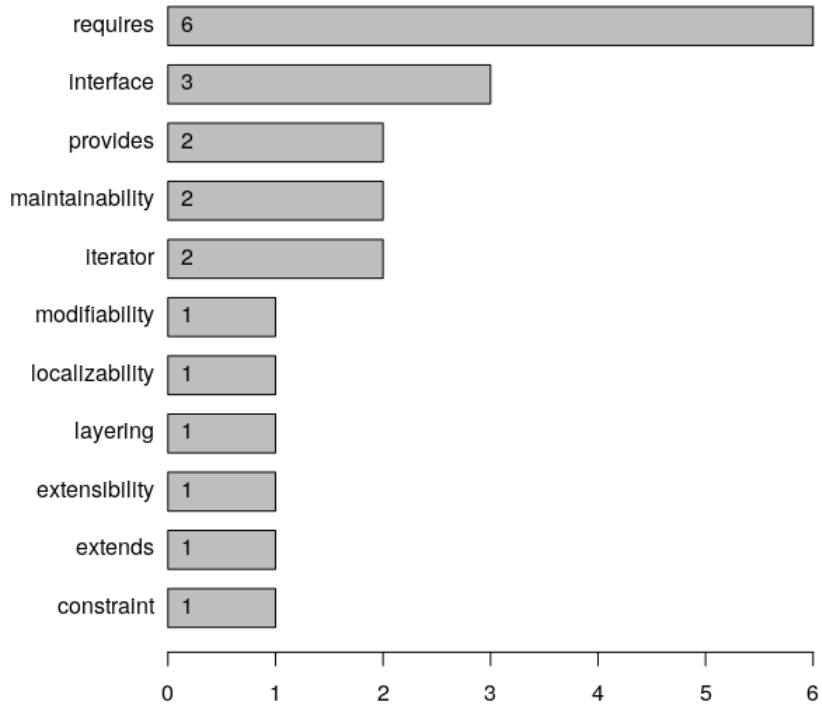


(b) Palavras-chave que apontaram *commits* arquiteturais - Tópico: **Mecanismos de Sincronização**

Figure 3.7: Palavras-chave que apontaram *commits* arquiteturais com os cinco tópicos arquiteturais mais identificados no projeto KDELibs (parte 1).

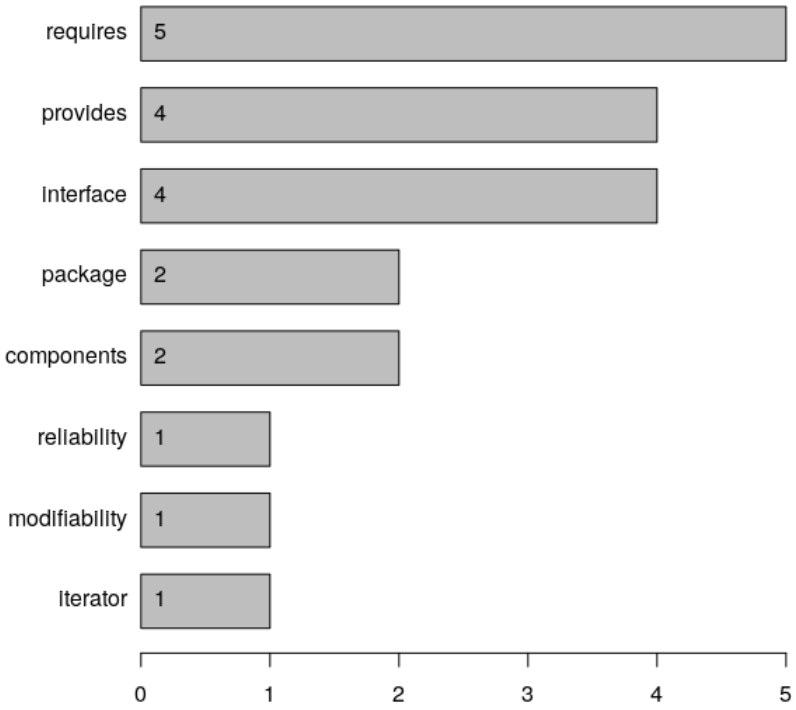


(a) Palavras-chave que apontaram *commits* arquiteturais - Tópico: **Estruturas do Sistema**



(b) Palavras-chave que apontaram *commits* arquiteturais - Tópico: **Requisitos não-funcionais**

Figure 3.8: Palavras-chave que apontaram *commits* arquiteturais com os cinco tópicos arquiteturais mais identificados no projeto KDELibs (parte 2).



(a) Palavras-chave que apontaram *commits* arquiteturais - Tópico: **Protocolos de Comunicação**

Figure 3.9: Palavras-chave que apontaram *commits* arquiteturais com os cinco tópicos arquiteturais mais identificados no projeto KDELibs (parte 3).

Em relação ao tópico arquitetural "Relação entre Elementos", esperava-se que fosse o tópico com o número mais significativo de registros, porque se trata de um projeto com muitas interações com outros projetos, porque possui um grande número de componentes e porque é um projeto que apresentou crescimento (no número de funcionalidades). Por outro lado, a quantidade notável de registros de alterações nos "Mecanismos de Sincronização" é surpreendente, a princípio, pois não é comum as alterações em um tópico arquitetural que afeta a sincronização de objetos transitórios no projeto. No entanto, como é uma biblioteca com uso diversificado e abrangente (de um ambiente gráfico às interações com dispositivos de E/S), esse volume de modificações nesse tópico justifica-se.

Como se obteve um pequeno número de mensagens de *commit* de informações arquiteturais, eram esperadas dificuldades no estabelecimento de relações entre os tópicos durante a evolução desse projeto, considerando o mês de registro. No entanto, dada a prevalência entre dois tópicos arquiteturais ("Relação entre Elementos" e "Mecanismos de Sincronização") em relação aos demais (44% do total formado por esses dois tópicos), é natural que os relacionamentos ocorram entre eles e os mais tópicos. No entanto, era esperado que os seguintes relacionamentos fossem estabelecidos entre os tópicos, pois é esperado que as mudanças no primeiro teriam impacto no segundo tópico:

- (i) “Organização Fundamental” e o conjunto que consiste em “Elementos Arquiteturais de Dados”, “Elementos Arquiteturais de Conexões”,
- (ii) “Motivações, Objetivos e Restrições” e “Estilos Arquiteturais”,
- (iii) “Estruturas Globais de Controle” e “Motivações, Objetivos e Restrições”,
- (iv) “Relação entre Elementos” e “Mecanismos de Acesso a Dados”,
- (v) “Mecanismos de Sincronização” e “Protocolos de Comunicação”,
- (vi) “Relação entre Elementos” e “Atribuições de Recursos” e,
- (vii) “Relação entre Elementos” e “Estrutura Global de Controle”.

Destes, apenas os dois últimos (vi e vii) foram confirmados por este estudo.

A concentração dos registros de tópico “Mecanismos de Acesso a Dados”, durante o período de lançamento de novas *releases*, pode ser justificada como ajustes pós-lançamento, ao verificar-se uma política inadequada de acesso aos dados por parte dos projetos associados. A Tabela 3.7 mostra que 7 dos 11 registros estão no período de lançamento de *patches*. Em relação ao tópico “Requisitos Não-Funcionais”, observa-se que o problema foi tratado no intervalo do release, pois são modificações que alteram vários aspectos do projeto e exigem estabilidade no momento do novo release. Encontrou-se uma periodicidade semelhante no tópico “Mecanismos de Sincronização”, mas a maioria de seus registros estava concentrada nas *releases* de *patches* (3.5.x). Isso se deve ao impacto desse tipo de modificação em projetos satélites, especialmente, aqueles que interagem com dispositivos de E/S e trabalham com objetos transientes.

Assim, a resposta para a **RQ1.3** é:

Os tópicos mais citados nas mensagens de *commit* arquitetural do projeto KDELibs são “Mecanismos de Sincronização” e “Relação entre Elementos”; sendo o segundo tópico o mais comum. Por outro lado, os tópicos “Organização Fundamental”, “Estilos arquiteturais” e “Elementos Arquiteturais de Dados” raramente aparecem.
Os tópicos “Relação entre elementos” e “Mecanismos de acesso a dados” são frequentemente citados em alterações que ocorrem no período de lançamento de novas *releases*.
O tópico “Relação entre Elementos” geralmente precede os tópicos “Estruturas Globais de Controle”, “Atribuições de Recursos” e “Outros Elementos Relacionados à Arquitetura”.
Por fim, a maioria das palavras-chave não está diretamente relacionada a um tópico arquitetural específico.

A presença de informações arquiteturais nas mensagens de *commit* provoca a pergunta sobre a existência de um perfil de desenvolvedor que colabore mais ativamente com a modificação da arquitetura do projeto. Caso exista esse perfil, a identificação de desenvolvedores que nele se encaixe, poderá ajudar a reduzir falsos positivos ao tentar detectar automaticamente modificações e informações arquiteturais.

3.4.4 Qual é o perfil dos colaboradores que realizam alterações na arquitetura?

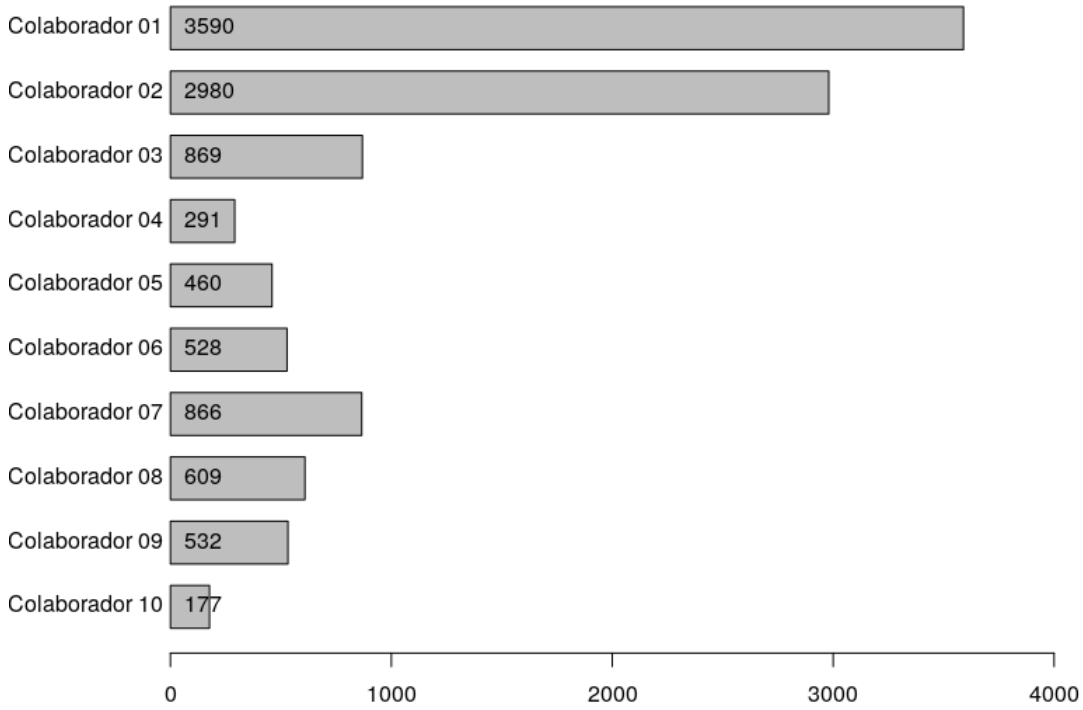
Quer-se conhecer o perfil dos principais colaboradores das mudanças arquiteturais. Dessa forma, planejou-se a **RQ1.4** para responder a essa pergunta. Conforme descrito na seção de design deste estudo, foram analisados um período de pouco mais de dez anos do projeto KDELibs. A resposta à **RQ1.4** está relacionada à analise da Figura 3.10, da Tabela 3.11 e da Tabela 3.12. A Figura 3.10 apresenta dois gráficos de barras: (a) o primeiro mostra o total de *commits* por desenvolvedor, (b) o segundo mostra a quantidade de *commits* arquiteturais por desenvolvedor. Ambos consideram os dez colaboradores com mais alterações arquiteturais registradas no projeto. A Tabela 3.11 lista os dez colaboradores que contribuíram com o maior número de *commits* arquiteturais.

A Tabela 3.12 complementa a Tabela 3.11. Ela mostra doze dos vinte e dois colaboradores com mais *commits* registrados no projeto KDELibs, excluindo os colaboradores listados na Tabela 3.12. A junção das tabelas 3.11 e 3.12 lista os vinte e dois principais colaboradores que contribuíram com o maior número de *commits* em geral (todos os tipos de *commit*). Ambas as tabelas mostram em relação a cada colaborador: (i) o número de *commits* em geral, (ii) o número de *commits* arquiteturais, (iii) o tempo de participação no projeto, (iv) a média de *commits* gerais por mês e a média de *commits* arquiteturais por mês. A Tabela 3.12 não apresenta a coluna contendo o tempo decorrido entre o primeiro e o último *commit* arquitetural.

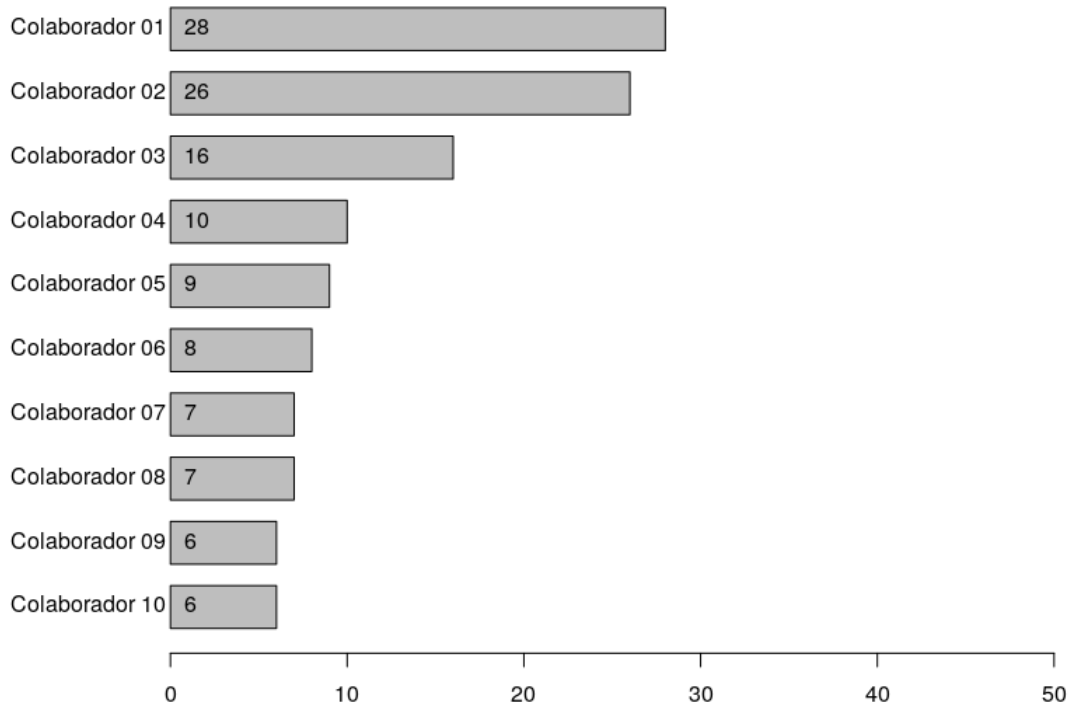
Table 3.11: Participação dos dez maiores colaboradores de *commits* com mudanças na arquitetura. Total de *commits* (TC), Quantidade de *commits* Arquiteturais (AC), Tempo de participação no projeto (TP) – em meses –, Tempo entre o primeiro e o último *commit* arquitetural (TFL) – em meses –, Média de *commits* registrados por mês (GCM) e Tempo médio para o registro de *commits* com modificações na arquitetura (AFT) — em meses/*commits*.

Autor	TC	AC	TP	TFL	GCM	AFT
Colaborador 1	3590	28 (0,8%)	122	110	29	3,9
Colaborador 2	2980	26 (0,9%)	89	84	32	3,2
Colaborador 3	869	16 (1,8%)	87	35	10	2,2
Colaborador 4	291	10 (3,4%)	59	39	5	3,9
Colaborador 5	460	9 (2%)	88	41	5	4,6
Colaborador 6	528	8 (1,5%)	50	19	4	2,4
Colaborador 7	866	7 (0,8%)	29	10	27	1,4
Colaborador 8	609	7 (1,1%)	91	48	6	6,9
Colaborador 9	532	6 (1,1%)	110	18	19	3,0
Colaborador 10	177	6 (3,4%)	84	40	15	6,7

A Tabela 3.11 mostra que, além dos dois principais colaboradores responsáveis por modificações na arquitetura, todos os outros colaboradores não fazem modificações na arquitetura com frequência. A coluna TFL mostra o tempo decorrido entre a primeira e a última modificação arquitetural registrada. Observando o colaborador "Colaborador 9", por exemplo, percebe-se que ele contribuiu para o projeto quase na mesma quantidade de meses que o maior colaborador ("Colaborador 1") – 110 x 122 meses –, no entanto, suas modificações arquiteturais se concentraram em apenas 18 meses do projeto. Uma



(a) Total de *commits* registrados pelos maiores colaboradores de modificações arquiteturais



(b) Quantidade de *commits* arquiteturais pelos maiores colaboradores de modificações arquiteturais

Figure 3.10: Total de *commits* registrados (a) Quantidade de *commits* registrados com modificação arquitetural (b) para os maiores colaboradores de modificações arquiteturais no projeto KDELibs.

situação parecida é observada quanto ao "Colaborador 7", que contribuiu por 29 meses, mas concentrou suas modificações arquiteturais em apenas 10 deles. Quando observado o tempo médio entre as modificações arquiteturais promovidas pelos principais colaboradores de modificações arquiteturais, vêem-se que novas modificações arquiteturais são feitas a cada 2 a 5 meses em média. Exceções a essa regra são os colaboradores "Colaborador 7"(que faz suas modificações arquiteturais a cada 1,4 meses em média), "Colaborador 8" e "Colaborador 10" (que fazem suas contribuições a cada 6,9 e 6,7 meses, respectivamente).

Observa-se, inclusive, que, excetuando-se os dois maiores colaboradores, os dez principais colaboradores que descrevem modificações arquiteturais, em suas mensagens de *commit*, concentram suas contribuições em períodos específicos do projeto.

Percebe-se a partir dos dados, que um pequeno número de colaboradores tem uma visão geral do sistema e modifica a arquitetura do projeto. Esses colaboradores podem ajudar a esclarecer os principais elementos, estilos e padrões arquiteturais usados e a melhorar a comunicação no projeto.

Table 3.12: Participação dos vinte e dois maiores colaboradores do projeto KDELibs (exceto os maiores colaboradores de modificações arquiteturais) com: Total de *commits* (TC), Quantidade de *commits* Arquiteturais (AC), Tempo de participação no projeto (TP) – em meses –, Tempo médio para o registro de *commits* com modificações na arquitetura (AFT) — em meses/*commits* —, Média de *commits* registrados por mês (GCM).

Autor	TC	AC	TP	AFT	GCM
Colaborador 11	1438	0	83	NA[1]	17,3
Colaborador 12	1375	3	85	28	16,2
Colaborador 13	1183	3	67	22	17,7
Colaborador 14	1057	4	56	14	18,9
Colaborador 15	824	3	83	27	10,3
Colaborador 16	813	1	72	72	11,3
Colaborador 17	791	3	66	22	12,0
Colaborador 18	684	3	54	18	12,7
Colaborador 19	639	3	45	15	14,2
Colaborador 20	628	0	54	NA[1]	11,6
Colaborador 21	609	0	130	NA[1]	4,7
Colaborador 22	546	3	86	29	6,4

[1]Não se aplica

Em resumo, perante a **RQ1.4**, pode-se afirmar:

Os principais colaboradores no projeto KDELibs também são os principais colaboradores de mudanças na arquitetura do projeto e fazem as suas contribuições de maneira homogênea.
Os dois principais colaboradores do projeto também são os principais colaboradores de alterações arquiteturais.
Os outros principais colaboradores de alterações arquiteturais posicionam-se entre os vinte principais colaboradores do projeto, exceto o Colaborador 4 e o Colaborador 10, que iniciaram suas contribuições mais recentemente.

Traçado o perfil dos colaboradores que efetuam modificações arquiteturais e as descrevem nas mensagens de *commit*, buscou-se diferenciá-las das mensagens de *commit* com outros tipos de mudanças. Especula-se que o tamanho, em caracteres, dessas mensagens pode ser um indicador de que as carregam informações sobre mudanças arquiteturais.

3.4.5 As mensagens de commit arquiteturais são mais longas do que outros tipos de mensagens de commit?

A pergunta de pesquisa **RQ1.5** propõe uma comparação entre o tamanho das mensagens de *commit* arquiteturais e o tamanho das mensagens de *commit* de propósito geral (sem alterações arquiteturais). Observa-se que as mensagens de *commit* arquiteturais usam uma média de 2,4 vezes mais caracteres do que outras mensagens de *commit* (345,4 *versus* 143,9 nos *commits* de uso geral). Aplicou-se o teste U de Mann-Whitney-Wilcoxon e obteve-se um *p-value* de $2{,}2e^{-16}$. O efeito observado foi grande (ΔCliff = 0,8918), a amplitude registrada foi de 3.476 caracteres para os *commits* arquiteturais e de 707 para os *commits* com outros tipos de mudanças. A Figura 3.11 (parte a) mostra a distribuição do tamanho das mensagens de *commit* em *boxplots*, comparando mensagens de *commit* arquiteturais e não-arquiteturais.

Testou-se a hipótese de que a razão entre o número de caracteres das mensagens de *commit* arquiteturais e as mensagens de *commit* não-arquiteturais muda a partir da exclusão de palavras irrelevantes. Ao removê-las e comparar os dois tipos de mensagens, constatou-se que a diferença de tamanho entre as mensagens aumenta. Apesar de obter o mesmo *p-value* ($2{,}2e^{-16}$) e um grande efeito no índice Cliff (ΔCliff = 0,9077), as mensagens de *commit* arquiteturais (sem palavras irrelevantes) usam em média de 3 vezes mais caracteres do que mensagens de *commit* não-arquiteturais (225,95 *versus* 75,28 nos *commits* de uso geral). A parte b da Figura 3.11 ilustra a comparação entre a distribuição de tamanho de mensagens com conteúdo arquitetural e aquelas sem conteúdo arquitetural (descartando palavras irrelevantes).

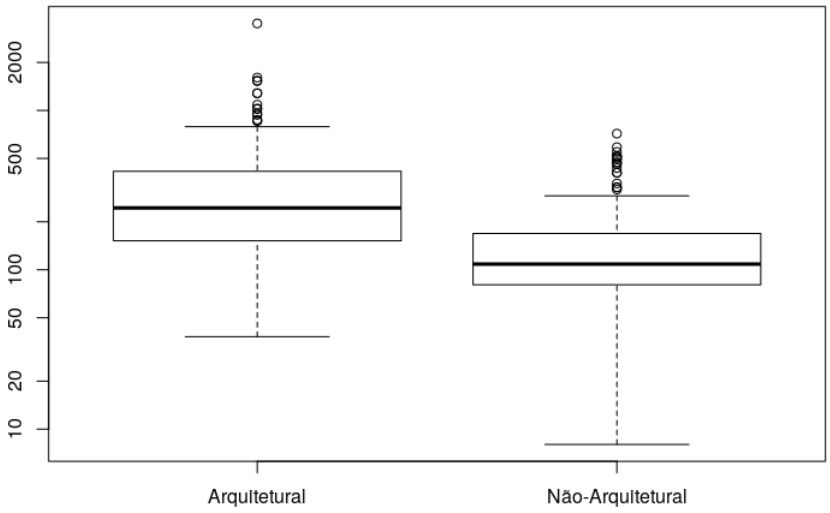
Em relação ao tamanho das mensagens, viu-se que as mensagens de *commit* arquiteturais são pelo menos duas vezes maiores (em número de caracteres) que as mensagens de *commit* de uso geral, em média.

Portanto, a resposta da **RQ1.5** é:

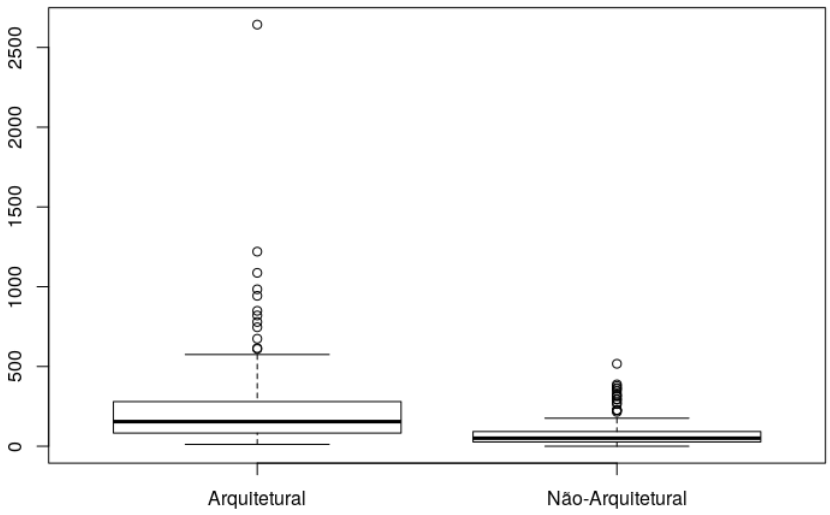
> As mensagens de *commit* que contêm informações arquiteturais tendem a ter mais caracteres do que mensagens de *commit* que versam sobre outros tipos de mudanças.

Os dados coletados confirmam a hipótese de que as mensagens com conteúdo arquitetural são mais extensas (em quantidade de caracteres) do que as mensagens de *commit* com os demais tipos de mudanças. Os exemplos de cada tópico arquitetural na Seção de design (3.3) deste estudo servem para ilustrar essa afirmação. Essa descrição alongada ocorre para deixar evidente para outros colaboradores o que pode ser afetado pelo projeto como resultado de alterações feitas.

A **RQ1.5** mostra que as mensagens de *commit*, com informações arquiteturais contêm, em média, mais caracteres que as mensagens de outros tipos de *commit*. Essa descoberta desperta a curiosidade sobre a tendência em se modificar mais arquivos em *commits*



(a) Tamanho das Mensagens Completas



(b) Tamanho das Mensagens sem palavras irrelevantes

Figure 3.11: Tamanho das mensagens de *commit* – Comprimento das mensagens (em caracteres). Mensagens completas (a) x Mensagens sem palavras irrelevantes (b).

arquiteturais que outros *commits*. Caso haja essa diferença, tal tipo de informação pode ajudar na automatização do processo de identificação de *commits* arquiteturais.

3.4.6 Commits arquiteturais alteram mais arquivos do que commits não-arquiteturais?

A subquestão de pesquisa **RQ1.6** propõe uma comparação entre o tamanho dos *commits* arquiteturais e o tamanho dos *commits* não-arquiteturais em termos do número de arquivos alterados. A Figura 3.12 mostra a distribuição do número de arquivos alterados (afetados) por *commits* arquiteturais e não-arquiteturais. Descobriu-se que os *commits* arquiteturais alteraram em média de 24% mais arquivos que *commits* não-arquiteturais. O teste U de Mann-Whitney-Wilcoxon indicou um *p-value* de $4{,}613e^{-7}$, embora o tamanho do efeito fosse pequeno ($\Delta\text{Cliff} = 0{,}0453$). A amplitude registrada foi de 210 arquivos para *commits* arquiteturais e 409 para *commits* não-arquiteturais.

Os dados coletados confirmam a hipótese de que *commits* com mensagens contendo informações arquiteturais modificam mais arquivos no projeto que outros tipos de *commits*. No entanto, acreditava-se que a diferença seria maior que a encontrada (24%), pois essas modificações devem ser transversais entre os componentes e, assim, partes distintas do projeto seriam alcançadas. Um exemplo disso seria uma alteração entre os mecanismos de acesso a dados. Uma vez modificados, espera-se que todos os componentes que contêm campos devam alterá-los ou a seus mecanismos de acesso.

Assim, a resposta da **RQ1.6** é:

> *Commits* com mensagens contendo informações arquiteturais tendem a modificar mais arquivos do que *commits* com outros tipos de mudanças.

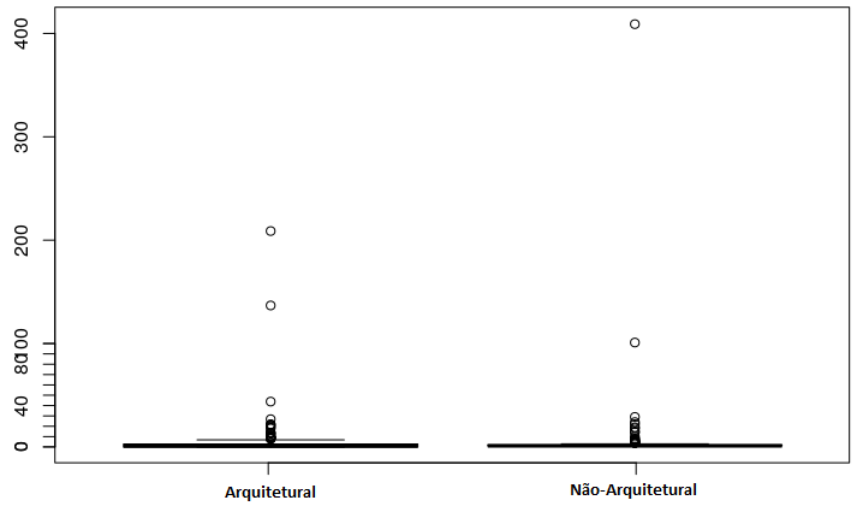


Figure 3.12: Abrangência de *commits* – Quantidade de arquivos/módulos afetados pelas mudanças. Comparação entre *commits* arquiteturais e *commits* não-arquiteturais.

A **RQ1.6** mostra que *commits* com mensagens contendo informações arquiteturais modificam, em média, mais arquivos do que outros tipos de *commits*. Essa descoberta

levanta o questionamento de que *commits* com modificação na arquitetura tendem a modificar repetidamente o mesmo arquivo/módulo. Se houver essa diferença, esse tipo de informação pode ajudar a separar os *commits* arquiteturais dos demais.

3.4.7 Commits arquiteturais alteram o mesmo conjunto de arquivos?

A pergunta **RQ1.7** questiona se *commits* arquiteturais alteram um conjunto semelhante de arquivos, ou seja, se existem conjuntos de arquivos que são modificados em pelo menos dois *commits* arquiteturais diferentes. Descobriu-se que o número de arquivos que são alterados por diferentes *commits* arquiteturais é apenas 12% superior ao número de arquivos alterados por diferentes *commits* não-arquiteturais.

A Figura 3.13 mostra a distribuição do número de arquivos alterados frequentemente por *commits* arquiteturais e não-arquiteturais. Assim como na **RQ1.6**, imaginava-se uma diferença maior. Aplicou-se o teste U de Mann-Whitney-Wilcoxon e obteve-se um *p-value* de $6{,}305e^{-8}$, com um tamanho de efeito pequeno ($\Delta\text{Cliff} = 0{,}205$). A Tabela 3.13 mostra a comparação dos dados de repetição de arquivos para *commits* arquiteturais e não-arquiteturais. Também observa-se que os arquivos com o maior índice de repetição foram: `kioslave/http/http.cpp` (8 repetições) e `plasma/containsment.cpp` (5 repetições) para os *commits* arquiteturais e não-arquiteturais, respectivamente, e que a maioria dos arquivos foi não foi alterado mais de uma vez (1.057 X 151 para *commits* arquiteturais e arquivos 998 x 67 para não-arquiteturais).

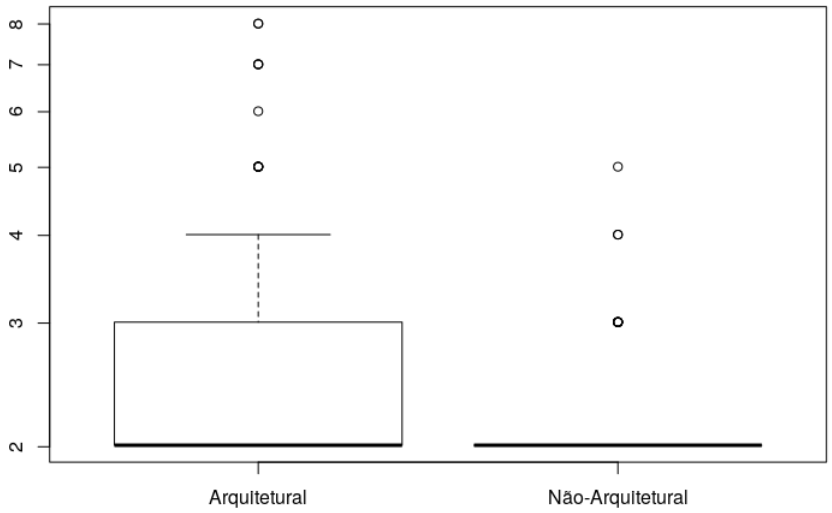


Figure 3.13: Quantidade de arquivos que sofrem modificações em diferentes *commits* (Módulos modificados constantemente): Comparação entre *commits* arquiteturais e não-arquiteturais.

Table 3.13: Quantidade de arquivos modificados repetidamente em diferentes *commits*.

	***Commits* Arquiteturais**	***Commits* em Geral**
Arquivos não-repetidos	1057	998
Arquivos modificados entre 1 e 5 diferentes *commits*	139	66
Arquivos modificados em mais que 5 *commits*	12	1

Assim, a resposta para a **RQ1.7** é:

Commits com mensagens contendo informações arquiteturais tendem a modificar os mesmos arquivos em maior frequência do que outros tipos de *commits*.

Os resultados confirmam a hipótese de que os *commits* com mensagens de *commit* que contêm informações arquiteturais tendem a modificar os mesmos arquivos com mais frequência do que outros tipos de *commits*. No entanto, esperava-se encontrar uma diferença maior entre os dois grupos, pois existem componentes que se concentram no núcleo de alguns tópicos arquiteturais. Portanto, uma vez que o tópico arquitetural seja modificado, espera-se que esses componentes sejam alterados no mesmo release. Um exemplo seriam os protocolos de comunicação. Em todas as modificações em um determinado protocolo, os componentes relacionados também seriam modificados por consequência.

3.5 IMPLICAÇÕES

Os resultados desta pesquisa podem servir para orientar futuros colaboradores em comunidades de Software Livre, pesquisadores de Engenharia de Software e educadores em Ciência da Computação.

3.5.1 Pesquisa

A busca por informações sobre a arquitetura de projetos de software expandiu as fontes de informação. Os resultados deste estudo apontam para aspectos inexplorados nesse tipo de pesquisa em fontes de dados não-estruturadas.

As informações arquiteturais podem ser encontradas em mensagens de *commit*, acompanhando a evolução dos projetos e, assim, ajudando a entender a evolução de sua arquitetura. Ficou comprovado que a pesquisa de palavras-chave é útil para encontrar esse tipo de informação e que pode ser aprimorada com o conhecimento sobre a evolução do projeto.

É possível considerar que há desenvolvedores que discutem mais esses aspectos (mas as informações não se limitam a eles). Também, pode-se citar que há períodos que concentram mais discussões sobre arquitetura, como o período de lançamento de novas *releases*. No entanto, a maioria das informações é distribuída ao longo da evolução do projeto. É cabível considerar que não há restrição nos tópicos arquiteturais discutidos. Todos os tópicos definidos na taxonomia utilizada foram encontrados. Devido à pequena quantidade de informações encontradas sobre aspectos estruturais da arquitetura, esse

pode não ser o método mais eficiente para recuperá-los.

Por outro lado, outras informações arquiteturais são acessíveis por esse método. É possível considerar que os *commits*, com mensagens referentes às alterações arquiteturais, têm uma descrição textual mais detalhada, alcançam mais arquivos e, geralmente, alteram os mesmos arquivos. Assim, juntos, esses aspectos sugerem novas pesquisas em outros projetos, com outros domínios, para que se possa aprofundar a busca de informações sobre mudanças na arquitetura ao longo da evolução de projetos.

3.5.2 Prática

Geralmente, os colaboradores, que desejam contribuir para um projeto de Software Livre, têm dificuldade de entender a construção do código, as opções e os padrões de design e de arquitetura utilizados. Assim, fornecer *insights*, que ajudem a sua compreensão do projeto, pode criar identidade dos colaboradores junto ao projeto e melhorar a sua produtividade. Os colaboradores, que entendem as diretrizes do projeto, tendem a oferecer contribuições melhores e por mais tempo (PANICHELLA et al., 2014).

Assim, os resultados desta pesquisa representam uma contribuição relevante para os novos colaboradores, pois ajudam a identificar como o projeto evoluiu do ponto de vista arquitetural. Esses colaboradores podem pesquisar mensagens de *commit* à medida que os tópicos específicos foram tratados (diretamente, pois foram encontradas menções nessas descrições) usando palavras-chave e mais informações sobre esses *commits*.

Essas informações incluem:

- As mensagens com conteúdo arquitetural contêm, em média, mais caracteres que mensagens de outros tipos de *commits*;
- Existe um conjunto de colaboradores discutindo tópicos arquiteturais com mais frequência;
- Existe um conjunto inicial de palavras-chave que ajuda a identificar mensagens com informações arquiteturais;
- *Commits* com informação arquitetural modificam mais arquivos do que outros e os componentes afetados por essas mudanças são modificados repetidamente;
- Uma parte significativa (mas não a maioria) dessas modificações ocorre durante o período de lançamento de novas *releases* do projeto, e;
- Há uma ampla discussão sobre informações arquiteturais (foram encontrados todos os tópicos da taxonomia aplicada no projeto estudado).

3.5.3 Educação

Os resultados deste trabalho para educadores de graduação e pós-graduação, que ministram cursos na área de Ciência da Computação e Estatística, podem ser úteis.

Os estudantes de Ciência da Computação podem buscar informações sobre o projeto em artefatos para além da documentação específica, a exemplo das mensagens de *commit* aqui estudadas.

Os engenheiros de software recém-treinados poderão procurar novas fontes de informações para obter dados que os ajudarão a entender o software legado e os projetos mal documentados ou com documentação desatualizada da arquitetura de software.

O trabalho de Brunet et al. (2014) já havia encontrado informações de design a partir dessa fonte de informações e observou-se que há mais informações disponíveis do que informações de design de software, a exemplo de informações sobre alterações arquiteturais.

Os estudantes de estatística poderão observar a aplicação de métodos para verificar a normalidade e a significância estatística em estudos de engenharia de software e recuperação de informações.

3.6 AMEAÇAS À VALIDADE

A seguir são apresentadas as ameaças à validade identificadas neste estudo.

Validade de Construto. Para minimizar o viés do método único, usou-se um teste de precisão e concordância para fornecer uma indicação das palavras-chave que formaram o conjunto utilizado na detecção de mensagens de *commit* com informações arquiteturais. Foram selecionados especialistas em arquitetura de software de vários locais, que também participam de projetos da indústria, para escolherem o conjunto de palavras-chave. Finalmente, para reduzir as ameaças sociais, devido à apreensão da avaliação, os participantes não foram avaliados.

Validade Externa. A primeira ameaça é o tipo de projeto de software. Adotou-se um projeto complexo. Outro ponto é o domínio: é um software central para o ecossistema do KDE. Um software de um domínio diferente pode apresentar menos preocupações com a arquitetura de software. No entanto, é possível argumentar que o fenômeno pode ser estudado em qualquer tipo de software. Outra ameaça à validade externa foi o uso de software que difere da prática industrial. Para mitigar essa ameaça, foi utilizado o projeto KDELibs, um software que tem uma longa evolução e interage com vários outros projetos. Finalmente, por usar apenas um projeto, fez-se necessário reduzir a generalização dos resultados. No entanto, é preciso enfatizar que este é um estudo exploratório. O número de especialistas, que participaram da escolha das palavras-chave utilizadas no estudo (cinco), não garante a generalização do conjunto produzido. Para minimizar esse problema, foi escolhido outro público - estudantes de pós-graduação, que trabalham com arquitetura de software, e aplicou-se um teste de similaridade para verificar a semelhança entre os conjuntos produzidos.

Validade Interna. A primeira ameaça interna que se deve considerar é a seleção de especialistas, porque eles foram escolhidos por meio de amostragem de conveniência. No entanto, outras ameaças foram identificadas. Um exemplo é a construção de um conjunto de palavras-chave produzido para ajudar a detectar informações arquiteturais em mensagens de *commit*. Ele foi construído com base na opinião de um pequeno número de pesquisadores especialistas em arquitetura de software. No entanto, nenhum

desses pesquisadores é desenvolvedor do projeto KDELibs. Por esse motivo, acredita-se que algumas palavras-chave relacionadas à arquitetura, que são específicas ao contexto dos colaboradores do KDELibs, podem estar ausentes do conjunto de palavras-chave produzido. Minimizou-se essa ameaça (poucos especialistas e nenhum desenvolvedor do KDELibs), convidando especialistas de diferentes locais e do campo da arquitetura de software. Também deve ser considerado o pequeno número de palavras-chaves selecionado pelos especialistas como sendo relacionado à arquitetura. No entanto, o número reduzido de palavras-chaves foi compensado com a escolha aleatória de mensagens não filtradas pela ferramenta ATM. Nenhuma dessas mensagens possuía conteúdo arquitetural, atestando, assim, a qualidade do conjunto produzido.

Outra ameaça é que os participantes poderiam ser afetados negativamente pelo tédio e pelo cansaço. Ofereceu-se um formulário simples, de preenchimento rápido, onde os participantes precisavam destacar as palavras-chave que acreditavam estar relacionadas à arquitetura.

O fato de os especialistas não serem participantes do projeto KDELibs pode contribuir para diminuir a detecção de mensagens de *commit*, uma vez que eles criaram um conjunto de palavras-chave que não são específicas para o KDELibs e podem ter tendência para o vocabulário acadêmico. Para atenuar essa ameaça, foram convidados estudantes de pós-graduação, com oito anos de experiência em desenvolvimento, em média, incluindo um estudante que contribui com projetos de Software Livre há mais de dez anos.

Além disso, apenas o autor deste trabalho avaliou as mensagens de *commit*, o que resulta em três ameaças: tédio, cansaço e viés. O fato do autor estudar arquitetura de software, há cinco anos, e ter atuado por quatro anos como arquiteto de software em sua experiência profissional pode ajudar a reduzir os vieses. Para minimizar as outras ameaças, a avaliação ocorreu em etapas, sendo realizada em cinco dias consecutivos com intervalo de dois dias; além disso, nos dias de avaliação, o número de mensagens de *commit* analisado foi mantido entre 60 e 100 (usando oitenta dias para avaliação completa).

A seguir, são apresentadas as conclusões e os trabalhos futuros deste estudo.

3.7 CONCLUSÕES

Esse capítulo mostrou que modificar a arquitetura é uma preocupação presente durante o ciclo de vida de um projeto. Também foi apresentada a possibilidade de identificação de informações arquiteturais em mensagens de *commit*, através da localização de traços arquiteturais. Todos os tópicos de informações arquiteturais foram encontrados no repositório analisado em mensagens de *commit*. Viu-se também que os colaboradores, que fazem alterações mais comuns, como correções de erros, refatoração ou implementação de novos recursos, são os mesmos que implementam alterações na arquitetura. Compreendeu-se que *commits* relacionados às alterações arquiteturais têm mensagens mais longas e alteram mais arquivos do que *commits* não-arquiteturais. Além disso, a maioria dos colaboradores concentra as suas mudanças arquiteturais em determinados períodos.

Os resultados trazem novas perspectivas para a recuperação de informações sobre a evolução arquitetural, tanto na academia quanto na indústria. Considerando que é difícil encontrar informações sobre a arquitetura de software de um projeto e sua evolução,

propõe-se a extração dessas informações a partir das mensagens de *commit*. É possível indicar quais parâmetros devem ser considerados na busca desses *commits*: perfil dos autores de alterações, tópicos mais frequentes, tamanho da mensagem de *commit*, número de arquivos modificados e repetição desses arquivos nas alterações.

Para a indústria e para projetos de Software Livre, esses resultados são úteis tanto para a compreensão da evolução de um projeto quanto para desenvolvedores iniciantes procurarem informações arquiteturais por si mesmos quando a documentação estiver obsoleta ou indisponível.

Para pesquisas de engenharia de software, os resultados abrem um novo tipo de fonte de informações arquiteturais: documentação não-estruturada do projeto. Até hoje, muitos pesquisadores usavam apenas o código-fonte como fonte de informação arquitetural. Em processos de desenvolvimento, com pouca documentação, os pesquisadores de engenharia de software são desafiados a encontrar informações arquiteturais em fontes de dados não convencionais, além do código-fonte, como listas de discussão e *issue trackers*.

O Apêndice A traz um *guideline* sobre como utilizar os traços arquiteturais para recuperar informações sobre mudanças na arquitetura de um projeto ao longo de sua evolução.

Chapter 4

MÉTRICAS DE SOFTWARE: CARACTERIZANDO MODIFICAÇÕES ARQUITETURAIS A PARTIR DA VARIAÇÃO DE VALORES DE MÉTRICAS DE CÓDIGO EM COMMITS

4.1 INTRODUÇÃO

Os desenvolvedores tomam decisões durante o ciclo de vida do software relacionadas a diferentes questões como o design (e a arquitetura), a codificação, os requisitos, a correção de problemas e outras questões. Essas decisões podem mudar durante a evolução do projeto e o conhecimento sobre essas mudanças é essencial durante a manutenção e a evolução do software, pois alterações inadvertidas podem resultar em resultados indesejáveis (GURP; BOSCH, 2002; FARID; AZAM; IQBAL, 2011).

Um tipo de ferramenta, que apoia o acompanhamento das alterações no projeto, são os Sistemas de Controle de Versão (SCV). Eles são capazes de apresentar a evolução dos artefatos do projeto, indicando o período das mudanças, autores e o que foi modificado. De posse dessas informações, os desenvolvedores podem avaliar as mudanças ocorridas em cada versão, analisando o código submetido e comparando-o à versão imediatamente anterior. As métricas, de um modo geral, podem ajudar nessa análise, pois mostram as condições de tamanho, complexidade, acoplamento e coesão do código produzido, tornando possível a comparação com o estado anterior do código.

A despeito de mudanças na arquitetura, Behnamghader et al. (2017) citam que a prevalência da decadência arquitetural, comprometendo a compreensão dos desenvolvedores sobre as mudanças, não tem sido suficiente para que sejam produzidos estudos empíricos sobre as mudanças arquiteturais em si. Ding et al. (2014) já haviam relatado a ausência de informações sobre a arquitetura de software e a sua evolução (modificações). Eles relatam que avaliaram 2000 projetos de Software Livre, hospedados nos quatro principais repositórios de projetos desse tipo, e apenas 108 possuíam algum tipo de documentação e, desses, apenas o modelo, o sistema e missão do sistema foram encontrados. Já Kleebaum et al. (2018) alertaram para o cenário de não-documentação das decisões arquiteturais e

apresentam no trabalho uma estratégia para auxiliar os desenvolvedores no processo de busca por informações arquiteturais.

Explorando as informações disponíveis nos SCVs, o trabalho de Motta, Souza e Sant'Anna (2018) selecionou um conjunto de versões do projeto KDELibs que contêm modificações na arquitetura, identificadas através de traços arquiteturais, dentro de um determinado período de tempo de seu ciclo de vida. Para a seleção realizada pelos autores, foram consideradas as modificações descritas nas mensagens de *commit* de cada versão. No entanto, esse estudo se apoia em um processo semiautomatizado e depende de avaliação do desenvolvedor e de sua experiência para que se obtenha êxito. Acredita-se que as métricas podem ajudar na caracterização de mudanças na arquitetura, uma vez que o código-fonte é a entrada para a maioria dos métodos de recuperação de parte das informações sobre a arquitetura de um sistema (arquitetura estática). Exemplos desse tipo de estudos são Mancoridis et al. (1999), Riva et al. (2004), Pinzger (2005), e Knodel et al. (2006).

Assim, neste estudo, pretende-se analisar as variações das métricas de tamanho, complexidade, acoplamento e coesão das versões apontadas no trabalho supramencionado (*commits* arquiteturais) e comparar essas variações com variações das métricas observadas em outras versões onde não houve modificação arquitetural (*commits* não-arquiteturais). As versões (*commits*) não-arquiteturais foram escolhidas aleatoriamente dentro do mesmo período delimitado para a escolha dos *commits* contendo alterações arquiteturais. Esse é o primeiro trabalho que usa o estudo da evolução de um projeto através de métricas para mostrar o que acontece no projeto após as mudanças arquiteturais em comparação com mudanças no projeto que não atingem a arquitetura do mesmo.

Neste estudo, foram calculadas 16 diferentes métricas que avaliam o tamanho, a complexidade, o acoplamento e a coesão do código-fonte do projeto analisado. Essas métricas foram apontadas por Nuñez-Varela et al. (2017) como estando entre a mais citadas em meio as 300 métricas encontradas nos estudos publicados entre 2010 e 2015. Para isso, foram escolhidas versões apontadas em um estudo anterior que foram identificadas como contendo modificações arquiteturais no projeto KDELibs. As respectivas versões anteriores de cada uma das versões escolhidas foram analisadas e, de posse disso, foram calculadas as diferenças entre essas métricas, buscando avaliar o que mudou, em termos de código-fonte, no projeto.

O mesmo procedimento foi utilizado com versões do projeto que foram escolhidas forma aleatória, as quais não tiveram alterações arquiteturais detectadas. Com a comparação entre as variações de métricas obtidas entre as versões com mudanças arquiteturais e as versões sem mudanças na arquitetura, espera-se identificar se há diferenças no comportamento dos valores das métricas estudadas. Os resultados obtidos indicam que 6 métricas (sendo 5 de tamanho e 1 de complexidade), dentre as avaliadas, apresentam variações mais acentuadas em *commits*, contendo mudanças arquiteturais que em *commits* onde a arquitetura não foi modificada. Da mesma forma, observou-se que 3 métricas (sendo 2 de acoplamento e 1 de coesão), dentre as avaliadas, apresentam variações mais acentuadas em *commits* contendo mudanças arquiteturais que em *commits* onde a arquitetura não foi modificada.

O restante deste capítulo está organizado da seguinte maneira: os trabalhos relacionados

estão presentes na Seção 4.2. O design do estudo é apresentado na Seção 4.3. A Seção 4.4 mostra os resultados deste trabalho. A Seção 4.5 apresenta as ameaças à validade identificadas. Finalmente, a Seção 4.6 mostra as conclusões obtidas.

4.2 TRABALHOS RELACIONADOS

Há algum tempo, a pesquisa em Engenharia de Software tem utilizado as métricas de software para caracterizar partes de software que precisam de manutenção, que estão propensas à bugs e que necessitam de *refactoring*, dentre outras utilidades. Nesse sentido, foi realizada uma revisão bibliográfica não-sistemática em busca de um conjunto de trabalhos que se relacionam com o tema desenvolvido neste estudo. Inicialmente, são apresentados estudos sobre métricas de software, em seguida, trabalhos que usam métricas para a predição de elementos de qualidade de software e outros temas relacionados e, por fim, trabalhos onde tentou-se usar métricas para ajudar na identificação de características em um projeto de software.

As métricas de software vêm sendo utilizadas para caracterizar elementos de projetos de software há algum tempo. Nesse sentido, e dada a diversidade das métricas disponíveis, alguns pesquisadores utilizaram as métricas de software para caracterizar elementos de projetos de software. Hitz e Montazeri (1995), por exemplo, fazem uma análise sobre as métricas de coesão e acoplamento, apontando a sua importância aos dados fornecidos sobre atributos de qualidade, tais como confiabilidade, manutenção e usabilidade. Eles apontam inconsistências de seu uso e aplicação – fundadas sobre terem sido criadas para avaliar sistemas de software estruturados e serem agora aplicados aos sistemas OO –, indicando elementos que devem ser considerados nos cálculos dessas métricas e propondo uma métrica de acoplamento que contemple os aspectos criticados.

Já Harrison, Counsell e Nithi (1997) analisaram o estado da arte das métricas de design orientado a objetos a partir das suítes de métricas de Chidamber e Kemerer (1994), Lorenz e Kidd (1994) e Abreu, Goulão e Esteves (1995). Como resultado, eles apontam problemas comuns aos três conjuntos de métricas, tais como a ambiguidade entre as métricas, a falta de avaliações empíricas e a desconexão com as definições de qualidade de software utilizadas. Os autores sugerem, como melhorias incorporação das definições de qualidade de software às métricas, a construção de ferramentas de suporte para coleta das mesmas e avaliações empíricas amplas de cada uma delas.

Xenos et al. (2000), por sua vez, propuseram uma taxonomia dessas métricas, buscando orientar seus usos e aplicações. Nesse estudo é apresentado o estado da arte relativo à métricas de qualidade de software à época. Como resultado, também indicaram uma taxonomia sobre métricas orientadas a objetos, a qual se divide em cinco categorias: Métricas de Classes, de Métodos, de Acoplamento, de Herança e de Sistemas. Além disso, esses autores fazem indicações sobre quando e como cada uma das métricas pode ser usada de acordo com as características de qualidade que um profissional deseja enfatizar.

Outros autores se dedicaram a estudar o efeito de métricas estáticas de software sobre projetos desenvolvidos sob o paradigma orientado a objetos, a exemplo de Sharma et al. (2012). Nessa análise, esses autores buscaram caracterizar elementos da orientação a objetos como construtores, classes e herança, a partir dos resultados obtidos com valores

de métricas estáticas.

Anteriormente, porém, Bengtsson (1998) trouxe à tela o desafio de propor um conjunto de métricas para arquitetura de software. O autor discorre sobre as dificuldades na adaptação de métricas orientadas a objetos para um nível de abstração mais alto como o nível arquitetural, e apresenta um método para design de arquitetura chamado ARCS. Para avaliá-lo, o autor propõe um modelo de previsão de manutenção baseado no modelo de (LI; HENRY, 1993), concluindo que a validação dos preditores não poderia ser feita com estudos tradicionais.

Em acordo com o dissertado acima, prever a necessidade e o esforço de manutenção, é um tema recorrente nos estudos usando métricas de software (LI; HENRY, 1993; MUTHANNA et al., 2000; DAGPINAR; JAHNKE, 2003; DARCY et al., 2005; OLAGUE; ETZKORN; COX, 2006; JIN; LIU, 2010). Dentre esses estudos, Li e Henry (1993) analisaram métricas procedurais aplicadas a sistemas orientados a objetos e refletiram sobre os possíveis usos dos resultados obtidos. Eles concluíram que há uma forte ligação entre métricas de sistemas orientados a objetos e esforço de manutenção e o esforço de manutenção pode ser previsto, usando a combinação de algumas das métricas estudadas.

Muthanna et al. (2000), por sua vez, apresentam um estudo que investigou o uso de métricas de design de software para estimar, estatisticamente, a capacidade de manutenção de grandes sistemas de software e para identificar módulos propensos a erros. Eles propõem um modelo de previsão linear baseado em um conjunto mínimo de métricas de software em nível de design e o avaliam aplicando-o a sistemas de software industriais.

Já Dagpinar e Jahnke (2003) apresentaram uma investigação sobre quais métricas conseguem predizer partes de um software que devem passar por manutenção, usando análise de regressão multivariada. Como resultado, obtiveram que métricas de tamanho e acoplamento interno, tais como TIIC[1], TNIIC[2], TIEC[3], TNIEC[4], LOC, TNOS[5] e NIM[6], são adequadas para esse fim, enquanto métricas de herança, coesão e acoplamento indireto não servem para indicar a necessidade de manutenção, a exemplo de DiT, NOC, LCOM, LCC[7].

Ainda nesse sentido, Darcy et al. (2005) examinaram a métrica complexidade estrutural (SC) através de um experimento controlado, envolvendo tarefas de manutenção e engenheiros de software profissionais. O estudo está fundado em três relacionamentos: (i) Complexidade estrutural e esforço de manutenção; (ii) Acoplamento e coesão como dimensões da complexidade estrutural; e (iii) Relacionamento entre Acoplamento e Coesão. Os autores afirmam que coesão e acoplamento devem ser considerados juntos ao projetar software, a fim de controlar, efetivamente, sua complexidade estrutural.

Olague, Etzkorn e Cox (2006), por sua vez, estudaram se a degradação do software pode ser medida, usando a complexidade ciclomática. Eles adotaram a abordagem

[1]Acoplamento total por importação de herança
[2]Acoplamento total por importação sem herança.
[3]Acoplamento total por exportação de herança.
[4]Acoplamento total sem exportação de herança.
[5]Número total de declarações.
[6]Número de métodos de instância.
[7]Falta de Coesão de Classe.

de definição de decaimento de software em termos da entropia de Shannon (2001) e complexidade ciclomática de McCabe (1976), usando critérios de limite de complexidade estabelecidos pelo setor. Os resultados apontaram que os componentes com alta complexidade ciclomática estavam associados à mais atividades de manutenção do que os componentes com menor complexidade ciclomática.

Já Jin e Liu (2010) apresentaram a aplicação de métodos não-supervisionados de aprendizagem para prever manutenção de software, baseando-se em sete métricas orientadas a objetos. Utilizando o método da máquina de vetores de suporte (SVM), os autores definiram que as métricas utilizadas (LCOM, NOC, DiT, WMC[8], RFC, DAC[9], MPC[10] e NOM), de forma agrupada, eram preditores úteis para o esforço de manutenção.

O uso de métricas para identificar e predizer defeitos também foi largamente utilizado ao longo da pesquisa em Engenharia de Software (TANG; KAO; CHEN, 1999; MISRA; BHAVSAR, 2003; THWIN; QUAH, 2005; JURECZKO; SPINELLIS, 2010). Tang, Kao e Chen (1999) investigaram a correlação entre métricas de projetos OO (propostas no estudo de (CHIDAMBER; KEMERER, 1994) - WMC, DiT, NOC, CBO e RFC) à ocorrência de falhas. Eles observaram que as métricas WMC e RFC são úteis na predição de falhas de software.

Já Misra e Bhavsar (2003) buscaram relacionar medidas preditoras de bugs e a ocorrência de bugs em um conjunto de projetos. Eles mostraram que poucas das métricas analisadas estão desconectadas dos bugs observados. Dessa forma, concluem que essas métricas são boas preditoras de bugs em sistemas e que seu uso se trata de um método consistente para controlar a qualidade dos projetos de software em evolução/desenvolvimento.

Thwin e Quah (2005), no mesmo sentido, apresentaram a aplicação de redes neurais sobre valores de métricas orientadas a objetos, buscando informar a previsão do número de defeitos por classe e a previsão de linhas alteradas por classe. Para esse estudo, os autores usaram métricas de software orientadas a objetos (herança, coesão, acoplamento e alocação de memória) e métricas de complexidade tradicionais. Como conclusão, os autores afirmaram que as métricas orientadas a objetos escolhidas, neste estudo, parecem ser úteis na previsão da qualidade do software.

Ainda em busca de prever a existência de bugs, Jureczko e Spinellis (2010) apresentaram uma ferramenta para calcular métricas de software (`Ckjm`). A partir dessa ferramenta, esses autores produziram modelos de previsão de defeitos, tendo como entrada as métricas extraídas com a ferramenta `Ckjm`, buscando apontar aos desenvolvedores quais as classes com maior propensão a defeitos e falhas.

Outros estudos relacionaram métricas à identificação de código relacionado à oportunidades de *refactoring* como (IYER, 2009; BIEGEL et al., 2011; CINNÉIDE et al., 2012; SIMONS; SINGER; WHITE, 2015). Buscando identificar trechos de código candidatos à *refactoring*, o trabalho de Iyer (2009) desenvolveu um estudo buscando relacionar métricas orientadas a objetos (RMI, WMC e LCOM) à oportunidade de refatoração. Como resultados, ela obteve que a evolução dos softwares leva ao aumento da complexidade e diminuição da qualidade do software e que as atividades de refatoração impactam, diretamente, na

[8]Peso dos métodos por classe.

[9]Acoplamento por abstração de dados.

[10]Acoplamento por passagem de mensagem.

complexidade e qualidade dos softwares desenvolvidos.

Nesse trabalho, Biegel et al. (2011) realizaram a replicação de um estudo usando métricas de similaridade (AST, Token e busca em texto) para identificar trechos de códigos candidatos à refatoração. Ao sobreporem os resultados obtidos com cada uma, encontraram resultados parecidos e concluíram que a aplicação dessas métricas, em conjunto, podem indicar oportunidades de refatoração, pois identificam códigos clonados, por exemplo.

Cinnéide et al. (2012), por sua vez, apresentaram um estudo onde cinco métricas de coesão foram utilizadas para indicar pontos de refatoração em 8 sistemas desenvolvidos em Java. Eles descobriram que, na maioria dos casos, as métricas discordaram sobre a necessidade de refatoração do código. Foi sublinhado que as métricas de coesão não apenas incorporam noções diferentes de coesão, mas também incorporam noções conflitantes desse tipo de métrica, refutando assim a possibilidade de unificação das métricas.

Já Simons, Singer e White (2015) fizeram um estudo comparando os pontos de melhorias em projetos de Software Livre que foram indicados por métricas estáticas de software (Falta de Coesão por método (LCOM) e as suas variações, Qualidade do Módulo (MQ) e Métrica de Avaliação (EVM)) e as opiniões de engenheiros de software sobre esses pontos de melhoria. Como conclusão, afirmaram que a melhor forma de identificar pontos de refatoração é uma abordagem humana em loop, pois os engenheiros discordaram, em sua maioria, dos resultados obtidos com a aplicação dessas métricas.

Estudos apresentando outros tipos de aplicações de métricas também foram encontrados nessa revisão de literatura a exemplo de Simon, Steinbruckner e Lewerentz (2001), Aggarwal et al. (2006), Meirelles et al. (2010). Simon, Steinbruckner e Lewerentz (2001) apresentaram um conjunto de métricas que pretendem indicar porções de código que contenham *bad smells*. Essa proposta foi validada junto aos desenvolvedores, usando uma ferramenta de visualização baseada nesse conjunto selecionado estatisticamente.

Aggarwal et al. (2006) fizeram uma análise sobre métricas orientadas a objetos. Eles estudaram 22 diferentes métricas e apresentaram uma análise estatística que culmina na indicação de 6 métricas como sendo suficientes para a descrição de classes. Para indicar essa redução, foi utilizada uma técnica estatística chamada Análise de Componentes Principais (PCA) em um estudo de correlação entre as métricas de tamanho e métricas de coesão, acoplamento e herança.

Por fim, o trabalho de Meirelles et al. (2010) apresentou uma pesquisa sobre a correlação entre métricas de software e a atratividade de projetos de Software Livre. Ao final, os pesquisadores descobriram uma relação direta entre a métrica LOC e adesão de novos desenvolvedores aos projetos. No sentido oposto, a complexidade estrutural apresenta uma relação inversa em relação à atratividade junto a novos desenvolvedores.

Como se vê, as métricas de software têm múltiplas aplicações na identificação e caracterização de fenômenos de desenvolvimento de software. A próxima seção apresenta o design do estudo realizado com o intuito de buscar novas informações sobre mudanças arquiteturais utilizando para isso a coleta de valores de métricas de código.

4.3 DESIGN DO ESTUDO

No intuito de buscar novas evidências sobre mudanças arquiteturais, utilizando o histórico evolutivo de um projeto e, dessa forma, compreender outros elementos dessas mudanças a partir do código-fonte, usando para isso as métricas disponíveis, foi realizado um estudo empírico com as seguintes etapas: (i) seleção de versões do projeto após mudanças arquiteturais (e que contenham traços arquiteturais em sua descrição) e de versões sem impacto na arquitetura; (ii) seleção das métricas utilizadas para a análise e construção das hipóteses sobre as alterações arquiteturais, e; (iii) extração dos valores das métricas selecionadas e processamento dos resultados obtidos. Nas subseções seguintes, serão detalhadas as subquestões de pesquisa, como foi realizada a escolha do projeto participante, o processo de coleta de métricas nas versões desse projeto e como os dados obtidos foram analisados.

4.3.1 Questões de Pesquisa

Em busca de informações sobre mudanças arquiteturais, estruturou-se este estudo em quatro subquestões de pesquisa, derivadas da questão **RQ2**, que organizaram as métricas investigadas em grupos distintos: tamanho, complexidade, acoplamento e coesão. Desta forma, as subquestões de pesquisa estudadas foram:

RQ2.1: Alterações que envolvem a arquitetura de um projeto se refletem em maiores variações no tamanho do código que outros tipos de alterações no projeto?
Nessa subquestão, será analisado o conjunto das métricas de tamanho coletadas nas versões do projeto com alterações arquiteturais e comparadas com alterações no projeto onde a arquitetura não sofreu alterações. Essa análise ocorrerá a partir do delta produzido entre os valores das métricas de tamanho obtidas nos *commits* com mudanças arquiteturais e os valores obtidos com suas respectivas versões anteriores. O mesmo cálculo será realizado com os *commits* onde não houve mudança na arquitetura. Uma vez produzidos esses conjuntos de dados, será realizada uma comparação estatística entre as variações obtidas nos valores das métricas de tamanho nos *commits* com mudanças na arquitetura do software e os *commits* sem mudanças na arquitetura.

RQ2.2: Alterações que envolvem a arquitetura de um projeto se refletem em maiores variações na complexidade do código que outros tipos de alterações no projeto?
Nessa subquestão, será analisado o conjunto das métricas de complexidade coletadas nas versões do projeto com alterações arquiteturais e comparadas com alterações no projeto onde a arquitetura não sofreu alterações. Essa análise ocorrerá a partir do delta produzido entre os valores das métricas de complexidade obtidas nos *commits* com mudanças arquiteturais e os valores obtidos com suas respectivas versões anteriores. O mesmo cálculo será realizado com os *commits* onde não houve mudança na arquitetura. Uma vez produzidos esses conjuntos de dados, será realizada uma comparação estatística entre as variações obtidas nos valores das métricas de complexidade nos *commits* com mudanças na arquitetura do software e os *commits* sem mudanças na arquitetura.

RQ2.3: Alterações que envolvem a arquitetura de um projeto se refletem em maiores variações no acoplamento do código-fonte que outros tipos de alterações no projeto?
Nessa subquestão, será analisado o conjunto das métricas de acoplamento coletadas nas versões do projeto com alterações arquiteturais e comparadas com alterações no projeto onde a arquitetura não sofreu alterações. Essa análise ocorrerá a partir do delta produzido entre os valores das métricas de acoplamento obtidas nos *commits* com mudanças arquiteturais e os valores obtidos com suas respectivas versões anteriores. O mesmo cálculo será realizado com os *commits* onde não houve mudança na arquitetura. Uma vez produzidos esses conjuntos de dados, será realizada uma comparação estatística entre as variações obtidas nos valores das métricas de acoplamento nos *commits* com mudanças na arquitetura do software e os *commits* sem mudanças na arquitetura.

RQ2.4: Alterações arquiteturais em um projeto se refletem em maiores variações na coesão dos módulos do projeto que outros tipos de alterações no projeto?
Nessa subquestão, será analisado o conjunto das métricas de coesão coletadas nas versões do projeto com alterações arquiteturais e comparadas com alterações no projeto onde a arquitetura não sofreu alterações. Essa análise ocorrerá a partir do delta produzido entre os valores das métricas de coesão obtidas nos *commits* com mudanças arquiteturais e os valores obtidos com suas respectivas versões anteriores. O mesmo cálculo será realizado com os *commits* onde não houve mudança na arquitetura. Uma vez produzidos esses conjuntos de dados, será realizada uma comparação estatística entre as variações obtidas nos valores das métricas de coesão nos *commits* com mudanças na arquitetura do software e os *commits* sem mudanças na arquitetura.

A Tabela 4.1 apresenta um resumo entre as subquestões de pesquisa apresentadas e as métricas levadas em conta na avaliação. A próxima subseção apresenta a metodologia empregada para a coleta de métricas ao longo da evolução do projeto KDELibs.

Table 4.1: Relação das métricas consideradas em cada subquestão de pesquisa.

	RQ2.1	**RQ2.2**	**RQ2.3**	**RQ2.4**
Acoplamento entre Objetos (CBO)			X	
Complexidade Ciclomática Média por Método (ACCM)		X		
Complexidade Estrutural (SC)				X
Conexões Aferentes por Classe (ACC)			X	
Falta de Coesão dos Métodos (LCOM4)				X
Linhas de Código (LOC)	X			
Média de Linhas de Código por Método (AMLoc)	X			
Número de Atributos (NOA)	X			
Número de Atributos Públicos (NPA)		X		
Número de Classes Filhas (NOC)			X	
Número de Métodos (NOM)	X			
Número de Métodos Públicos (NPM)		X		
Número Médio de Parâmetros (ANPM)	X			
Profundidade da Árvore de Herança (DiT)			X	
Resposta por Classe (RFC)			X	
Tamanho do Maior Método (MMLoc)	X			

4.3.2 Coletando Métricas de Código ao longo da evolução do projeto

O projeto KDELibs foi escolhido como objeto deste estudo. Isso se deveu aos fatores a seguir: (i) trata-se de um grande projeto em termos de tamanho do código-fonte, (ii) é central para o ecossistema KDE e por isso interage com diversos outros projetos desse ecossistema, (iii) esteve ativo por um período de tempo considerável (mais de 20 anos), (iv) teve diversos colaboradores enquanto ativo e (v) já possui um estudo que identificou, a partir de mensagens de *commit*, versões onde foram apontadas mudanças arquiteturais no projeto.

Para analisar as versões do projeto KDELibs, foi escolhida a ferramenta Analizo (TERCEIRO et al., 2010). Essa ferramenta foi escolhida porque (i) se trata de uma ferramenta livre – por isso suscetível a ajustes (caso fosse necessário) –, (ii) já testada – com 31 estudos publicados que a utilizaram – e (iii) que conta com desenvolvedores acessíveis.

4.3.2.1 Métricas consideradas neste estudo

Para este estudo foram selecionadas todas as métricas calculadas pela ferramenta Analizo. Elas foram selecionadas porque constam entre as mais citadas no mapeamento sistemático realizado por Nuñez-Varela et al. (2017) que buscaram estudos publicados entre 2010 e 2015 sobre as métricas de código-fonte. Essas métricas foram coletadas para cada versão do código-fonte do projeto KDELibs escolhidas para participar deste estudo. Apesar de oferecidos pela ferramenta e registrados no banco de dados produzido, os sumários estatísticos disponibilizados pela ferramenta (curtose, média, moda, soma, variância, etc) para cada métrica não foram utilizados neste estudo. Esses dados foram descartados, porque a análise se concentrou nos valores de métricas obtidas em conjunto e em separado para cada versão do projeto participante do estudo.

Foi utilizada a classificação de métricas definida no trabalho de Meirelles (2013) e elas foram separadas em quatro tipos: Tamanho, Complexidade, Acoplamento e Coesão. As métricas de tamanho coletadas foram: Média de linhas de código por método (AMLoc), Número Médio de Parâmetros (ANPM), Linhas de código (LOC), Número de Atributos (NOA), Número de métodos (NOM) e Tamanho do Maior Método (MMLoc). O grupo de métricas de complexidade extraídas foram: Complexidade Ciclomática Média por Método (ACCM), Número de atributos públicos (NPA) e Número de métodos públicos (NPM). Já as métricas de acoplamento estudadas foram: Conexões Aferentes por Classe (ACC), Acoplamento entre Objetos (CBO), Profundidade da Árvore de herança (DiT), Número de Classes filhas (NOC) e Resposta por Classe (RFC). Por fim, o grupo de métricas de coesão estudadas foram: Falta de Coesão dos Métodos (LCOM4) e Complexidade Estrutural (SC). Nas subseções, a seguir, será apresentada a classificação e serão revisitadas as suas respectivas definições.

Métricas de Tamanho

As métricas de tamanho quantificam o tamanho de um software ou de partes dele. Neste estudo, as métricas de tamanho utilizadas a partir dos cálculos efetuados pela

ferramenta Analizo são: Média de linhas de código por método (AMLoc), Número Médio de Parâmetros (ANPM), Linhas de código (LOC), Número de Atributos (NOA), Número de métodos (NOM) e Tamanho do Maior Método (MMLoc). Trata-se de diferentes medidas de tamanho do código-fonte produzido.

A métrica "Média de Linhas de Código por Método (AMLoc)" mensura o tamanho de um arquivo/módulo através do cálculo do tamanho médio dos métodos, aplicando-se a razão entre o total de linhas de código e a quantidade de métodos de um arquivo/módulo (LORENZ; KIDD, 1994). A métrica "Número Médio de Parâmetros (ANPM)", por sua vez, mensura o tamanho de um módulo/arquivo a partir da média de parâmetros presentes nas assinaturas dos métodos/funções que ele possui. Seu cálculo é expresso pela razão entre a quantidade de parâmetros presentes nas assinaturas dos métodos e o número de métodos de um módulo/arquivo (BASILI; ROMBACH, 1987).

A métrica "Linhas de Código (LOC)" mede o tamanho de um arquivo/módulo considerando a quantidade total de linhas no código-fonte de um arquivo/módulo (BOEHM, 1984). Para esse estudo, foi considerada uma de suas variações a métrica ELOC. Em sua valoração, ela descarta as linhas em branco, assim como as linhas, contendo apenas comentários no corpo do código (SHEPPERD, 1990). Já a métrica, "Número de Atributos (NOA)" mensura o tamanho de um arquivo/módulo, através da contagem de seus atributos (ou variáveis) (HENDERSON-SELLERS, 1995).

A métrica "Número de Métodos (NOM)" mensura o tamanho de um arquivo/módulo através da contagem de métodos de um arquivo/módulo (HENDERSON-SELLERS, 1995). Por fim, a métrica "Tamanho do Maior Método (MMLOC)" mensura o tamanho de um arquivo/módulo, indicando o tamanho do maior método de uma classe através da análise da quantidade de linhas de cada um dos métodos disponíveis (TERCEIRO et al., 2010). Na próxima subseção, serão apresentadas as métricas de complexidade utilizadas neste estudo.

Métricas de Complexidade

As métricas de complexidade são relacionadas à manutenção do código, pois quanto mais interações, maior a chance de mudanças em uma parte do código afetar outras que estejam relacionadas. Além disso, atributos e métodos marcados como de acesso público permitem que módulos externos os manipulem diretamente, inserindo aí mais um aspecto a ser observado em operações de manutenção. As métricas de complexidade calculadas pela ferramenta Analizo são: Complexidade Ciclomática Média por método (ACCM), Número de atributos públicos (NPA) e Número de métodos públicos (NPM).

A métrica "Complexidade Ciclomática Média por Método (ACCM)" mensura a complexidade de um método ou uma função através da contagem do número de caminhos independentes (desvios de fluxos) que ele pode executar até o seu final (MCCABE, 1976). A métrica "Número de Atributos Públicos (NPA)", por sua vez, mensura a complexidade de um arquivo/módulo através do grau de encapsulamento de uma classe, contando a quantidade de atributos públicos nela presentes (LORENZ; KIDD, 1994). Já a métrica, "Número de Métodos Públicos (NPM)" mensura a complexidade de um arquivo/módulo através da contagem do número de métodos/funções públicas disponíveis (LORENZ; KIDD, 1994). Na próxima subseção, serão apresentadas as métricas de acoplamento utilizadas neste

estudo.

Métricas de Acoplamento
As métricas de acoplamento calculadas pela ferramenta Analizo são: Conexões Aferentes por Classe (ACC), Acoplamento entre Objetos (CBO), Profundidade da Árvore de Herança (DiT), Número de Classes Filhas (NOC) e Resposta por Classe (RFC).

A métrica "Conexões Aferentes por Classe (ACC)" mensura o grau de acoplamento do arquivo/módulo analisado através da contagem de arquivos/módulos que dependem do arquivo/módulo avaliado (MARTIN, 1994). Já a métrica "Acoplamento entre Objetos (CBO)" mensura o acoplamento do arquivo/módulo analisado através da contagem de arquivos/módulos que dependem do arquivo/módulo (conexão eferente) avaliado e dos arquivos/módulos que dele depende (conexão aferente) (CHIDAMBER; KEMERER, 1994).

A métrica "Profundidade da Árvore de Herança (DiT)" mensura o acoplamento do arquivo/módulo através da contabilidade de superclasses do arquivo/módulo analisado (CHIDAMBER; KEMERER, 1994). Outra métrica de acoplamento é o "Número de Classes Filhas (NOC)" mensura o acoplamento do arquivo/módulo através da contabilidade das subclasses do arquivo/módulo analisado (CHIDAMBER; KEMERER, 1994).

Por fim, a métrica "Resposta por Classe (RFC)" mensura o acoplamento do arquivo/módulo através da contagem de todos os métodos daquele arquivo/módulo e todos as funções/métodos invocados diretamente por esse arquivo/módulo (CHIDAMBER; KEMERER, 1994). Na próxima subseção, serão apresentadas as métricas de coesão utilizadas neste estudo.

Métricas de Coesão
As métricas de coesão calculadas pela ferramenta Analizo são: Falta de Coesão dos Métodos (LCOM4) e Complexidade Estrutural (SC).

A métrica "Falta de Coesão dos Métodos (LCOM4)" mensura a coesão de um arquivo/módulo, através da construção de grafo não-orientado, onde os *nós* são as funções/métodos e as variáveis/atributos de um arquivo/módulo e, para cada função/método, existe uma aresta entre o nó que o representa e uma outra função/método ou variável/atributo que é acessado (CHIDAMBER; KEMERER, 1994; HITZ; MONTAZERI, 1995). O valor da LCOM4 é determinado pelo número de componentes fracamente conectados nessa estrutura.

A métrica "Complexidade Estrutural (SC)" mensura a coesão dos arquivos/módulos analisados através do produto de duas outras métricas: CBO e LCOM4. O desejável para essa métrica, segundo (MEIRELLES, 2013), é que se mantenham aqui baixos valores, pois indicam alta coesão e baixo acoplamento. A subseção seguinte apresenta o processo de coleta de dados.

4.3.2.2 Definição do conjunto analisado

Para esse estudo, foram selecionados 928 *commits*, separados em dois conjuntos. A partir do estudo de Motta, Souza e Sant'Anna (2018) foram selecionados 232 *commits*

classificados naquele estudo como sendo relacionados à modificações arquiteturais no projeto KDELibs. Essa classificação deu-se de acordo com as descrições de mudanças ocorridas disponíveis em mensagens de *commit* registradas no Sistema de Controle de Versões (SCV) utilizado. Também foram selecionadas as versões imediatamente anteriores a cada uma dessas 232 versões, contendo mudanças arquiteturais, com o objetivo de se identificar a variação ocorrida em cada versão do projeto (aqui chamadas de delta). Como se pretende diferenciar a variação dos valores das métricas estudadas nos *commits* com modificações arquiteturais em relação às métricas estudadas nos *commits* não-arquiteturais, foram selecionadas de forma aleatória 232 outras versões do projeto, dentro do mesmo período do projeto onde foram selecionadas as 232 versões com mudanças na arquitetura. Da mesma forma que no outro conjunto, foram selecionadas os *commits* anteriores para cada *commit* não-arquitetural selecionado, tendo o mesmo fundamento: estudar a variação dos valores das métricas em modificações não-arquiteturais.

4.3.2.3 O processo de coleta de dados

Visando automatizar o processo de coleta das métricas, uma vez que a ferramenta Analizo possui interface de linha de comando e disponibiliza os seus resultados na saída padrão, foi construído um *script* que gerenciou a execução do Analizo. Como entrada, esse *script* solicita um conjunto de chaves (utilizadas para identificar cada versão no Git - SCV utilizado pelo projeto KDELibs) relacionadas a cada *commit* escolhido para participar deste estudo. Esse *script* redireciona os resultados oferecidos pelo Analizo para arquivos de texto individualizados para cada versão analisada.

A seguir, foi executada a ferramenta Analizo em cada uma das 928 versões selecionadas para o estudo (232 com mudanças arquiteturais reportadas e as suas respectivas versões anteriores e outras 232 versões não-arquiteturais selecionadas e suas versões respectivas anteriores). Para cada arquivo/módulo que compõe as versões do projeto selecionadas, foi gerado o valor correspondente para cada métrica extraída pelo Analizo. Esses dados foram armazenados em um *dataset* hospedado em um SGBD PostgreSQL, normalizado na 3FNBC[11], contando com 18 tabelas, chamado Metrics e cujo modelo de dados é apresentado na Figura 4.1. Essa figura apresenta o projeto físico implementado no banco de dados Metrics. O Apêndice D apresenta os scripts utilizados nesse estudo.

4.3.2.4 A formação dos conjuntos analisados

Para a análise dos dados coletados, os valores obtidos para cada métrica foram agrupados em diferentes conjuntos. Dessa forma, para cada métrica tem-se os conjuntos de dados (A_w, A_i^{w-1}, N_w e N_i^{w-1}) descritos abaixo.

Seja V o conjunto finito de todas as versões do projeto KDELibs e V_x uma versão qualquer desse conjunto que contém um conjunto finito de arquivos (F_i^x), de tamanho n. Existe um conjunto A finito contido em V ($A \subset V$) tal que, para todo A_w existe um V_x correspondente. O conjunto A contém as versões do projeto KDELibs onde

[11]Extensão da 3Forma Normal proposta por Boyce-Codd (CODD, 1974)

houve modificações arquiteturais e cada versão A_w é composta por um conjunto finito de arquivos (A_i^w), de tamanho m. Para cada A_w, existe um conjunto A_{w-1} (que também possui correspondente em V), o qual contém um conjunto de arquivos (A_i^{w-1}) de tamanho o.

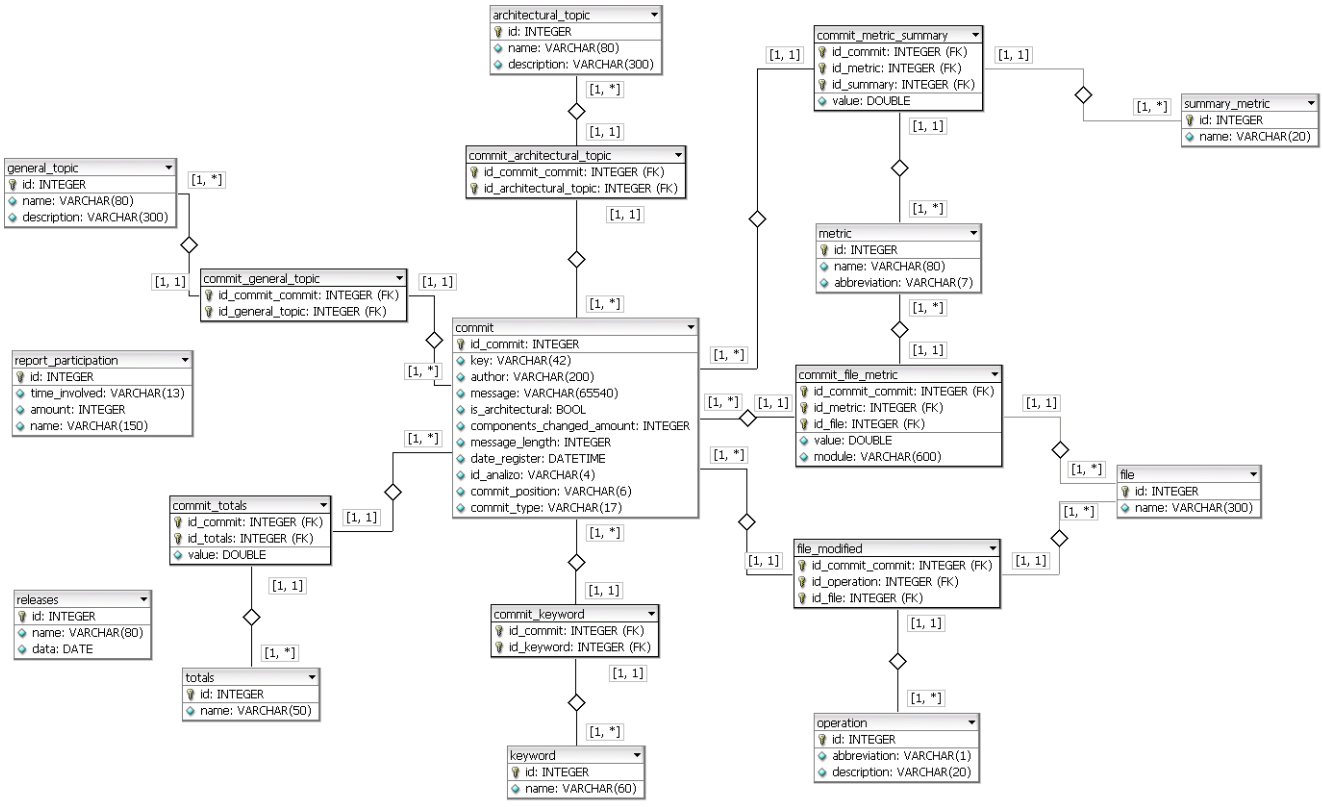


Figure 4.1: Modelo de dados: Banco de dados Metrics.

Existe também um conjunto N finito contido em V ($N \subset V$), tal que para todo N_w existe um V_x correspondente. O conjunto N contém um conjunto de versões do projeto KDELibs, onde não houve modificação na arquitetura e que foram escolhidas, aleatoriamente, e cada versão N_w é composta por um conjunto finito de arquivos (N_i^w) de tamanho p. Para cada N_w, existe um conjunto N_{w-1} (que também possui correspondente em V) que contém um conjunto de arquivos (N_i^{w-1}) de tamanho q.

Além disso, existem dois conjuntos: ParArquitetural P_a, formado pela intersecção do conjunto A_i^w e o conjunto A_i^{w-1} e outro conjunto ParNaoArquitetural P_{na} formado pela intersecção do conjunto N_i^w e o conjunto N_i^{w-1}.

Por fim, definiram-se dois conjuntos Deltas. O primeiro D_a, é formado por um conjunto de d_{a_i}, tal que, para cada arquivo a_i, presente no conjunto A_i^w e seu par b_i no conjunto A_i^{w-1}, existe um d_{a_i} que é igual a diferença do valor da métrica M calculada para o par de arquivos na versão A_w e A_{w-1} para todos os arquivos a_i com correspondência em b_i. O segundo D_{na}, é formado por um conjunto de d_{g_i}, tal que, para cada arquivo g_i, presente no conjunto N_i^w e seu par h_i no conjunto N_i^{w-1}, existe um d_{g_i} que é igual a diferença do valor da métrica M calculada para o par de arquivos na versão N_w e N_{w-1}

para todos os arquivos g_i com correspondência em h_i. A Figura 4.2 ilustra o conteúdo desses conjuntos. As equações abaixo formalizam a formação desse conjunto. Assim, seja:

$$\begin{cases} A_i^w = & \{(a_1, a_1^w), ..., (a_{n_i}, a_{n_i}^w)\} \\ A_i^{w-1} = & \{(b_1, b_1^{w-1}), ..., (b_{m_i}, b_{m_i}^{w-1})\} \\ N_i^w = & \{(g_1, g_1^w), ..., (g_{p_i}, g_{p_i}^w)\} \\ N_i^{w-1} = & \{(h_1, h_1^{w-1}), ..., (h_{q_i}, h_{q_i}^{w-1})\} \end{cases}$$

para i =1,..,232.

Considerando a métrica M, e para i =1,..,232 define-se

$$A_i^x \cap A_i^{x-1} = \{(a, |M(a^x) - M(b^{x-1})|) : \exists j \in \{1, .., m_i\} \cap \{1, .., o_i\} \text{ tal que } a_j = b_j\}$$

e

$$N_i^w \cap N_i^{w-1} = \{(g, |M(g^w) - M(h^{w-1})|) : \exists j \in \{1, .., p_i\} \cap \{1, .., q_i\} \text{ tal que } g_j = h_j\}$$

para i =1,..,232.

Agora, para i =1,..,232 denote-se que com c_{a_i} o número de elementos do conjunto $A_i^w \cap A_i^{w-1}$ e c_{n_i} o número de elementos do conjunto $N_i^w \cap N_i^{w-1}$. Assim pode-se calcular para i =1,..,232,

$$\overline{D_{A_i}} = \frac{\sum_{k=1}^{c_{a_i}} M(a_k^x) - M(a_k^{x-1})}{c_{a_i}}$$

e

$$\overline{D_{N_i}} = \frac{\sum_{k=1}^{c_{n_i}} M(a_k^x) - M(a_k^{x-1})}{c_{n_i}}$$

Sendo o primeiro conjunto formado pelo delta médio dos arquivos (d_{A_i}) de uma determinada métrica M, a partir de *commits* relacionados às mudanças arquiteturais para cada versão A_x (D_{N_i}), e; o segundo formado pelo delta médio dos arquivos de uma determinada métrica M a partir de *commits* sem modificações na arquitetura para cada métrica N_y (D_{a_i}). O Pseudocódigo 4.1 (partes 1 e 2) ilustra o processo de formação desses conjuntos e o processo de coleta das métricas.

Neste pseudocódigo - escrito numa linguagem Java-*like* - são apresentados 9 métodos abrigados na classe `AutomaticMethodExtraction` que procedem a formação dos conjuntos supramencionados. Essa classe contém os seguintes atributos: lista de *commits* do projeto, lista de chaves identificadoras de *commits* arquiteturais e não-arquiteturais, lista de *commits* arquiteturais e não-arquiteturais, lista de valores de métricas para os *commits* arquiteturais e não-arquiteturais e de seus respectivos *commits* anteriores. Esses métodos proveem desde o preenchimento dos conjuntos de *commits* selecionados (A_i^w, A_i^{w-1}, N_i^w, N_i^{w-1}), até a invocação da ferramenta de cálculo dos valores das métricas para cada *commit* e de seus respectivos *commits* anteriores.

Pseudocodigo 4.1: Formação dos conjuntos de versões analisadas e cálculo de métricas (parte 1).

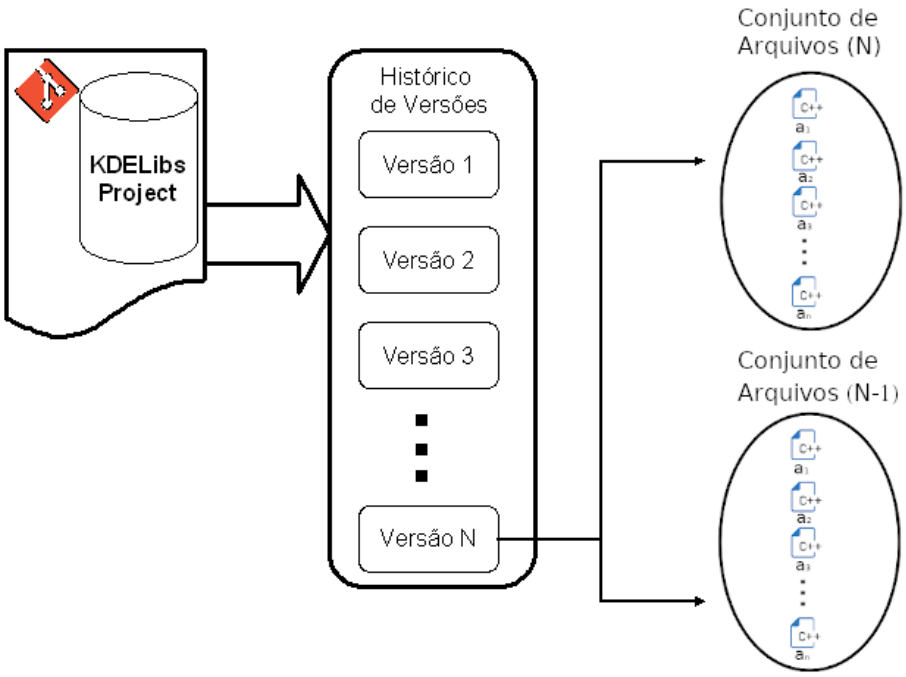


Figure 4.2: Esquema explicativo: Formação dos conjuntos de versões (com alterações arquiteturais e sem alterações arquiteturais).

```
1  public class AutomaticMethodExtraction{
2  /** Campos */
3  // Lista de todos os commits do projeto KDELibs
4  private List<Version> V;
5  // Lista de chaves (Git) dos commits arquiteturais
6  private List<Key> lka;
7  // Lista de chaves (Git) dos commits nao arquiteturais
8  private List<Key> lkna;
9  // Lista de commits Arquiteturais do KDELibs
10 private List<Version> A;
11 // Lista de commits Nao-arquiteturais selecionados do KDELibs
12 private List<Version> N;
13 // Lista de Metricas coletadas para commits arquiteturais
14 private List<Metric m, double value> lma;
15 // Lista de Metricas coletadas para commits nao-arquiteturais
16 private List<Metric m, double value> lmna;
17 /* Lista de Metricas coletadas para os commits anteriores aos
18    commits arquiteturais */
19 private List<Metric m, double value> lma_;
20 /* Lista de Metricas coletadas para os commits anteriores aos
21    commits nao-arquiteturais */
22 private List<Metric m, double value> lmna_;
```

Pseudocodigo 4.1: Formação dos conjuntos de versões analisadas e calculo de metricas (parte 2).

```
23
24 /** Metodos */
25
26 // Corpo principal do codigo
27 public static void main(String args[]){
28         AutomaticMethodExtraction ame = new AutomaticMethodExtraction();
29         ame.putArchitecturalCommitSet();
30         ame.putNonArchitecturalCommitSet();
31         ame.extractAllMetrics();
32 }
33
34 public AutomaticMethodExtraction(){
35         this.V = Git.loadVersions();
36         loadArchitecturalKeys();
37         loadNonArchitecturalKeys();
38         lma = new List<Metric>();
39         lmna = new List<Metric>();
40 }
41
42 // Carrega as chaves dos commits arquiteturais do KDELibs
43 public void loadAarchitecturalKeys(){
44         this.lka = new List<Key>();
45         String[] keys = (String []) File.readLines(
46                                 "architectural_keys.txt");
47         for(String i:keys)
48                 this.lka.add(i);
49 }
50 // Carrega as chaves dos commits arquiteturais do KDELibs
51 public void loadNonAarchitecturalKeys(){
52         this.lkna = new List<Key>();
53         String[] keys = (String []) File.readLines(
54                                 "non_architectural_keys.txt");
55         for(String i:keys)
56                 this.lkna.add(new Key(i));
57 }
58
59 /** Forma o conjunto de commits arquiteturais do KDELibs a partir
60     da lista de chaves disponivel */
61 public void putArchitecturalCommitSet(){
62         this.A = new List<Version>();
63         for(Key k:lka)
64                 A.add(Git.rollbackVersion(k));
65 }
66
67 /** Forma o conjunto de commits nao-arquiteturais do KDELibs
68     a partir da lista de chaves disponivel */
```

Pseudocodigo 4.1: Formação dos conjuntos de versões analisadas e calculo de metricas (parte 3).

```
69 public void putNonArchitecturalCommitSet(){
70         this.N = new List<Version>();
71         for(Key k:lkna)
72                 N.add(Git.rollbackVersion(k));
73 }
74
75 // Extrai todas as metricas
76 public void extractAllMetrics(){
77         extractMetricsFromArchitecturalCommits();
78         extractMetricsFromNonArchitecturalCommits();
79 }
80
81 // Extrai as metricas do conjunto de commits arquiteturais
82 public void extractMetricsFromArchitecturalCommits(){
83   for(Verson i:A){
84           lma.add(Analizo.extractMetrics(i));
85           lma_.add(Analizo.extractMetrics(
86                           V.get(getBeforeVersionKey(i))));}
87 }
88
89 // Extrai as metricas do conjunto de commits nao-arquiteturais
90 public void extractMetricsFromNonArchitecturalCommits(){
91         for(Verson i:N){
92                 lmna.add(Analizo.extractMetrics(i));
93                 lmna_.add(Analizo.extractMetrics(
94                                 V.get(getBeforeVersionKey(i))));}
95 }
96 // Retorna o commit anterior a partir de uma chave passada como parametro
97 public Version getBeforeVersionKey(Key k){
98         return getVersion(Git.getBefore(k).getKey());}
99 }
```

4.3.3 Análise dos Dados Obtidos

Para analisar os dados obtidos, formou-se, para cada métrica, o conjunto de dados M obedecendo o descrito nos conjuntos ParArquitetural e ParNaoArquitetural. A partir dessa organização, para cada *commit*, calculou-se a média dos deltas obtidos para cada arquivo, e a média desses deltas para cada *commit* foi assumido como sendo a variação média de cada versão. Em seguida, comparou-se o total de versões onde a média foi <0 (variação negativa: redução), >0 (variação positiva: aumento) e $=0$ (manutenção dos valores anteriores).

A partir desses dados, através do pacote estatístico R[12], investigou-se o comportamento de cada métrica a partir dos valores obtidos, calculando os valores de média, mediana, valor máximo, valor mínimo e amplitude para os conjuntos A_x (D_{N_i}) e N_y (D_{a_i}).

Para testar a significância estatística das variações dos valores das métricas, foi utilizado

[12]https://www.r-project.org/

o teste U de Mann-Whitney-Wilcoxon (MANN; WHITNEY, 1947; WILCOXON, 1992) sobre o conjunto de deltas obtidos para cada métrica sobre os *commits* arquiteturais e comparando-os com o conjunto de deltas obtidos sobre os *commits* não-arquiteturais. Esses deltas foram expressos em *boxplots*, na razão de um para cada métrica analisada. A quantidade de *commits* onde cada valor de métrica variou (positiva ou negativamente) ou se mateve foi apresentada em *mosaic plots*. Para testar a significância estatística do comportamento predominante (variação ou manutenção de valores) na comparação entre *commits* arquiteturais e *commits* não arquiteturais, foi utilizado o teste Qui-Quadrado (PEARSON, 1900). Foram destacados os *commits* onde as variações de cada métrica registraram valores diferenciados perante os demais.

Para cada conjunto de métrica, foram apresentados os tópicos arquiteturais (MOTTA; SOUZA; SANT'ANNA, 2018) que tiveram mais *commits* com a tendência identificada. Em seguida, os resultados obtidos foram comparados com as conclusões de outros estudos que utilizaram essas métricas, caso tenham sido identificados. Por fim, visando diminuir a quantidade de métricas a serem consideradas numa análise de mudanças arquiteturais, foi realizada uma Análise de Componentes Principais (HOTELLING, 1933) com os dados produzidos por este estudo, indicando qual o conjunto de métricas deve ser avaliado prioritariamente.

Na seção de resultados, são apresentados os resultados das análises realizadas para cada métrica coletada neste estudo.

4.4 RESULTADOS

Foram submetidas, sequencialmente, cada versão selecionada para participação do estudo, à execução do Analizo, resultando na extração das métricas, conforme descrito na Seção 4.3 e obteveram-se os seguintes resultados: Foram gerados 108.857.680 valores de métricas para 10.628 arquivos de código-fonte analisados (sendo 6.803.605 para cada métrica) ao longo da evolução do projeto. Ressalte-se que o número de arquivos analisados por versão é variável (4.238, na versão mais recente – em 25/02/2016 – e 3.077, na versão mais antiga – em 19/08/2006). Os conjuntos de valores de métricas obtidos, foram organizados, conforme a explicação disponível na Subseção 4.3.2.4. A Tabela 4.2 apresenta os valores médios de cada métrica analisada em 4 versões do projeto. Nas próximas subseções, serão apresentados os dados obtidos para cada subquestão de pesquisa.

4.4.1 Alterações arquiteturais em um projeto produzem maiores variações no tamanho do código do projeto que outros tipos de alterações?

A **RQ2.1** questiona se as mudanças arquiteturais produzem maiores alterações no tamanho do código do projeto que outros tipos de alterações. Para responder a essa pergunta, foram analisadas as métricas de tamanho definidas na Seção 4.3.2.1. As subseções, a seguir, apresentam os resultados obtidos para cada métrica e, ao final, é apresentada uma discussão acerca do conjunto de métricas de tamanho analisadas.

Table 4.2: Valores médio das métricas coletadas ao longo do código do projeto em 4 diferentes versões do projeto KDELibs.

	Commit **A**[1]	*Commit* **B**[2]	*Commit* **C**[3]	*Commit* **D**[4]
Quantidade de Arquivos analisados na versão	3.066	3.227	4.116	4.238
Quantidade de Módulos analisados na versão	3.850	4.289	5.261	5.345
Tamanho do Código				
Linhas de Código (LOC)	130,28	122,58	120,69	118,02
Média de Linhas de Código por Método (AMLoc)	13,59	12,56	13,22	13,10
Número de Atributos (NOA)	3,35	2,86	2,90	2,71
Número de Métodos (NOM)	9,40	9,53	9,49	9,31
Número Médio de Parâmetros (ANPM)	1,10	1,07	1,04	1,03
Tamanho do Maior Método (MMLoc)	43,01	38,95	39,50	39,37
Complexidade do Código				
Complexidade Ciclomática Média por Método (ACCM)	2,40	2,28	2,24	2,52
Número de Atributos Públicos (NPA)	0,95	1,14	1,21	1,23
Número de Métodos Públicos (NPM)	8,08	8,33	8,28	8,14
Acoplamento do Código				
Acoplamento entre Objetos (CBO)	5,03	4,13	4,13	4,10
Conexões Aferentes por Classe (ACC)	10,23	9,70	9,76	9,46
Número de Classes Filhas (NOC)	0,44	0,45	0,45	0,44
Profundidade da Árvore de Herança (DiT)	1,28	1,33	1,29	1,25
Resposta por Classe (RFC)	38,86	35,60	35,26	33,47
Coesão do Código				
Complexidade Estrutural (SC)	30,26	26,89	27,71	27,88
Falta de Coesão dos Métodos (LCOM4)	4,48	4,57	4,60	4,68

[1] *Commit* com id 2cadf74d84d086d0147a900adf95251e25490b87 registrado em 19/08/2006.

[2] *Commit* com id d46e71899b4c8bfe2f8a14a015a63231760bd5c6 registrado em 26/06/2008.

[3] *Commit* com id ae80c992876029e9357800e8be1a15135497be40 registrado em 04/06/2010.

[4] *Commit* com id 8bc7d3107ace3b28ad35ac03f7a82dd7c68494cb registrado em 24/10/2016.

4.4.1.1 Média de Linhas de Código por Método (AMLoc)

A Tabela 4.3 apresenta os dados da estatística descritiva, obtida a partir dos conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica AMLoc para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos centésimos, indicando que o tamanho médio dos métodos variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada commit(versão), a variação verificada foi de 174 linhas adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 104 linhas adicionais para *commits* sem mudanças na arquitetura. Essa diferença é estatisticamente significativa (teste U de Mann-Whitney-Wilcoxon, *p-value* de 3,8 e^{-9}).

A Figura 4.3(a) apresenta um *boxplot* ilustrando as variações dos valores da métrica **Média de Linhas de Código por Método** para os *commits* com mudanças arquiteturais em comparação aos *commits* com mudanças de outra natureza. Observou-se que, em poucos *commits*, houve mudança acentuada na distribuição de código entre os métodos

Table 4.3: Estatística Descritiva, obtida para as variações da métrica AMLoc para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,047	0,028
Mediana	0,00	0,00
Amplitude	11,314	3,424
Mínimo	-2,384	-1,535
Máximo	8,930	1,889

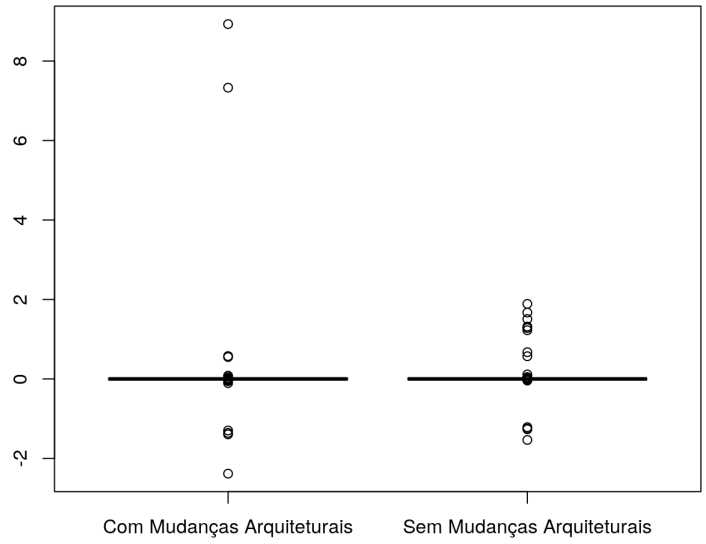
— a exemplo da versão δ[13] (de 18/09/2010), onde se verificou uma variação média de 8,930 e da versão θ[14] (de 08/11/2011), com uma variação média de 7,330. Para os *commits*, que efetuaram outros tipos de modificações no projeto, a variação mais acentuada nessa métrica foi no *commit* ψ[15] (de 14/10/2007) com uma variação de 1,889. Já a parte b da Figura 4.3 mostra que a proporção de *commits* nos quais há variação nos valores da métrica AMLoc é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença é estatisticamente significativa (teste qui-quadrado, *p-value* = 0,024).

Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações positivas** (aumento dos valores da métrica) que os *commits* sem mudanças na arquitetura (96 *commits versus* 81 *commits* sem mudanças na arquitetura) para os valores obtidos para a AMLoc. Nesse conjunto, é preciso destacar que na minoria dos casos (tanto em *commits* com mudanças arquiteturais – 55 versões – quanto nos *commits* sem alterações na arquitetura – 78 versões) a média de distribuição de código entre os métodos não mudou (delta médio = 0). A Figura 4.4 mostra o comparativo entre a quantidade de *commits* onde a métrica AMLoc apresentou variações no projeto KDELibs. Dentre esses *commits*, **não se registrou diferença** estatisticamente **significativa** (teste qui-quadrado, *p-value* = 0,944) na proporção de aumento/redução da métrica entre os dois conjuntos de *commits* (arquiteturais e não-arquiteturais).

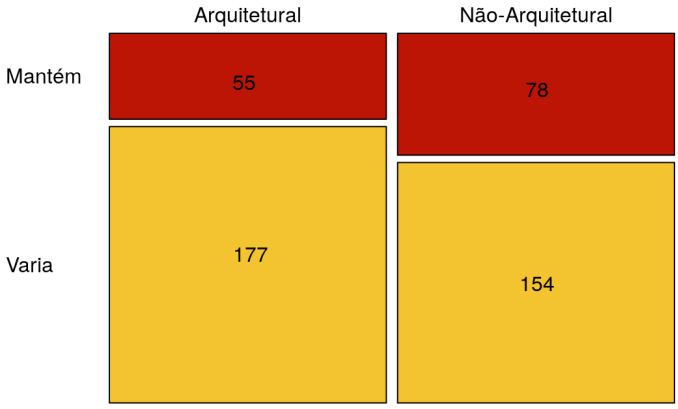
4.4.1.2 Número Médio de Parâmetros (ANPM)

A Tabela 4.4 apresenta os dados da estatística descritiva, obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica ANPM para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos milésimos, indicando que o número médio de parâmetros variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 11 parâmetros adicionais na soma de todos os arquivos modificados

[13]Git key 017506a5b1ef302a109bd88f0c78cc5a9a594dce
[14]Git key 62a05a00a5cf6672fa6b85b450b18a05dd2ed7a8
[15]Git key 128241940065247157c150b21538da636c67dc90



(a) ΔAMLoc – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔAMLoc – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.3: Métrica ΔAMLoc para *commits* com e sem mudanças na arquitetura.

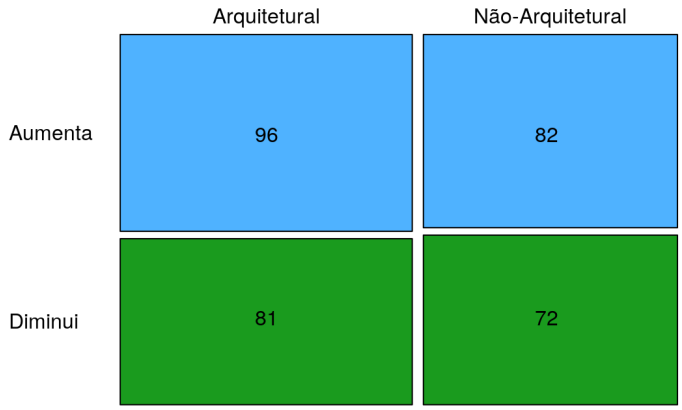


Figure 4.4: ΔAMLoc: Comparação quantidade de *commits* com mudanças arquiteturais e *commits* sem mudanças arquiteturais – aumento x redução.

para *commits* com mudanças na arquitetura e de 3 parâmetros adicionais para *commits* sem mudanças na arquitetura. A diferença observada entre os conjuntos é significativa, estatísticamente (teste U de Mann-Whitney-Wilcoxon, *p-value* de 2,2 e^{-16}).

Table 4.4: Estatística Descritiva obtida para as variações da métrica ANPM para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,003	0,001
Mediana	0,00	0,00
Amplitude	0,835	0,198
Mínimo	-0,116	-0,091
Máximo	0,719	0,107

A Figura 4.5 apresenta o comparativo das variações dos valores da métrica **Número Médio de Parâmetros** (ANPM) de cada versão analisada do projeto KDELibs (parte a) e a proporção de *commits* arquiteturais que sofreram variações no valor dessa métrica (parte b). Foi observado que não houve mudança acentuada no número médio de parâmetros presentes nas assinaturas dos métodos/funções nos *commits* estudados. A variação, mais expressiva, encontrada foi na versão δ (de 18/09/2010), onde se verificou uma variação de 0,719. Na parte b dessa figura, é apresentada a proporção de *commits* nos quais há variação nos valores da métrica ANPM é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença é estatisticamente significativa (teste qui-quadrado, *p-value* = 2,2 e^{-3}).

Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações negativas** (redução dos valores da métrica) que os *commits* sem mudanças na arquitetura (77 *commits versus* 38 *commits* sem mudanças na arquitetura) para os valores obtidos referentes ao número médio de parâmetros por arquivo/módulo. Nessa comparação entre os conjuntos, é preciso destacar que houve um número significativo de *commits* onde o número médio de parâmetros entre os métodos/funções não mudou (delta médio = 0), tanto em *commits* com mudanças arquiteturais quanto nos *commits* sem alterações na arquitetura (96 x 130 respectivamente). A Figura 4.6 mostra a quantidade de *commits*, por versão, onde a métrica ANPM aumentou ou diminuiu no projeto KDELibs, mostrando, lado a lado, *commits* onde houve alteração na arquitetura e *commits* nos quais essa alteração não foi realizada. Pode-se observar que, nos *commits* com mudanças na arquitetura, a proporção de *commits* nos quais há **diminuição** na métrica ANPM, **é maior** dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença é estatisticamente significativa (teste qui-quadrado, *p-value* = 4,7 e^{-3}).

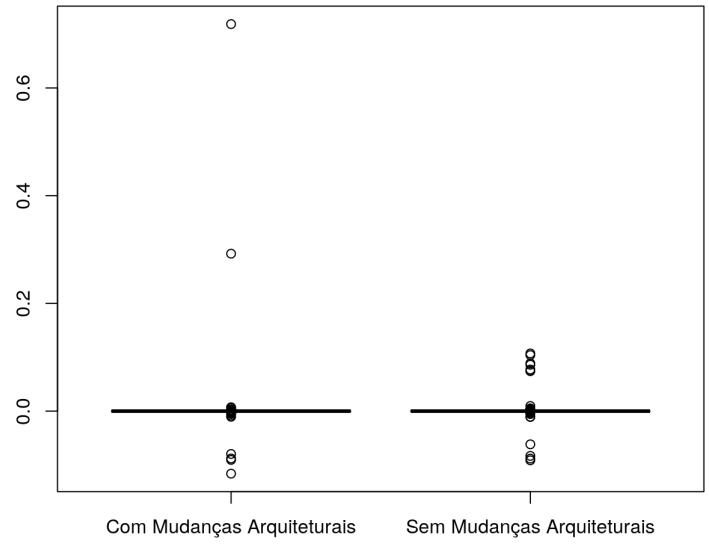
4.4.1.3 Linhas de Código (LOC)

A Tabela 4.5 apresenta os dados da estatística descritiva, obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica LOC para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos décimos, indicando que a média de linhas de código dos arquivos/módulos variou tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 1.617 linhas adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 1.043 linhas adicionais para *commits* sem mudanças na arquitetura. Essa diferença é significativa estatisticamente (teste U de Mann-Whitney-Wilcoxon, *p-value* de 1,7 e^{-11}).

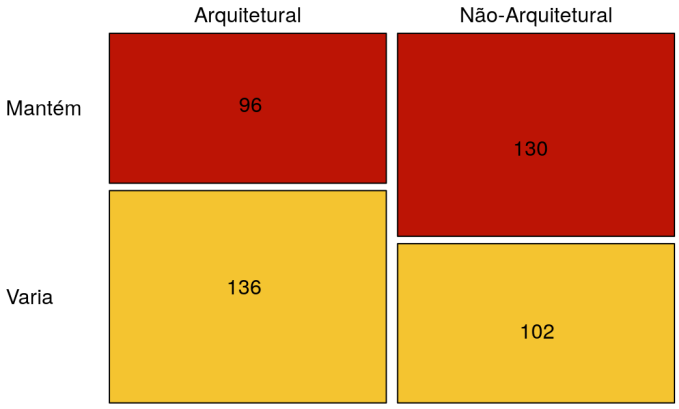
Table 4.5: Estatística Descritiva obtida para as variações da métrica LOC para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,437	0,282
Mediana	0,00	0,00
Amplitude	129,493	49,253
Mínimo	-27,380	-22,950
Máximo	102,113	26,303

A Figura 4.7 apresenta o comparativo dos valores das variações médias para a métrica **Linhas de Código** (LOC), de cada versão analisada do projeto KDELibs em um *boxplot* (parte a) e a proporção de *commits* onde a média dos valores da métrica LOC sofreu alteração ou se mantiveram (parte b). A parte b da Figura 4.7 mostra que a proporção de *commits* nos quais há **variação** nos valores da métrica LOC é maior dentre os *commits*



(a) ΔANPM – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔANPM – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.5: Métrica ΔANPM para *commits* com e sem mudanças na arquitetura.

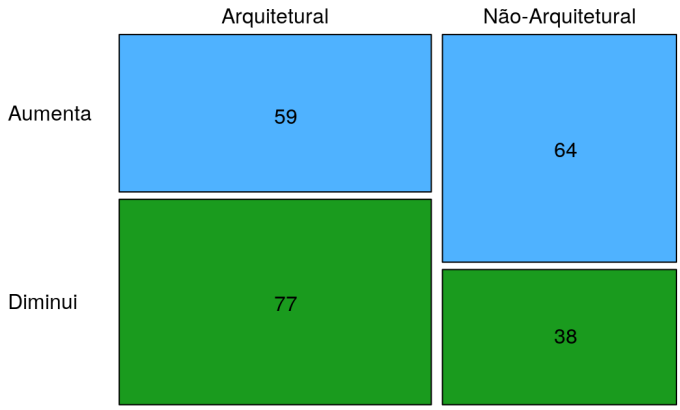


Figure 4.6: ΔANPM: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

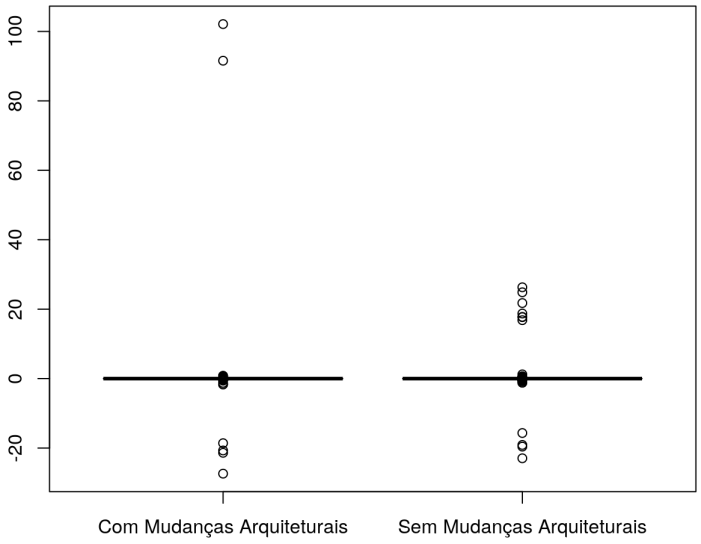
arquiteturais que nos *commits* não-arquiteturais. Essa diferença é, estatisticamente significativa (teste qui-quadrado, *p-value* de 0,017).

Foi observado também que não houve mudança aguda no número de linhas de código nos *commits* estudados. Apenas 2 *commits* com alterações arquiteturais tiveram acréscimos superiores a 10 linhas em média: *commits* δ (de 18/09/2010), onde se verificou um aumento médio de 102 linhas e no *commit* θ (de 08/11/2011), em que se verificou um aumento médio de 91 linhas. Nos *commits* sem alterações na arquitetura, houve 7 *commits* com aumento médio maior que 10 linhas, com destaque para os *commits* ψ (de 14/10/2007), onde se verificou um aumento médio de 26 linhas e λ[16] (de 22/08/2007) com um aumento médio de 25 linhas.

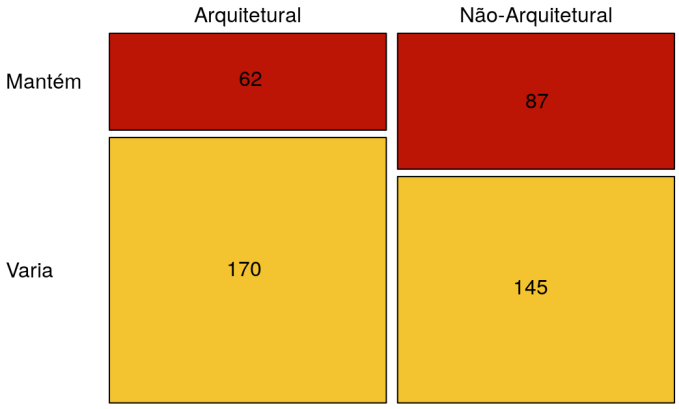
Observou-se que, em geral, os *commits* com modificações arquiteturais, registraram mais **aumentos** (variações positivas) na quantidade de linhas de código que os *commits* sem mudanças na arquitetura (98 *commits versus* 58 *commits* sem mudanças na arquitetura). Enquanto os *commits* sem alteração na arquitetura não registraram variação na quantidade de linhas de código (delta médio = 0) em 87 versões, os *commits* com alteração na arquitetura registram apenas 62 versões com esse comportamento. Por fim, observou-se um número mais recuado de *commits*, no qual a quantidade de linhas de código foi reduzida nos *commits* com alteração arquitetural (72 *commits*), ao passo que, nos *commits* sem alterações na arquitetura, houve uma quantidade maior de reduções (87 *commits*).

A Figura 4.8 mostra a quantidade de *commits*, por versão, em que a métrica LOC aumentou ou diminuiu no projeto KDELibs. Dessa forma, é possível perceber que, enquanto para *commits* com alterações na arquitetura predomina o **aumento** na quantidade

[16]Git key 4b86514252a730bdf139689a8d5b589ce30b37e1



(a) ΔLOC – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔLOC – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.7: Métrica ΔLOC para *commits* com e sem mudanças na arquitetura.

de linhas de código, para os *commits* sem alterações na arquitetura, a maior proporção é de **diminuição** na quantidade de linhas de código (Figura 4.8). Essa variação é, estatisticamente significativa (teste qui-quadrado, *p-value* de 0,003).

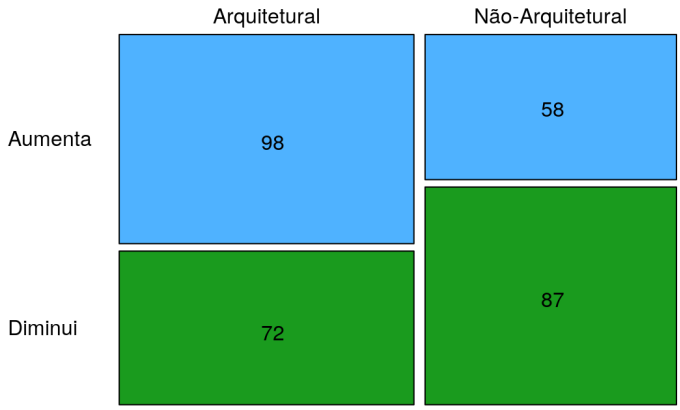
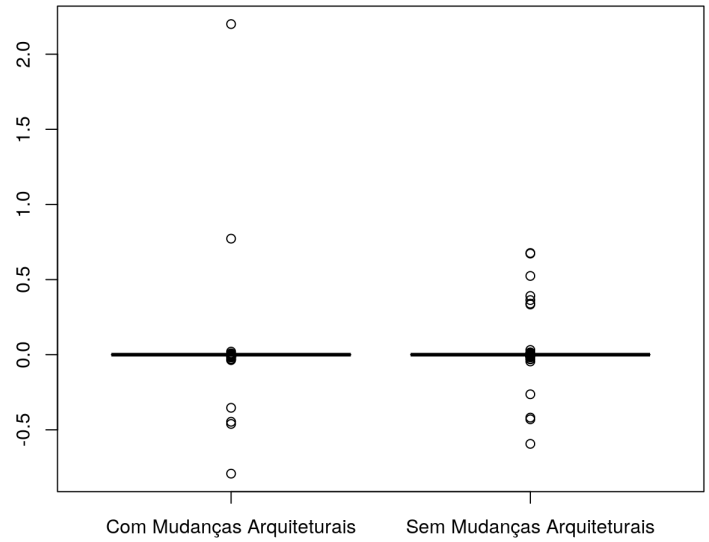


Figure 4.8: ΔLOC: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

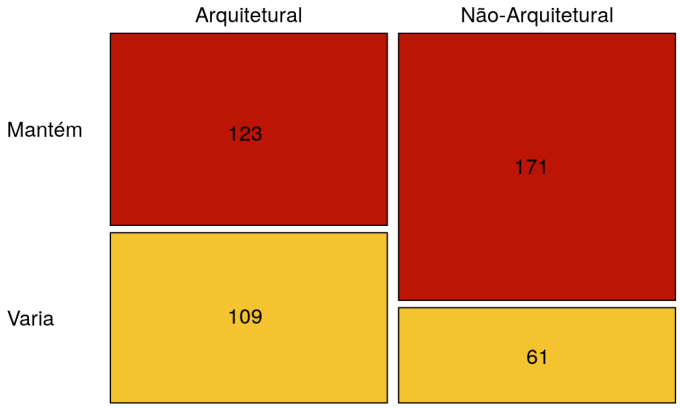
4.4.1.4 Número de Atributos (NOA)

A Tabela 4.6 apresenta os dados da estatística descritiva, obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica NOA para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos milésimos, indicando que o número de atributos variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos, por versão, é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 11 atributos adicionais na soma de todos os arquivos/módulos modificados para *commits* com mudanças na arquitetura e de 22 atributos adicionais para *commits* sem mudanças na arquitetura. A diferença apontada é estatisticamente siginificativa (teste U de Mann-Whitney-Wilcoxon, *p-value* de 2,2 e^{-16}).

A Figura 4.9(a) apresenta o comparativo das variações para os valores da métrica **Número de Atributos** (NOA) para *commits* com mudanças arquiteturais e os *commits* sem mudanças arquiteturais. Foi observado apenas um *commit* com mudança acentuada no número de atributos dos arquivos/módulos analisados — a versão δ (de 18/09/2010), no qual se verificou uma variação média de 2,200. Para os *commits* que entregaram outros tipos de modificações no projeto não foi registrado mudança com valor acima de 1,00. Já



(a) ΔNOA – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔNOA – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.9: Métrica ΔNOA para *commits* com e sem mudanças na arquitetura.

Table 4.6: Estatística Descritiva obtida para as variações da métrica NOA para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,003	0,006
Mediana	0,00	0,00
Amplitude	2,992	1,27
Mínimo	-0,792	-0,593
Máximo	2,200	0,677

a parte b dessa figura apresenta a proporção de *commits* nos quais há variação nos valores da métrica NOA. Ela é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. Essa diferença é significativa estatisticamente (teste qui-quadrado, *p-value* = 5,9 e^{-6}).

Observou-se que, em geral, dentre os *commits* com modificações arquiteturais que apresentaram variações nos valores da métrica NOA, foram negativas (redução dos valores da métrica) que os *commits* sem mudanças na arquitetura (64 *commits versus* 30 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica NOA. Nesse conjunto, é preciso destacar que, **na maioria dos casos**, (tanto em *commits* com mudanças arquiteturais quanto nos *commits* sem alterações na arquitetura - 123 x 171), o **número de atributos não mudou** (delta médio = 0). A Figura 4.10 mostra a quantidade de *commits*, por versão, no qual a métrica NOA aumentou ou diminuiu no projeto KDELibs. A **proporção** de *commits* **com alteração** no valor da métrica foi **minoritária**, e a Figura 4.10 ilustra a constatação de que **não houve diferença**, estatisticamente **significativa** (teste qui-quadrado, *p-value* de 0,299), na proporção de aumento/redução da métrica NOA entre os dois conjuntos de *commits* (arquiteturais e não-arquiteturais).

4.4.1.5 Número de Métodos (NOM)

A Tabela 4.7 apresenta os dados da estatística descritiva, obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica NOM, para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos centésimos para *commits* com mudanças na arquitetura e dos milésimos para *commits* sem mudanças na arquitetura, indicando que o número de métodos variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 111 métodos adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 33 métodos adicionais para *commits* sem mudanças na arquitetura. A diferença é estatisticamente significativa (teste U de Mann-Whitney-Wilcoxon, *p-value* de 3,4 e^{-4}).

Nessa comparação entre os conjuntos, é preciso destacar que houve um número acentuado de *commits* onde o número de métodos não mudou (delta médio = 0), tanto em *commits*

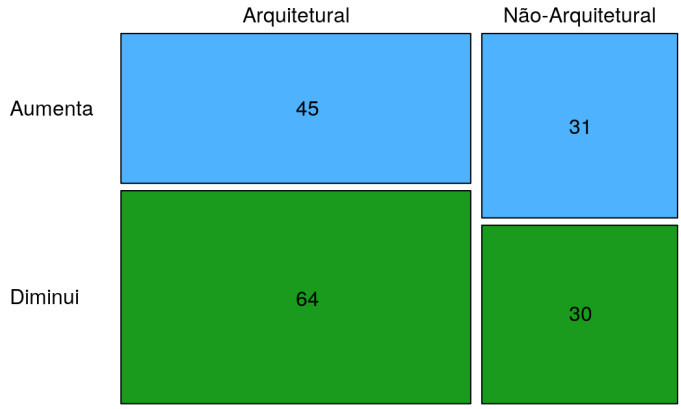
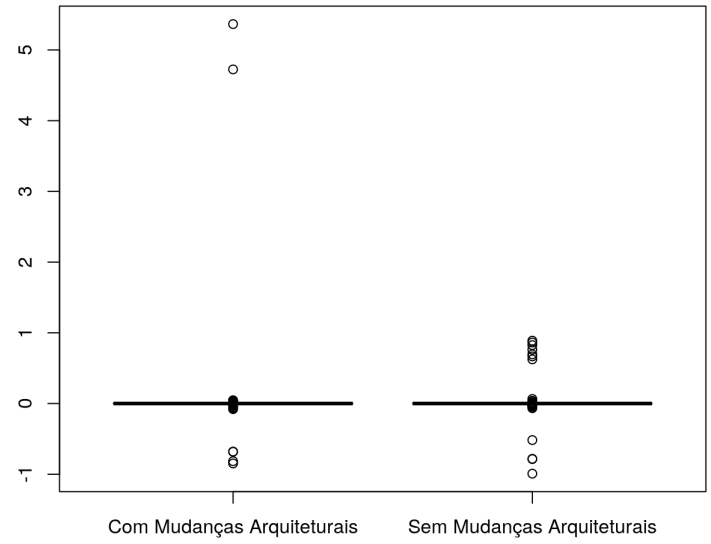


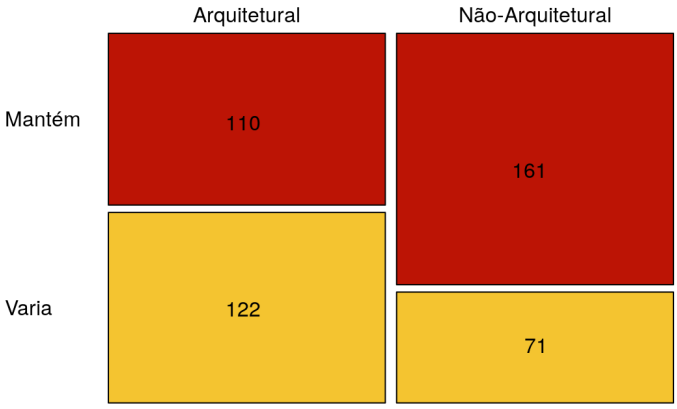
Figure 4.10: ΔNOA: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

Table 4.7: Estatística Descritiva obtida para as variações da métrica NOM para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,030	0,009
Mediana	0,00	0,00
Amplitude	6,213	1,881
Mínimo	-0,847	-0,992
Máximo	5,366	0,889



(a) ΔNOM – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔNOM – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.11: Métrica ΔNOM para *commits* com e sem mudanças na arquitetura.

com mudanças arquiteturais quanto nos *commits* sem alterações na arquitetura (110 x 161 respectivamente). No entanto, a maior parte dos *commits* com modificações na arquitetura apresentou variação na métrica NOM, ao contrário dos *commits* sem alterações na arquitetura. Viu-se também que, em poucos *commits*, houve mudança aguda no número de métodos dos arquivos/módulos das versões analisadas — a exemplo da versão δ (de 18/09/2010), no qual se verificou uma variação média de 5,366 e da versão θ (de 08/11/2011), com uma variação média de 4,725. Não foi registrado nenhum *commit* com modificações de outra natureza, em que a diferença do número de métodos superou 1,00.

A Figura 4.11(a) apresenta o comparativo das variações para os valores da métrica **Número de Métodos** (NOM) para *commits* com mudanças arquiteturais e os *commits* com mudanças de outra natureza. A parte b dessa figura, por sua vez, mostra que a proporção de *commits* nos quais há variação nos valores da métrica NOM é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença é estatisticamente significativa (teste qui-quadrado, *p-value* = 2,5 e^{-6}).

Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações positivas** (aumento do valor da métrica) que os *commits* sem mudanças na arquitetura (65 *commits versus* 28 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica NOM. Já a Figura 4.12 mostra a quantidade de *commits*, por versão, onde a métrica NOM aumentou/diminuiu no projeto KDELibs. Essa proporção de aumento/redução nos valores da métrica NOM entre os dois conjuntos de *commits* **não é**, estatisticamente significativa (teste qui-quadrado, *p-value* de 0,088) – Figura 4.12. No caso dos *commits* com alterações arquiteturais, percebe-se que a quantidade de *commits* com **variação** nos valores da métrica NOM **é maior** que a quantidade de *commits* sem variações nos valores dessa métrica, apesar de ocorrerem, mudanças com valores pequenos, em sua **maioria** trata-se de **acréscimos** no valor dessa métrica. Para os *commits* sem alteração na arquitetura, ocorre o inverso: a maioria dos *commits* analisados não sofre alteração nos valores dessa métrica e os que sofrem mudança, em sua maioria, apresentam reduções nos valores de NOM.

4.4.1.6 Tamanho do Maior Método (MMLoc)

A Tabela 4.8 apresenta os dados da estatística descritiva, obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica MMLOC para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos décimos para *commits* arquiteturais e dos centésimos para os *commits* não-arquiteturais, indicando que o tamanho do maior método variou tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 566 linhas adicionais na soma dos maiores métodos de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 285 linhas adicionais nos maiores métodos para *commits* sem mudanças na arquitetura. Essa diferença é estatisticamente significativa (teste U de Mann-Whitney-Wilcoxon, *p-value* de 2,1 e^{-10}).

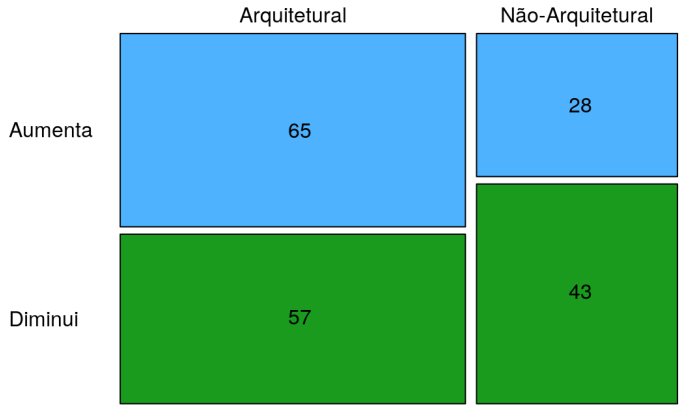
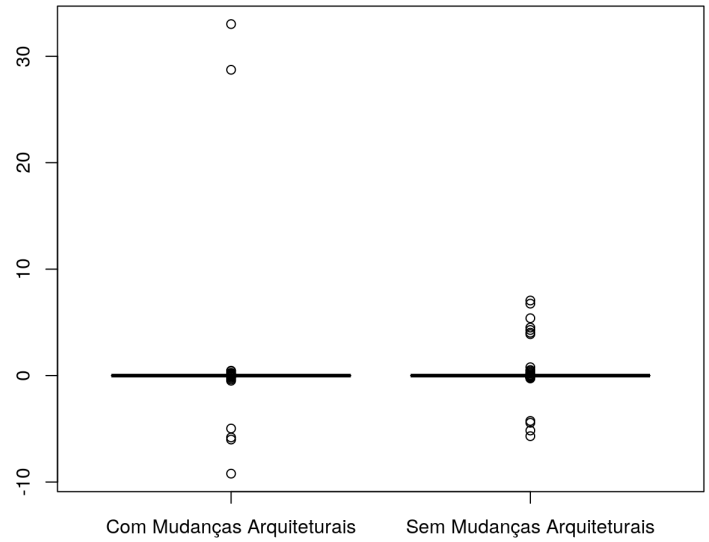


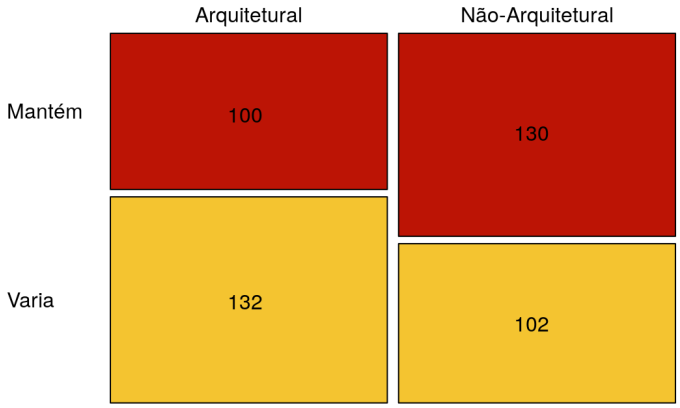
Figure 4.12: ΔNOM: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

Table 4.8: Estatística Descritiva obtida para as variações da métrica MMLOC para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,153	0,077
Mediana	0,00	0,00
Amplitude	42,236	12,746
Mínimo	-9,213	-5,692
Máximo	33,023	7,054



(a) ΔMMLOC – Variação dos valores da métrica por versão: *commits* com mudanças arquiteturais x *commits* sem mudanças arquiteturais



(b) ΔMMLOC – Quantidade de *commits* com alteração no valor da métrica: *commits* com mudanças arquiteturais x *commits* sem mudanças arquiteturais

Figure 4.13: Métrica ΔMMLOC para *commits* com e sem mudanças na arquitetura.

A Figura 4.13(a) apresenta um *boxplot* contendo as variações médias da métrica **Tamanho do Maior Método** para *commits* com mudanças arquiteturais e os *commits* com mudanças de outra natureza. Já a parte b dessa figura apresenta a proporção de *commits* nos quais há variação nos valores da métrica MMLOC é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença registrada é estatisticamente significativa (teste qui-quadrado, *p-value* = 0,007).

Foi observado também que, em poucos *commits*, houve mudança aguda no tamanho do código do maior método/função de cada arquivo/módulo analisado — a exemplo da versão δ (de 18/09/2010), com uma variação média de 33,023 e da versão θ (de 08/11/2011), onde se verificou uma variação média de 28,730. Não foram registradas, em meio as versões sem mudanças arquiteturais, *commits* em que a métrica variou acima de 10 unidades. A versão com maior variação nesse grupo foi a versão ψ(de 14/10/2007), onde se verificou uma variação média de 7,054.

Viu-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações positivas** (aumento dos valores da métrica) que os *commits* sem mudanças na arquitetura (67 *commits versus* 51 *commits* sem mudanças na arquitetura) para os valores da métrica MMLOC. Nesse conjunto, é preciso destacar que, para *commits* com mudanças na arquitetura foram detectados 100, onde não houve alteração no tamanho do maior método de cada arquivo/módulo analisado. Por outro lado, para *commits* em que não houve alterações na arquitetura, viu-se que, em 130 deles, o valor dessa métrica não foi alterado. A Figura 4.14 mostra a quantidade de *commits*, por versão, onde a métrica MMLOC aumentou ou diminuiu no projeto KDELibs. Embora tenha sido registrada a predominância da **variação** (Figura 4.13(b)) nos valores da métrica MMLOC, **não se registrou diferença significativa** estatisticamente (teste qui-quadrado, *p-value* de 1,0) na proporção de aumento/redução dessa métrica entre os dois conjuntos de *commits* (arquiteturais e não-arquiteturais).

4.4.1.7 Discussão sobre as métricas de tamanho coletadas

Ao analisar os valores das métricas de tamanho coletadas, observa-se, usando o teste U de Mann-Whitney-Wilcoxon, significância estatística nos dados obtidos sobre todas as métricas de tamanho. Dessa forma, pode-se considerar que as modificações na arquitetura produzem variações de diferentes intensidades de outros tipos de alterações no projeto a partir da avaliação dos valores de métricas de tamanho. Analisando a Tabela 4.9, pode-se observar que, nos *commits* estudados, a tendência das versões onde houve mudança na arquitetura é de variação em relação às respectivas versões anteriores (a exceção foi a métrica **Número de Atributos**).

As Figuras 4.16, 4.17 e 4.18 apresentam um conjunto de *mosaic plots*, apresentando a comparação individual para cada métrica de tamanho avaliada. Esses resultados mostram que, em *commits* onde a arquitetura do projeto é modificada, considerando o projeto analisado, o tamanho do código aumenta. Trata-se de um resultado inesperado, uma vez que a mudança na arquitetura, de um modo geral, poderia racionalizar o tamanho do código. Lehman et al. (1997) escreveram sobre a relação direta entre a manutenção da satisfação do usuário e o aumento das funcionalidades disponíveis (refletidas no tamanho

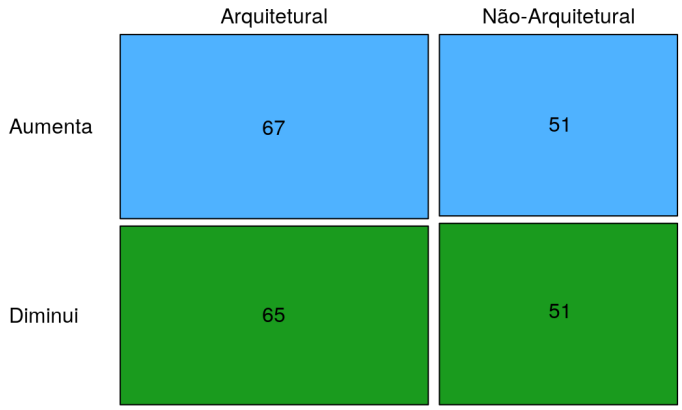


Figure 4.14: ΔMMLOC: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

do código). Por outro lado, para as mudanças sem alterações na arquitetura, dada a natureza diversa dessas mudanças, não se formaram expectativas. Registre-se as exceções para o Número Médio de Parâmetros - que diminuiu - e para o Número de Métodos - que se manteve. As proporções obtidas (aumento/diminuição/manutenção) para os valores de cada métrica de tamanho analisada são, estatísticamente, significativas. A Tabela 4.9 também apresenta os *p-values* obtidos pelo teste qui-quadrado para os valores de cada métrica apresentada nas Figuras 4.16, 4.17 e 4.18.

Table 4.9: Resumo dos testes estatísticos realizados com os valores das métricas de Tamanho obtidos frente ao projeto KDELibs considerando todos os arquivos de cada versão, comparando *commits* com modificações arquiteturais (CCMA) e *commits* sem mudanças arquiteturais (CSMA).

Métrica	**Mann-Whitney-Wilcoxon**			**Teste Qui-Quadrado**	
	P-value	**Significância Estatística**	**Tendência Verificada**	***P-value***	**Significância Estatística**
AMLOC	$3,8e^{-9}$	SIM	Variação (Aumento)	$0,024$	SIM
ANPM	$2,2e^{-16}$	SIM	Variação (Redução)	$2,2e^{-3}$	SIM
LOC	$1,7e^{-11}$	SIM	Variação (Aumento)	$0,017$	SIM
NOA	$2,2e^{-16}$	SIM	Manutenção	$5,9e^{-6}$	SIM
NOM	$3,4e^{-4}$	SIM	Variação (Aumento)	$2,5e^{-6}$	SIM
MMLOC	$2,1e^{-10}$	SIM	Variação (Aumento)	$0,007$	SIM

O capítulo anterior deste estudo definiu quinze diferentes tópicos arquiteturais. Dessa forma, analisar, individualmente, cada um desses tópicos perante os resultados obtidos

para cada métrica, contribui para a explicação dos resultados obtidos e para a construção de inferências. Assim, os tópicos arquiteturais que sobressaíram para cada métrica foram:

- AMLoc: A maior parte dos *commits* arquiteturais, que registraram **aumento** de seus valores, se referiam aos tópicos (XI) Relação entre Elementos, (VIII) Mecanismos de Sincronização, (VII) Protocolos de Comunicação e (XIII) Requisitos Não-Funcionais (66 *commits* de um total de 96 - 69%). No grupo de *commits* com tendência majoritária, foram registrados 14 tópicos com esse tipo de comportamento.

- ANPM: A maior parte dos *commits* arquiteturais, que registraram **diminuição** de seus valores, se referiam aos tópicos (XI) Relação entre Elementos, (VIII) Mecanismos de Sincronização e (VI) Estruturas do Sistema (41 *commits* de um total de 77 - 53%). Nesse grupo de *commits*, foram registrados 14 tópicos com esse tipo de comportamento.

- LOC: A maior parte dos *commits* arquiteturais, que registraram **aumento** de seus valores, se referiam aos tópicos (VIII) Mecanismos de Sincronização, (XI) Relação entre Elementos, (VII) Protocolos de Comunicação e (XIII) Requisitos Não-Funcionais (65 de 98 - 66%). O grupo que definiu a tendência majoritária para essa métrica registrou 13 diferentes tópicos arquiteturais.

- NOA: A maior parte dos *commits* arquiteturais, que registraram a **manutenção** de seus valores, se referiam aos tópicos (XI) Relação entre Elementos, (VIII) Mecanismos de Sincronização, (VII) Protocolos de Comunicação e (XIII) Requisitos Não-Funcionais (76 *commits* de um total de 123 - 62%). Nesse grupo de *commits*, foram registrados 14 tópicos com esse tipo de comportamento.

- NOM: A maior parte dos *commits* arquiteturais, que registraram **aumento** de seus valores, se referiam aos tópicos (VIII) Mecanismos de Sincronização, (XI) Relação entre Elementos e (XIII) Requisitos Não-Funcionais (41 *commits* de um total de 65 - 63%). Nesse grupo de *commits*, foram registrados 11 tópicos com esse tipo de comportamento.

- MMLOC: A maior parte dos *commits* arquiteturais, que registraram **aumento** de seus valores, se referiam aos tópicos (XI) Relação entre Elementos e (VIII) Mecanismos de Sincronização (34 *commits* de um total de 67 - 51%). Nesse grupo de *commits*, foram registrados 13 tópicos com esse tipo de comportamento.

Ao observar os tópicos arquiteturais, definidos neste estudo, individualmente, obtiveram-se resultados interessantes. Para a métrica **AMLoc**, os dois tópicos com mais *commits* identificados (Relação entre Elementos e Mecanismos de Sincronização), na maioria das vezes, mantiveram seus valores ou os diminuíram, respectivamente. Os tópicos Requisitos não-funcionais e Protocolos de Comunicação, por sua vez, seguiram a tendência geral dessa métrica. Assim, sugere-se que, em mudanças arquiteturais, onde esses dois tópicos arquiteturais são abordados, o tamanho médio dos métodos aumenta. Não se podem inferir tendências para os demais tópicos. Já a métrica **ANPM** apresentou 1 tópico

"divergente" em relação à tendência geral de diminuição de valores das métricas de tamanho. Quando a mudança se referia aos Elementos Arquiteturais de Conexão, a maioria dos *commits* manteve o valor da métrica.

Para a métrica **LOC**, quatro tópicos formaram maioria no mesmo sentido que a tendência geral (aumento) dos valores dessa métrica: Requisitos não-funcionais, Protocolos de Comunicação, Estruturas Globais de Controle e Organização Fundamental. Os *commits* relacionados aos Elementos Arquiteturais de Processamento e às Motivações, Objetivos e Restrições foram na direção oposta: a maioria dos *commits*, com esses tipos de mudanças, apresentaram redução na quantidade de linhas de código. A análise por tópicos da métrica **NOA** denota que a tendência de manutenção dos valores é seguida por cinco tópicos. A maioria dos *commits* que abordou a Relação entre Elementos, Requisitos não-funcionais, Protocolos de Comunicação, Elementos Arquiteturais de Conexão e Estruturas Globais de Controle não alteraram seus valores, seguindo a tendência verificada para a maioria dos *commits* em geral. Os *commits*, que abordaram a Atribuição de Recursos, por outro lado, tiveram os valores da métrica NOA, em sua maioria, reduzidos.

A métrica **NOM** apresentou apenas dois tópicos que concordaram com a tendência geral de aumento em seus valores: Requisitos não-funcionais e Estruturas Globais de Controle. Por outro lado, a maioria dos *commits*, relacionados aos tópicos Relação entre Elementos, Protocolos de Comunicação, Elementos Arquiteturais de Conexão e Mecanismos de Acesso aos Dados mantiveram os valores dessa métrica inalterados, em sua maioria. A métrica, Tamanho do Maior Método (**MMLOC**), registrou como tendência o aumento de seus valores na maioria dos *commits*. No entanto, a análise por tópico mostrou que os tópicos Relação entre Elementos, Elementos Arquiteturais de Conexão, Mecanismos de Acesso a Dados e Elementos Arquiteturais de Processamento divergiram dessa tendência. Para os três primeiros, a maioria dos valores da métrica se manteve, enquanto para o último tópico os valores diminuíram.

Em resumo, a análise por tópico dos resultados obtidos para cada métrica indicou:

- Os tópicos com mais *commits* registrados (Relacionamento entre Elementos e Mecanismos de Sincronização), não apresentam uma tendência clara em relação aos valores obtidos para as métricas de tamanho. Para os *commits* relacionados ao tópico Mecanismos de Sincronização, o valor majoritário não superou a metade dos casos para nenhuma das 6 métricas de tamanho computadas. Já os *commits* relacionados ao tópico Mecanismos de Sincronização, os valores obtidos para as métricas ANPM, NOA, NOM e MMLOC indicaram a manutenção dos valores, após ocorridas as mudanças e para as demais métricas de tamanho, nenhuma das tendências verificadas superou a metade dos *commits* elencados.

- Houveram tópicos, onde a mudança arquitetural impactou decisivamente em alguma(s) das métricas de tamanho de código. Nesse grupo, pode-se destacar que as modificações relacionadas aos (i) **Protocolos de Comunicação** aumentam o tamanho médio dos métodos e o tamanho do código, e mantém o número de atributos e o número de métodos; (ii) **Requisitos não-funcionais** aumentam o tamanho médio dos métodos, o tamanho do código e o número de métodos, mas mantém o número de atributos; (iii) **Elementos Arquiteturais de Conexão** mantém o número

médio de parâmetros, o número de atributos, o número de métodos e o tamanho do maior método; (iv) **Mecanismos de Acesso à Dados** mantêm o número de atributos, o número de métodos e o tamanho do maior método; (v) **Elementos Arquiteturais de Processamento** diminuem o tamanho do código, o número de atributos, o número de métodos e o tamanho do maior método; (vi) **Estruturas Globais de Controle** aumentam o tamanho do código e o número de métodos e mantém o número de atributos, e; (vii) **Organização Fundamental** aumenta o número de linhas de código.

Ao comparar os resultados obtidos neste estudo com outros que envolveram a relação entre métricas de tamanho de software com esforço de manutenção, bugs e necessidade de *refactoring*, observou-se que: (i) LOC aumenta nas mudanças da arquitetura quando comparadas com outros tipos de mudança, seguindo a mesma lógica da predição de defeitos (JURECZKO; SPINELLIS, 2010) (maior número de linhas de código = maior quantidade de defeitos) e com o grau de atratividade de um projeto de software livre (MEIRELLES et al., 2010) (maior LOC = maior interesse da comunidade em contribuir com o projeto); (ii) para a métrica NOM repete-se o padrão de comportamento para *commits* com mudança na arquitetura quando se compara com outros tipos de mudança, um resultado similar ao obtido nos estudos de (TANG; KAO; CHEN, 1999) e (THWIN; QUAH, 2005). Esses apontam que o aumento dessa métrica é um indicador da presença de bugs, preditora de esforço de manutenção segundo Li e Henry (1993) e Jin e Liu (2010) e deve ser considerada na construção de métricas para arquitetura de software segundo Bengtsson (1998). Não foram encontrados estudos comparáveis para as demais métricas de tamanho analisadas (AMLOC, ANPM, NOA e MMLOC).

Table 4.10: Tabela de Rotação da Análise de Componente Principal para variações nas métricas de tamanho do projeto KDELibs.

Métrica	**PC1**	**PC2**	**PC3**	**PC4**	**PC5**
AMLOC	0.5263845	-0.0080706796	0.1777828	-0.48012311	-0.67877041
ANPM	0.5261550	0.0004281847	0.2403768	-0.36526780	0.72935528
NOA	0.5054479	0.0085338845	0.3360637	0.79069975	-0.07940212
NOM	0.3072459	-0.7108614780	-0.6270266	0.08034396	0.02565982
MMLOC	0.3101655	0.7032338251	-0.6360119	0.06633237	0.01866769

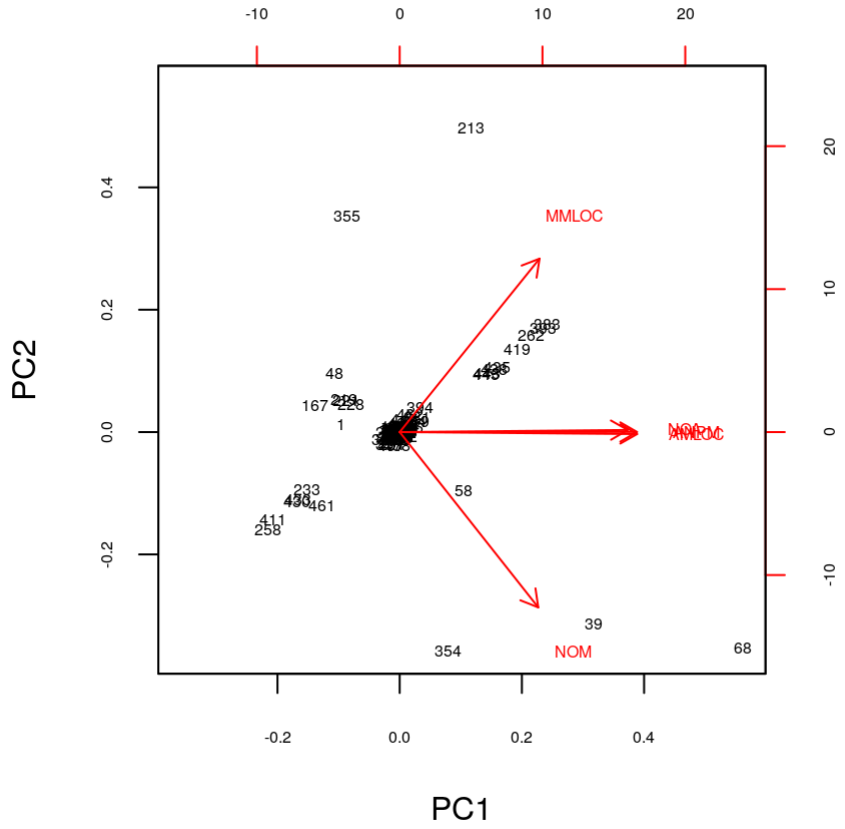


Figure 4.15: Análise de Componente Principal - Variações nos valores das Métricas de Tamanho (normalizadas) por *commit*.

A Análise de Componente Principal (HOTELLING, 1933) realizada, considerando todos os valores das variações das métricas de tamanho, exceto "Linhas de Código", apontou que as métricas AMLoc, ANPM e NOA representam a mesma dimensão de tamanho do projeto, uma vez que os coeficientes que as representam nas componentes *PC1* e *PC2* têm valor muito próximo (ver Tabela 4.10), refletindo na sobreposição de seus vetores conforme ilustra o gráfico *PC1* x *PC2* presente na Figura 4.15.

Em linhas gerais, os coeficientes apresentados na Tabela 4.10 sugerem que em caso de variação nos valores de AMLoc, as variações ocorridas nos valores de ANPM e NOA seguirão a mesma tendência (com a o sinal divergente, contudo). Sobre NOM e MMLOC, os valores observados sugerem que MMLOC e NOM registram variações parecidas, porém com sinais invertidos – quando MMLOC aumenta NOM diminui e vice-versa. A Figura 4.15 apresenta os vetores com as variações das métricas de tamanho consideradas nessa análise, a partir dos coeficientes de *PC1* e *PC2*. Essas dimensões da análise foram adotadas, pois juntas explicam 87,2% da variância encontrada nesse conjunto. Dessa forma, é possível afirmar que, para caracterizar mudanças arquiteturais, é suficiente

observar as variações nos valores das métricas NOM, MMLOC e de uma das três que se revelaram sobrepostas: AMLoc, ANPM e NOA.

4.4.2 Alterações arquiteturais em um projeto produzem maiores variações na complexidade do código do projeto que outros tipos de alterações?

A **RQ2.2** questiona se as mudanças arquiteturais produzem maiores alterações na complexidade do código do projeto que outros tipos de alterações. Para responder a essa pergunta, foram analisadas as métricas de complexidade definidas na Seção 4.3.2.1. As subseções, a seguir, apresentam os resultados obtidos para cada métrica analisada. Ao final, é apresentada uma discussão acerca do conjunto de valores de métricas de complexidade coletadas.

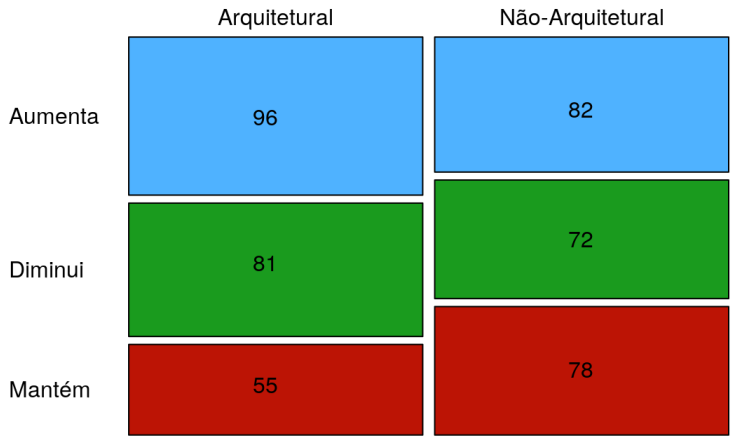
4.4.2.1 Complexidade Ciclomática Média por Método (ACCM)

A Tabela 4.11 apresenta os dados da estatística descritiva, obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica ACCM para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos milésimos, indicando que a complexidade ciclomática média dos métodos variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 22 unidades de complexidade adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 11 unidades de complexidade adicionais para *commits* sem mudanças na arquitetura. Essa diferença é estatisticamente significativa (teste U de Mann-Whitney-Wilcoxon, *p-value* de 2,2 e^{-16}).

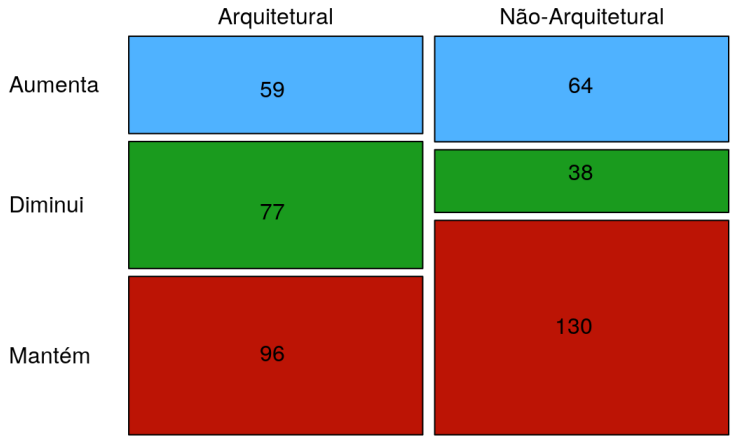
Table 4.11: Estatística Descritiva obtida para as variações da métrica ACCM para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,006	0,003
Mediana	0,00	0,00
Amplitude	1,440	0,941
Mínimo	-0,305	-0,316
Máximo	1,135	0,625

A Figura 4.19(a) apresenta um *boxplot* contendo as variações para os valores da métrica **Complexidade Ciclomática Média por Método** para *commits* com mudanças arquiteturais e os *commits* com mudanças de outra natureza. Não foram observadas mudanças acentuadas na complexidade ciclomática média por método, nas versões analisadas do projeto KDELibs. A maior variação registrada foi de 1,135 na versão δ, a qual efetuou alterações na arquitetura do projeto (de 18/09/2010). A parte b dessa figura, por sua vez, mostra que a proporção de *commits* nos quais há variação nos valores da métrica ACCM

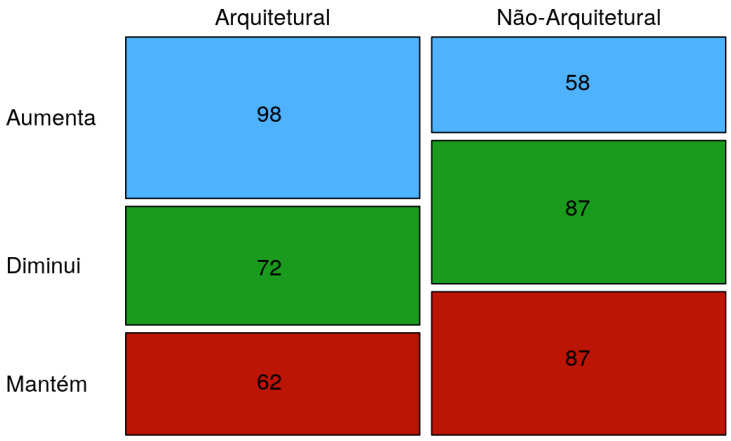


(a) Comparação dos valores da métrica ΔAMLoc entre *commits* com mudanças na arquitetura e *commits* sem mudanças na arquitetura.

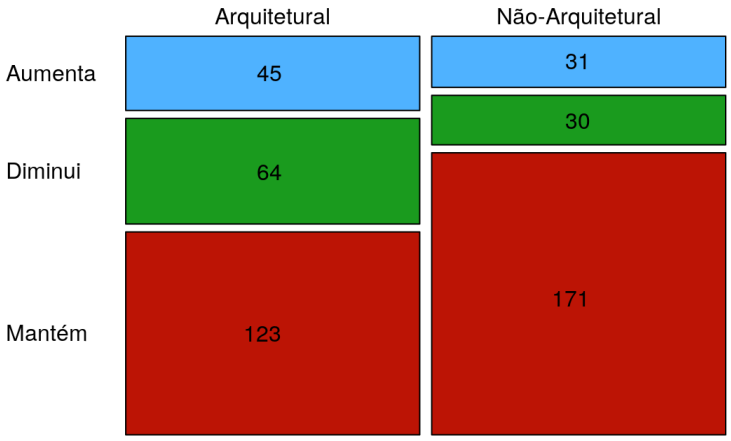


(b) Comparação dos valores da métrica ΔANPM entre *commits* com mudanças na arquitetura e *commits* sem mudanças na arquitetura.

Figure 4.16: Comparação das variações médias para os valores das métricas de tamanho consideradas neste estudo para *commits* com e sem mudanças na arquitetura (parte 1).

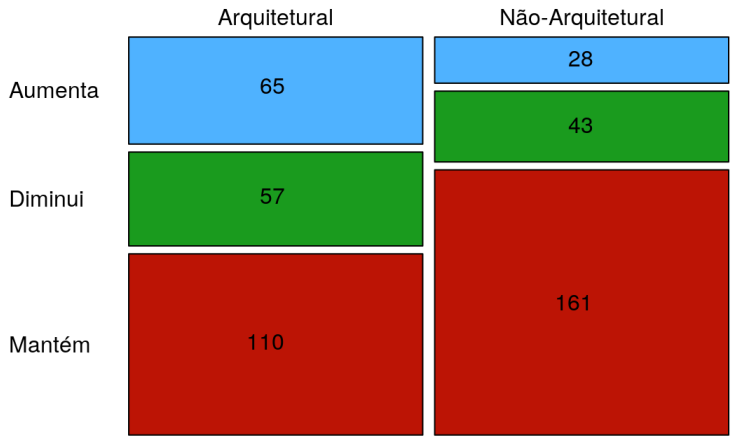


(a) Comparação dos valores da métrica ΔLOC entre *commits* com mudanças na arquitetura e *commits* sem mudanças na arquitetura.

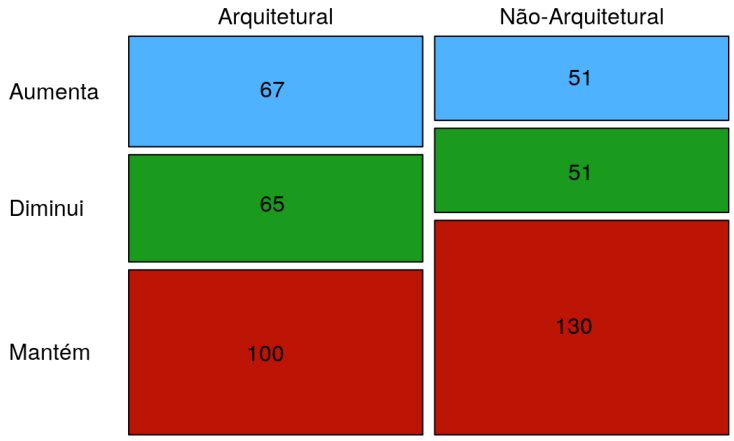


(b) Comparação dos valores da métrica ΔNOA entre *commits* com mudanças na arquitetura e *commits* sem mudanças na arquitetura.

Figure 4.17: Comparação das variações médias para os valores das métricas de tamanho consideradas neste estudo para *commits* com e sem mudanças na arquitetura (parte 2).



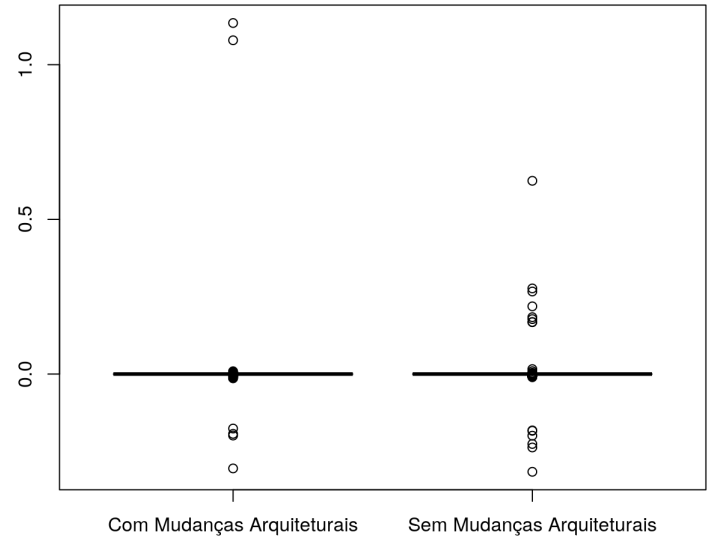
(a) Comparação dos valores da métrica ΔNOM entre *commits* com mudanças na arquitetura e *commits* sem mudanças na arquitetura.



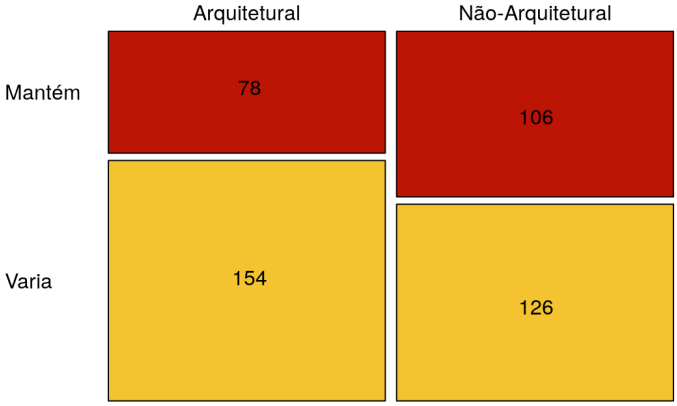
(b) Comparação dos valores da métrica ΔMMLOC entre *commits* com mudanças na arquitetura e *commits* sem mudanças na arquitetura.

Figure 4.18: Comparação das variações médias para os valores das métricas de tamanho consideradas neste estudo para *commits* com e sem mudanças na arquitetura (parte 3).

é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença é estatisticamente significativa (teste qui-quadrado, *p-value* = 0,010).



(a) ΔACCM – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔACCM – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.19: Métrica ΔACCM para *commits* com e sem mudanças na arquitetura.

Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações negativas** (diminuição dos valores da métrica) que os *commits* sem mudanças na arquitetura (78 *commits versus* 72 *commits* sem mudanças na arquitetura)

para os valores obtidos para a métrica ACCM. Nesse conjunto, é preciso destacar que foram registrados 78 *commits* com modificações arquiteturais sem alterações no valor da métrica ACCM e outros 106 *commits* sem modificações arquiteturais (delta médio = 0). A Figura 4.20 mostra a quantidade de *commits*, por versão, em que a métrica ACCM aumentou ou diminuiu no projeto KDELibs. Embora tenha predominado a **variação** dos valores da métrica ACCM (Figura 4.19(b)), **não se registrou diferença** estatisticamente **significativa** (teste qui-quadrado, *p-value* = 0,335) na proporção de aumento/redução dos valores da métrica entre os dois conjuntos de *commits* (arquiteturais e não-arquiteturais).

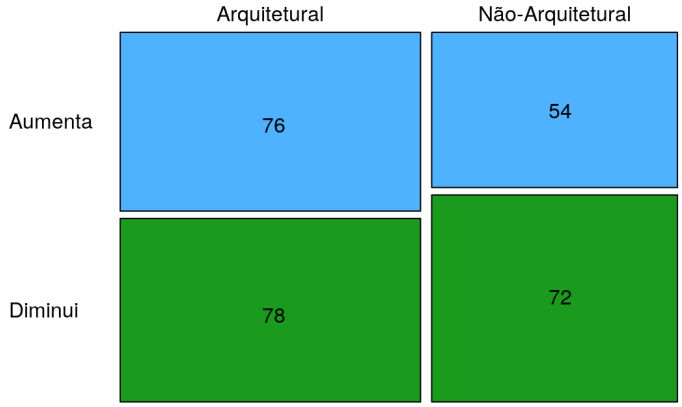


Figure 4.20: ΔACCM: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

4.4.2.2 Número de Atributos Públicos (NPA)

A Tabela 4.12 apresenta os dados da estatística descritiva obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica NPA para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos milésimos, indicando que o número de atributos públicos variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 26 atributos públicos adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 4 atributos públicos a menos para *commits* sem mudanças na arquitetura. Essa é uma diferença estatisticamente significativa (teste U de Mann-Whitney-Wilcoxon, *p-value* de 8,9 e^{-6}).

Table 4.12: Estatística Descritiva obtida para as variações da métrica NPA para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

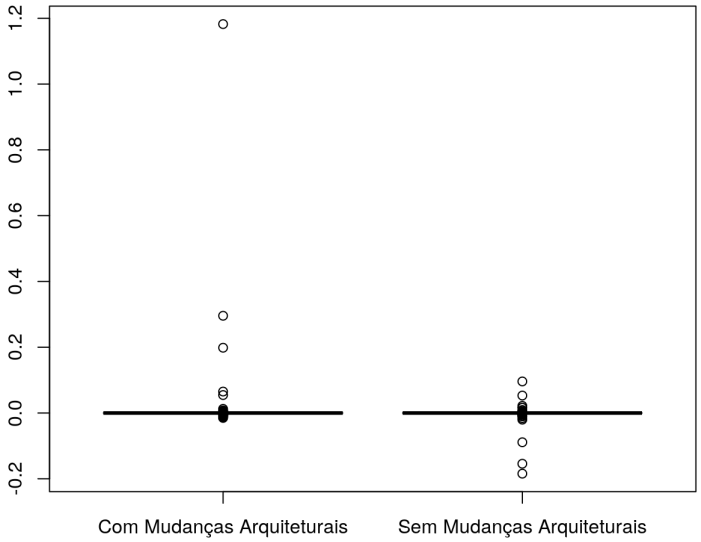
	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,007	-0,001
Mediana	0,00	0,00
Amplitude	1,197	0,280
Mínimo	-0,015	-0,184
Máximo	1,182	0,096

A Figura 4.21(a) apresenta um *boxplot* contendo as variações dos valores para a métrica **Número de Atributos Públicos** em *commits* com mudanças arquiteturais e os *commits* com mudanças de outra natureza. Não foram observadas mudanças significativas na quantidade de atributos públicos registrados nos arquivos/módulos do projeto. A maior variação registrada foi na versão δ (de 18/09/2010), em que se verificou uma variação média de 1,182. Já a parte b dessa figura apresenta a proporção de *commits* nos quais há variação nos valores da métrica NPA. Essa proporção é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença reportada é estatisticamente significativa (teste qui-quadrado, *p-value* = $1{,}1\ e^{-5}$).

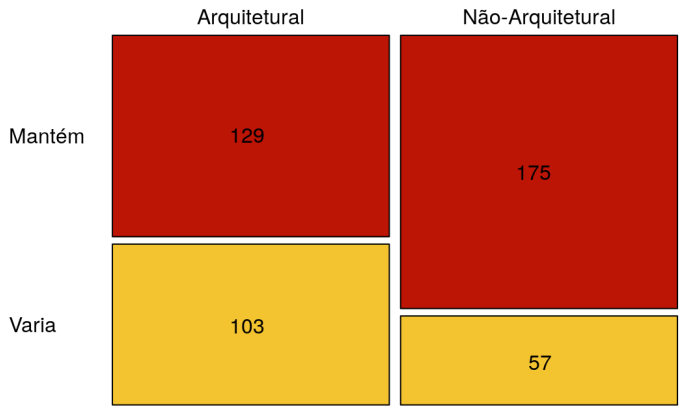
Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações negativas** (redução dos valores da métrica) que os *commits* sem mudanças na arquitetura (56 *commits versus* 26 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica NPA. Nesse conjunto é preciso destacar que na maioria dos casos (tanto em *commits* com mudanças arquiteturais quanto nos *commits* sem alterações na arquitetura - 129 x 175), a quantidade de atributos públicos não mudou (delta médio = 0). A Figura 4.22 mostra a quantidade de *commits*, por versão, na qual a métrica NPA aumentou ou diminuiu no projeto KDELibs. Dessa forma, é possível perceber que, embora em ambos os conjuntos haja a predominância da **manutenção** (Figura 4.21(b)) nos valores obtidos para a métrica estudada, nos *commits* com mudanças arquiteturais foi registrado um número considerável de *commits* com redução no número de atributos públicos, um efeito contrário ao observado para os *commits* sem mudanças arquiteturais. **Essa diferença** de proporções dos dois conjuntos de *commits* (arquiteturais e não-arquiteturais), **não é** estatisticamente **significativa** (teste qui-quadrado, *p-value* = 0,37), no entanto.

4.4.2.3 Número de Métodos Públicos (NPM)

A Tabela 4.13 apresenta os dados da estatística descritiva obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica NPM para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos centésimos para *commits* arquiteturais e dos milésimos para os *commits* não-arquiteturais, indicando que o número de métodos públicos variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por



(a) ΔNPA – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔNPA – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.21: Métrica ΔNPA para *commits* com e sem mudanças na arquitetura.

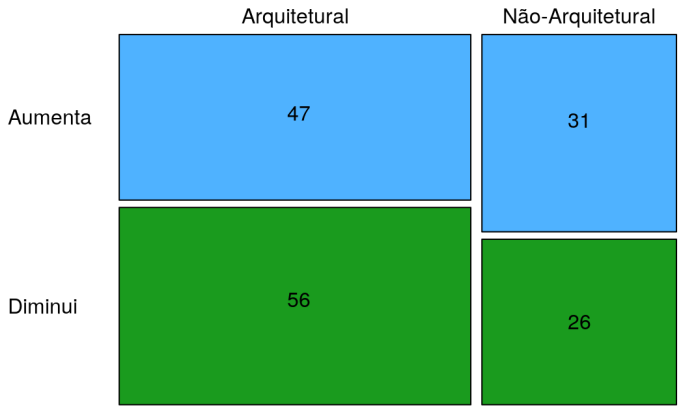


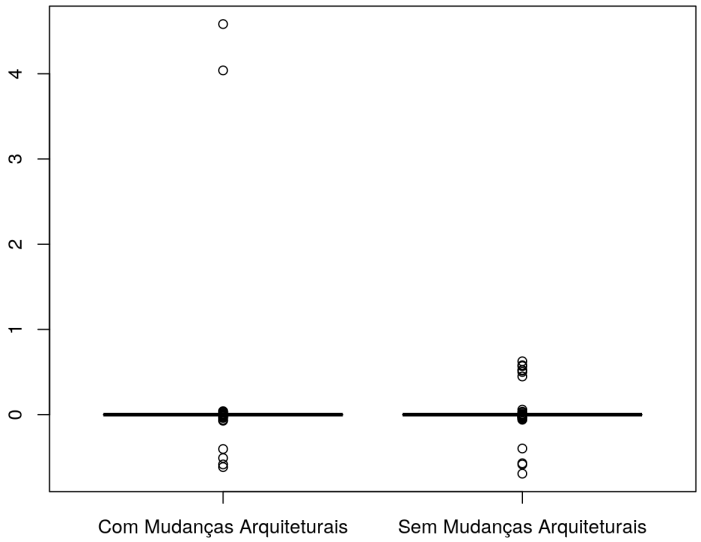
Figure 4.22: ΔNPA: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 104 métodos públicos adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 22 métodos públicos adicionais para *commits* sem mudanças na arquitetura. As variações registradas, no entanto, não são estatisticamente significativas (teste U de Mann-Whitney-Wilcoxon, *p-value* de 0,493).

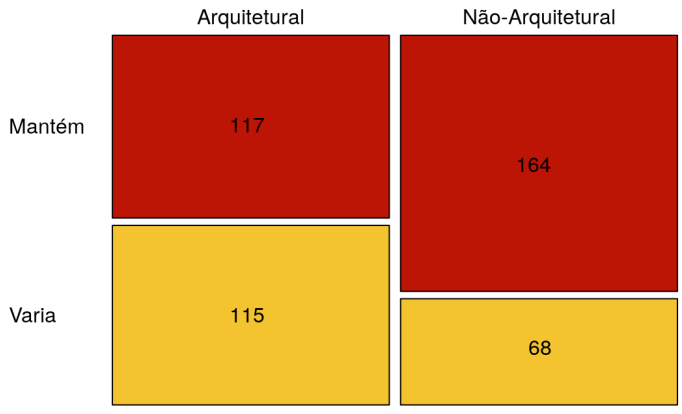
Table 4.13: Estatística Descritiva obtida para as variações da métrica NPM para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,028	0,006
Mediana	0,00	0,00
Amplitude	5,198	1,318
Mínimo	-0,615	-0,692
Máximo	4,583	0,626

A Figura 4.23(a) apresenta um *boxplot*, contendo as variações dos valores obtidos para a métrica **Número de Métodos Públicos** em *commits* com mudanças arquiteturais e os *commits* com mudanças de outra natureza. Foi observado também que, em apenas 2 *commits*, houve mudança acentuada no número de métodos públicos — a exemplo da versão δ (de 18/09/2010), onde se verificou uma variação média de 4,583 e da versão θ (de 08/11/2011), com uma variação média de 4,039. Não foram registradas modificações significativas nos *commits* onde não houve modificações na arquitetura. A parte b dessa figura, por sua vez, apresenta a proporção de *commits* nos quais há variação nos valores



(a) ΔNPM – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔNPM – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.23: Métrica ΔNPM para *commits* com e sem mudanças na arquitetura.

nos valores da métrica NPM. Essa proporção é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença é estatisticamente significativa (teste qui-quadrado, *p-value* = 1,2 e^{-5}).

Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações positivas** (aumento dos valores da métrica) que os *commits* sem mudanças na arquitetura (62 *commits versus* 26 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica NPM. Nesse conjunto é preciso destacar que, na maioria dos casos, (tanto em *commits* com mudanças arquiteturais quanto nos *commits* sem alterações na arquitetura - 117 x 164), o número de métodos publicos não mudou (delta médio = 0). A Figura 4.24 mostra a quantidade de *commits*, por versão, onde a métrica NPM aumentou ou diminuiu no projeto KDELibs. Dessa forma é possível perceber que embora em ambos os conjuntos haja a predominância da **manutenção** (Figura 4.23(b)) dos valores obtidos para a métrica estudada, nos *commits* com mudanças arquiteturais foi registrado um número considerável de *commits* com aumento na média de métodos públicos por *commit*, um efeito divergente ao observado no conjunto de *commits* sem alteração na arquitetura, onde as variações nos valores da métrica NPM ocorreram em número menor. A diferença registrada entre as proporções de aumento/redução dos valores da métrica NPM entre os dois conjuntos é estatisticamente significativa (teste qui-quadrado, *p-value* de 0,05).

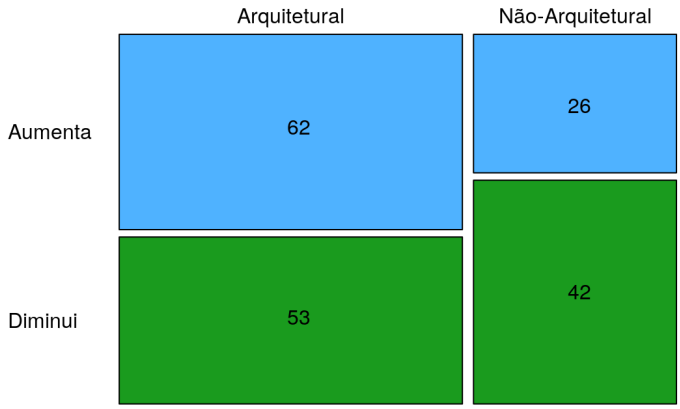


Figure 4.24: ΔNPM: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

4.4.2.4 Discussão sobre as métricas de complexidade coletadas

Ao analisar os valores das métricas de complexidade coletadas, percebe-se, usando o

teste U de Mann-Whitney, significância estatística nos dados obtidos sobre as métricas de complexidade (à exceção da métrica **Número de Métodos Públicos**). Dessa forma, pode-se considerar que as modificações na arquitetura produzem variações diferentes de outros tipos de alterações no projeto a partir da avaliação dos valores de métricas de complexidade. Observando a Tabela 4.14, pode ser visto que não houve um comportamento generalizável. Enquanto a **Complexidade Ciclomática por Método** apresentou um cenário majoritário de variação nos dados, os valores das métricas **Número de Métodos Públicos** e **Número de Atributos Públicos** tiveram a maior parte dos *commits* sem variações em seus valores. Por outro lado, nos *commits* com outros tipos de modificações, todas as métricas de complexidade tiveram predominância de manutenção de seus valores, contrariando a segunda lei de Lehman (LEHMAN et al., 1997), a qual prevê que a complexidade do código de um projeto aumenta à medida que ele evolui.

Table 4.14: Resumo dos testes estatísticos realizados com os valores das métricas de Complexidade obtidos frente ao projeto KDELibs, considerando todos os arquivos de cada versão, comparando *commits* com modificações arquiteturais (CCMA) e *commits* sem mudanças arquiteturais (CSMA).

Métrica	**Mann-Whitney-Wilcoxon**			**Teste Qui-Quadrado**	
	P-value	**Significância Estatística**	**Tendência Verificada**	***P-value***	**Significância Estatística**
ACCM	$2,2e^{-16}$	SIM	Variação (Redução)	$0,010$	SIM
NPA	$8,9e^{-6}$	SIM	Manutenção	$1,1e^{-5}$	SIM
NPM	$0,493$	NÃO	Manutenção	$1,2e^{-5}$	SIM

As Figuras 4.26 e 4.27 apresentam um conjunto de *mosaic plots*, apresentando a comparação individual para cada métrica de complexidade avaliada. Através dessas figuras, é possível visualizar os resultados declarados acima. Trata-se de um resultado esperado, uma vez que as métricas de complexidade analisadas referem-se a detalhes de design, não sendo, portanto, características tratadas no nível de abstração em que se encontra a arquitetura do software. As proporções obtidas (aumento/diminuição/manutenção) para os valores de cada métrica de complexidade analisada são, estatísticamente, significativas. A Tabela 4.14 apresenta os *p-values* obtidos pelo teste qui-quadrado para os valores de cada métrica apresentada nas Figuras 4.26 e 4.27. Surpreende, no entanto, o resultado obtido pela complexidade ciclomática, pois conforme afirmado acima, as modificações arquiteturais não se referem aos detalhes de design ou implementação.

Assim como nas métricas de tamanho, analisar individualmente os resultados obtidos perante cada tópico arquitetural colabora na compreensão dos dados coletados. Dessa forma, pode-se afirmar, frente a esses dados:

- ACCM: A maior parte dos *commits* arquiteturais, que registrou **redução** de seus valores, se referia aos tópicos (VIII) Mecanismos de Sincronização, (XIII) Requisitos Não-Funcionais e (XI) Relação entre Elementos (42 de 78 - 54%). Nesse grupo de *commits*, foram registrados 14 tópicos com esse tipo de comportamento.

- NPA: A maior parte dos *commits* arquiteturais, que registrou a **manutenção** de seus valores, se referia aos tópicos (XI) Relação entre Elementos, (VIII) Mecanismos de Sincronização, (XIII) Requisitos Não-Funcionais e (VII) Protocolos de Comunicação, (79 de 129 - 61%). Nesse grupo de *commits*, foram registrados 14 tópicos com esse tipo de comportamento.

- NPM: A maior parte dos *commits* arquiteturais, que registrou a **manutenção** de seus valores, se referia aos tópicos (XI) Relação entre Elementos, (VII) Protocolos de Comunicação e (VIII) Mecanismos de Sincronização (67 *commits* de um total de 117 - 57%). Nesse grupo de *commits*, foram registrados 14 tópicos com esse tipo de comportamento.

Ao analisar, individualmente, os valores de métrica obtidos para cada tópico arquitetural, é possível assinalar os comportamentos diferentes entre eles. Para a métrica **ACCM**, a tendência verificada é de redução de seus valores em *commits* com alterações na arquitetura. Os tópicos Estruturas do Sistema e Mecanismos de Sincronização acompanharam a tendência majoritária dessa métrica. Por outro lado, os tópicos Requisitos não-funcionais e Estruturas Globais de Controle apresentaram tendência majoritária "divergente" da tendência geral para a métrica ACCM. Enquanto a maioria dos valores obtidos para essa métrica apresentou redução, para esses dois tópicos, a tendência majoritária foi de acréscimo. Tanto para a métrica **NPA** quanto para a métrica **NPM**, a tendência majoritária obtida foi a manutenção dos valores. A maioria dos tópicos seguiu essa tendência. Os tópicos Relação entre Elementos, Mecanismos de Sincronização, Protocolos de Comunicação, Elementos Arquiteturais de Conexão e Mecanismos de Acesso aos Dados apresentaram tendência convergente à tendência geral para ambas as métricas. Apenas o tópico Elementos Arquiteturais de Processamento apresentou tendência divergente (diminuição de valores) para a métrica NPA. Para métrica NPM, apenas o tópico Estruturas Globais de Controle teve tendência divergente da tendência geral verificada (aumento de seus valores).

A Análise de Componente Principal (HOTELLING, 1933) procedida, levando em consideração todos os valores das variações das métricas de complexidade, apontou que as métricas NPM, ACCM representam a mesma dimensão de complexidade do projeto, uma vez que os coeficientes que as representam nas componentes *PC1* e *PC2* têm valor muito próximo (ver Tabela 4.15), refletindo na quase sobreposição de seus vetores conforme mostra o gráfico *PC1* x *PC2* presente na Figura 4.25.

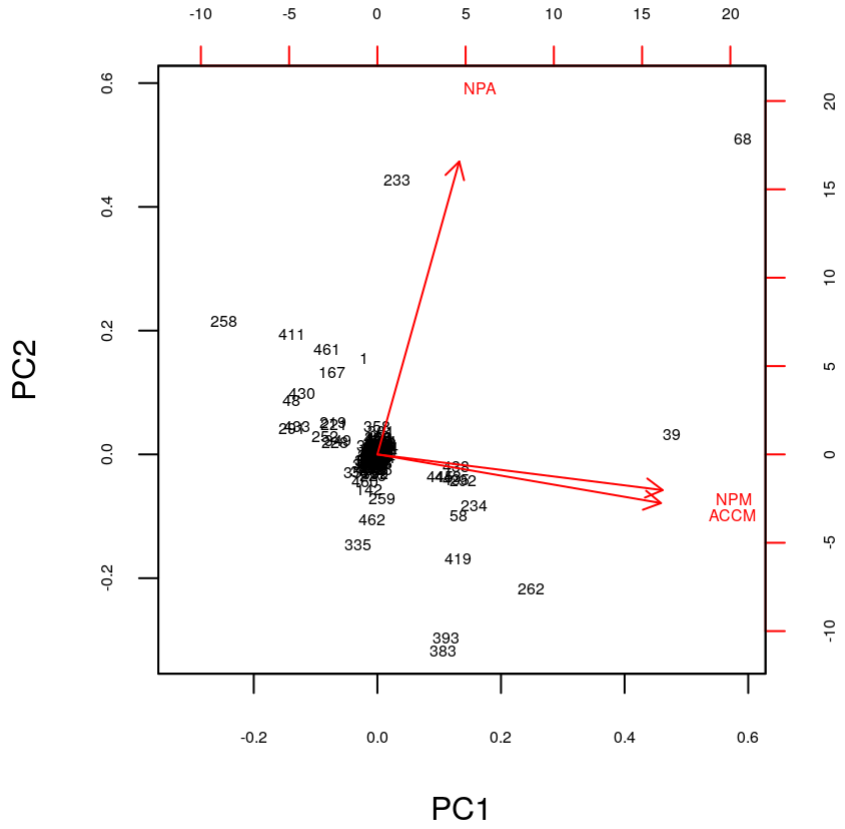


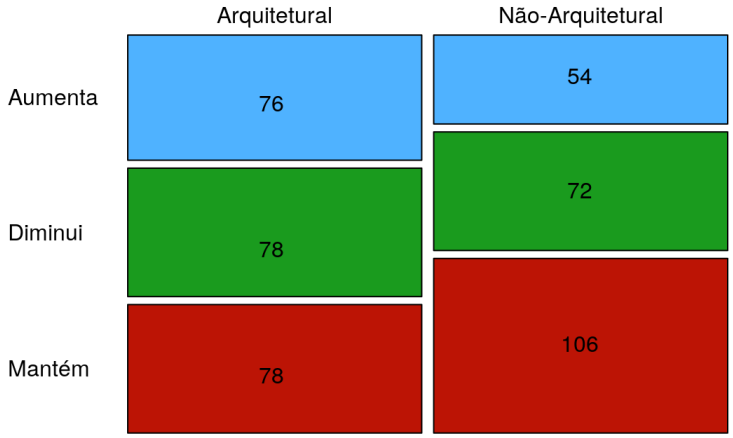
Figure 4.25: Análise de Componente Principal - Variação dos valores das Métricas de Complexidade (normalizadas) por *commit*.

Table 4.15: Tabela de Rotação da Análise de Componente Principal para variações nas métricas de complexidade do projeto KDELibs.

Métrica	**PC1**	**PC2**	**PC3**
ACCM	0.6908200	-0.1627240	-0.7044776
NPM	0.6949539	-0.1194279	0.7090670
NPA	0.1995165	0.9794171	-0.0305825

Em linhas gerais, os coeficientes apresentados na Tabela 4.15 sugerem que em caso de variação nos valores de ACCM, as variações ocorridas nos valores de NPM seguirão a mesma tendência. Sobre NPA, os valores observados sugerem que trata-se de uma métrica de complexidade com tendência divergente em relação às demais. A Figura 4.25, apresenta os vetores com as variações das métricas de complexidade consideradas nessa análise, a partir dos coeficientes de *PC1* e *PC2*. Essas dimensões da análise foram adotadas, pois

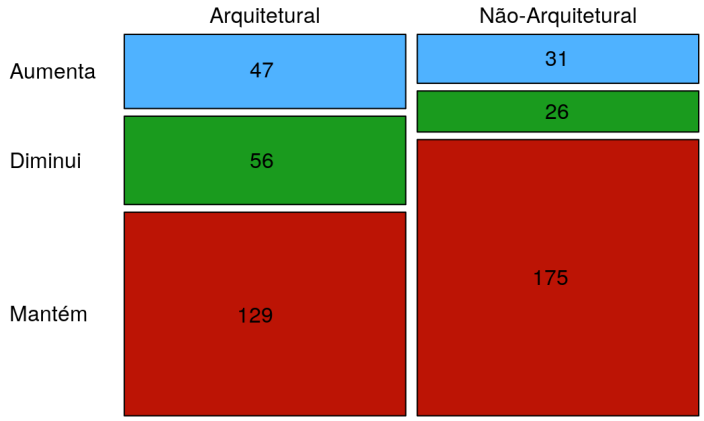
juntas explicam 93,1% da variância encontrada nesse conjunto. Dessa forma, é possível afirmar que, para caracterizar mudanças arquiteturais, é suficiente observar as variações nos valores da métrica NPA e de uma das duas que se revelaram com dimensões de complexidade próximas: ACCM e NPM.



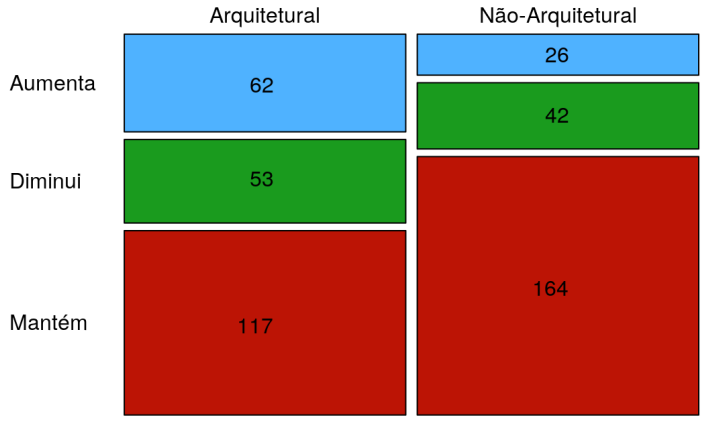
(a) Comparação dos valores da métrica ΔACCM entre *commits* com mudanças na arquitetura e *commits* sem mudanças na arquitetura.

Figure 4.26: Comparação das variações médias para os valores das métricas de complexidade consideradas neste estudo para *commits* com e sem mudanças arquiteturais (parte 1).

Ao comparar os resultados obtidos entre este e outros estudos envolvendo a relação entre métricas de complexidade de software com esforço de manutenção, bugs e necessidade de *refactoring*, observa-se que os valores de ACCM se reduzem nas mudanças da arquitetura (em relação a outros tipos de mudança), divergindo da lógica de predição de esforço de manutenção apresentadas por Muthanna et al. (2000) e Olague, Etzkorn e Cox (2006). Não se encontrou estudos comparáveis para as demais métricas (NPA e NPM).



(a) Comparação dos valores da métrica ΔNPA entre *commits* com mudanças na arquitetura e *commits* sem mudanças na arquitetura.



(b) Comparação dos valores da métrica ΔNPM entre *commits* com mudanças na arquitetura e *commits* sem mudanças arquiteturais.

Figure 4.27: Comparação das variações médias para os valores das métricas de complexidade consideradas neste estudo para *commits* com e sem mudanças arquiteturais (parte 2).

4.4.3 Alterações arquiteturais em um projeto induzem a maiores variações no grau de acoplamento do código do projeto que outros tipos de alterações?

A **RQ2.3** questiona se as mudanças arquiteturais produzem maiores alterações no grau de acoplamento do código do projeto que outros tipos de alterações. Para responder a essa pergunta, analisaram-se as métricas de acoplamento definidas na Seção 4.3.2.1. As subseções a seguir apresentam os resultados obtidos para cada métrica e, ao final, é apresentada uma discussão acerca do conjunto de métricas de acoplamento analisadas.

4.4.3.1 Conexões Aferentes por Classe (ACC)

A Tabela 4.16 apresenta os dados da estatística descritiva obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica ACC para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos centésimos para *commits* com alterações na arquitetura e dos milésimos para os demais *commits*, indicando que o número de conexões aferentes por classe variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 207 conexões aferentes adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 26 conexões aferentes adicionais para *commits* sem mudanças na arquitetura. Essas variações nos valores obtidos entre os dois conjuntos de *commits* avaliados (arquiteturais e não-arquiteturais) são estatisticamente significativas (teste U de Mann-Whitney-Wilcoxon, *p-value* de 0,017).

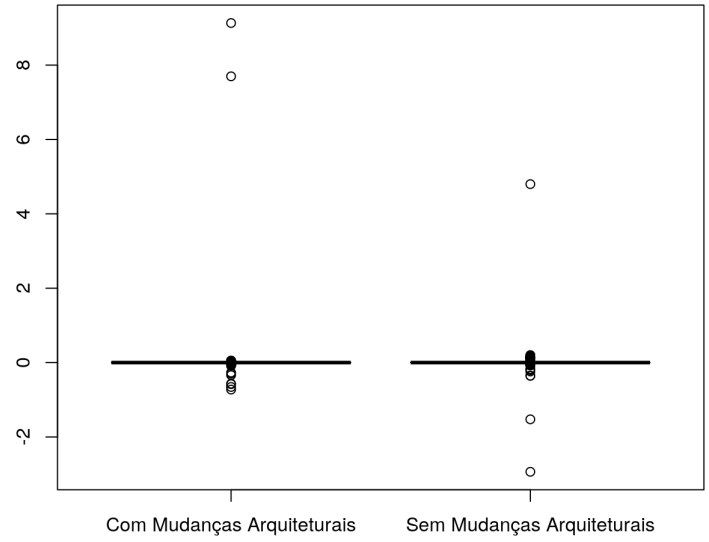
Table 4.16: Estatística Descritiva obtida para as variações da métrica ACC para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,056	0,007
Mediana	0,00	0,00
Amplitude	9,856	7,736
Mínimo	-0,725	-2,937
Máximo	9,131	4,799

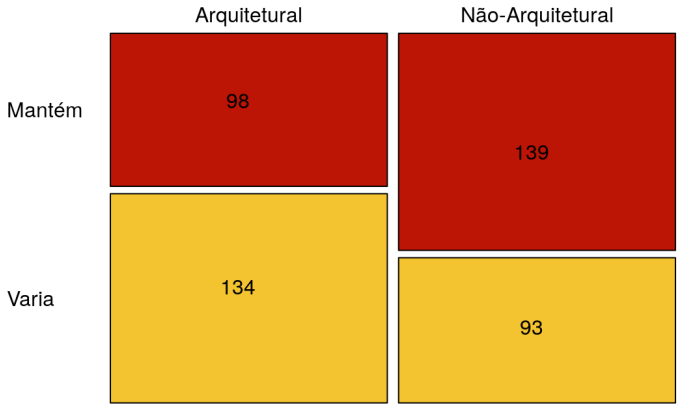
A Figura 4.28(a) apresenta um *boxplot*, contendo as variações dos valores obtidos para a métrica **Conexões Aferentes por Classe** para *commits* com mudanças arquiteturais e os *commits* com mudanças de outra natureza. Foi observado também em poucos *commits* houve mudança acentuada na quantidade de conexões aferentes por classe — a exemplo da versão δ (de 18/09/2010), onde se verificou uma variação média de 9,131 e da versão θ (de 08/11/2011), com uma variação média de 7,698. Para os *commits* que entregaram outros tipos de modificações no projeto, a variação mais significativa nessa métrica foi no *commit* ρ[17] (de 24/10/2016) com uma variação de 4,799. A parte b dessa figura, por sua

[17]Git key 8bc7d3107ace3b28ad35ac03f7a82dd7c68494cb

vez, apresenta a proporção de *commits* nos quais há variação nos valores da métrica ACC. Essa proporção é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença registrada é estatisticamente significativa (teste qui-quadrado, $p\text{-}value = 2{,}0e^{-4}$).



(a) ΔACC – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔACC – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.28: Métrica ΔACC para *commits* com e sem mudanças na arquitetura.

Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações positivas** (acréscimo na métrica) que os *commits* sem mudanças na arquitetura (73 *commits versus* 48 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica ACC. Nesse conjunto, é preciso destacar que, na maioria dos casos de *commits* sem alterações na arquitetura (139), a quantidade de conexões aferentes por classe não mudou (delta médio = 0). No grupo de *commits* com modificações na arquitetura, foram registradas 98 versões, onde não houve alteração nos valores da métrica ACC. A Figura 4.29 mostra a quantidade de *commits*, por versão, onde a média dos valores da métrica ACC aumentou ou diminuiu no projeto KDELibs. Embora a **variação** dos valores apurados para a métrica ACC tenha sido **predominante**, **não se registrou diferença** estatisticamente **significativa** (teste qui-quadrado, *p-value* de 0,772) na proporção de aumento/diminuição dos valores da métrica entre os conjuntos de *commits* (arquiteturais e não-arquiteturais).

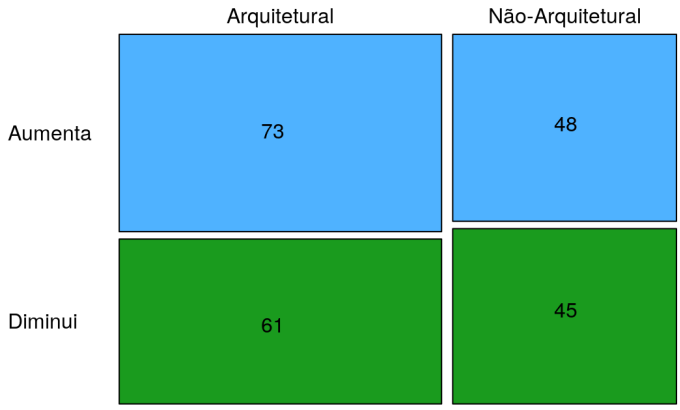


Figure 4.29: ΔACC: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

4.4.3.2 Acoplamento entre Objetos (CBO)

A Tabela 4.17 apresenta os dados da estatística descritiva obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica CBO para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos centésimos para os *commits* com alterações na arquitetura e dos milésimos para os demais *commits*, indicando que o acoplamento entre objetos variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos

por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 104 classes adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 7 classes a menos para *commits* sem mudanças na arquitetura. As variações registradas nos valores obtidos para a métrica ΔCBO entre os dois conjuntos de *commits* avaliados (arquiteturais e não-arquiteturais) são estatisticamente significativas (teste U de Mann-Whitney-Wilcoxon, *p-value* de $2{,}2e^{-16}$).

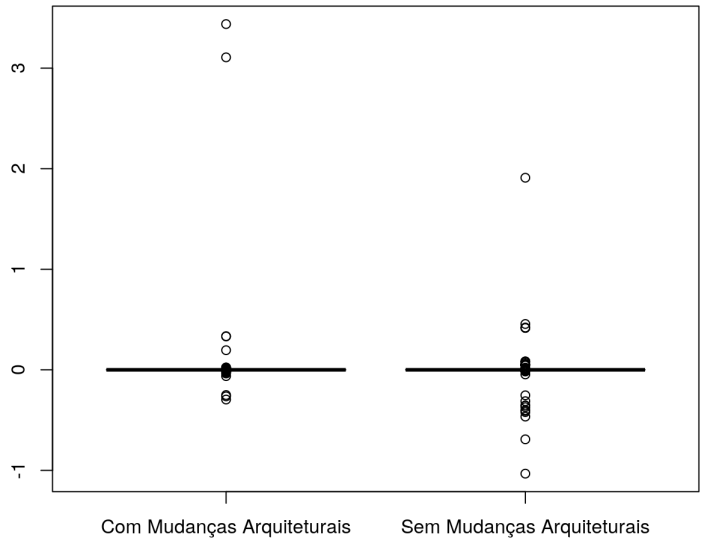
Table 4.17: Estatística Descritiva obtida para as variações da métrica CBO para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,028	-0,002
Mediana	0,00	0,00
Amplitude	3,734	2,943
Mínimo	-0,296	-1,033
Máximo	3,438	1,910

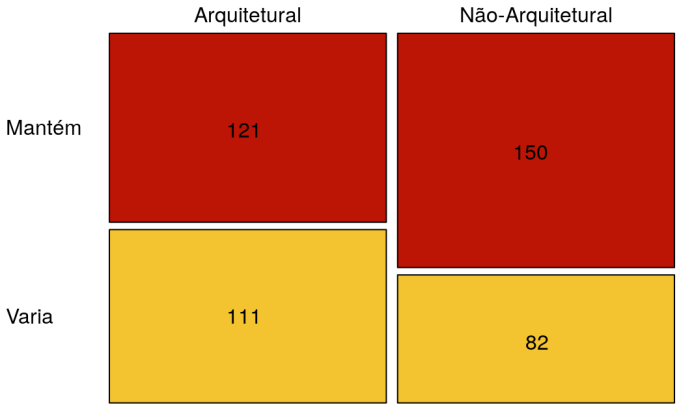
A Figura 4.30(a) apresenta um *boxplot*, contendo as variações dos valores obtidos para a métrica **Acoplamento entre Objetos** em *commits* com mudanças arquiteturais e os *commits* com mudanças de outra natureza. Foi observado também que, em poucos *commits* houve mudança acentuada no acoplamento entre objetos — a exemplo da versão δ (de 18/09/2010), onde se verificou uma variação média de 3,438 e da versão θ (de 08/11/2011), com uma variação média de 3,107. Para os *commits* que entregaram outros tipos de modificações no projeto, a variação mais significativa nessa métrica foi no *commit* ρ (de 24/10/2016) com uma variação de 1,910. Já a parte b dessa figura mostra que, a proporção de *commits* nos quais há variação nos valores da métrica CBO, é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. Essa diferença é estatisticamente significativa (teste qui-quadrado, *p-value* = $8{,}3e^{-3}$).

Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações negativas** (diminuição dos valores da métrica) que os *commits* sem mudanças na arquitetura (56 *commits versus* 38 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica CBO. Nesse conjunto, é preciso destacar que, na maioria dos casos (tanto em *commits* com mudanças arquiteturais quanto nos *commits* sem alterações na arquitetura - 121 x 150), a média do acoplamento entre objetos não mudou (delta médio = 0). A Figura 4.31 mostra a quantidade de *commits*, por versão, em que a média dos valores da métrica CBO aumentou ou diminuiu no projeto KDELibs. Conforme descrito nessa seção, a **manutenção** dos valores apurados para a métrica ΔCBO foi **predominante**. Nesse sentido, **não se registrou diferença** estatisticamente **significativa** (teste qui-quadrado, *p-value* de 0,675) na proporção de aumento/diminuição dos valores da métrica (onde houve variação) entre os conjuntos de *commits* (arquiteturais e não-arquiteturais).

4.4.3.3 Profundidade da Árvore de Herança (DiT)



(a) ΔCBO – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔCBO – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.30: Métrica ΔCBO para *commits* com e sem mudanças na arquitetura.

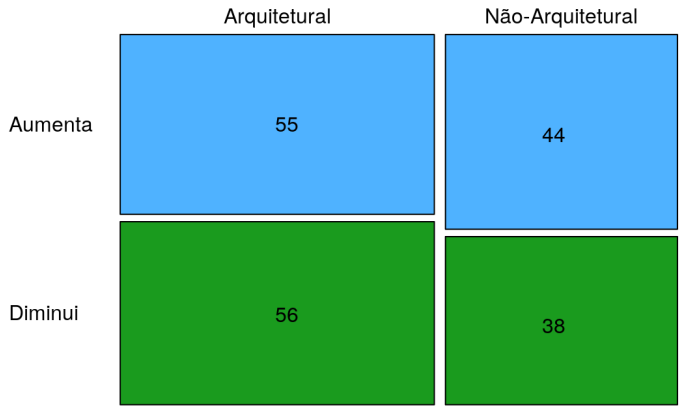


Figure 4.31: ΔCBO: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

A Tabela 4.18 apresenta os dados da estatística descritiva obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica DiT para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos milésimos, indicando que a profundidade da árvore de herança variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos, por versão, é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 15 níveis de herança adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 11 níveis de herança a menos para *commits* sem mudanças na arquitetura. As variações registradas nos valores obtidos para a métrica ΔDiT entre os dois conjuntos de *commits* avaliados (arquiteturais e não-arquiteturais) são estatisticamente significativas (teste U de Mann-Whitney-Wilcoxon, *p-value* de $2{,}2e^{-16}$).

Table 4.18: Estatística Descritiva obtida para as variações da métrica DiT para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,004	-0,003
Mediana	0,00	0,00
Amplitude	0,650	0,330
Mínimo	-0,143	-0,178
Máximo	0,507	0,152

A Figura 4.32(a) apresenta um *boxplot*, contendo as variações dos valores coletados para a métrica **Profundidade da Árvore de Herança** em *commits* com mudanças arquiteturais e os *commits* em que houveram mudanças de outra natureza. Não foram observados *commits* com mudanças significativas nos valores obtidos acerca da profundidade da árvore de herança. A parte b dessa figura, por sua vez, apresenta a proporção de *commits* nos quais há variação nos valores da métrica DiT. Essa proporção é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. Destaque-se que a diferença registrada é estatisticamente significativa (teste qui-quadrado, *p-value* = $2{,}9e^{-4}$).

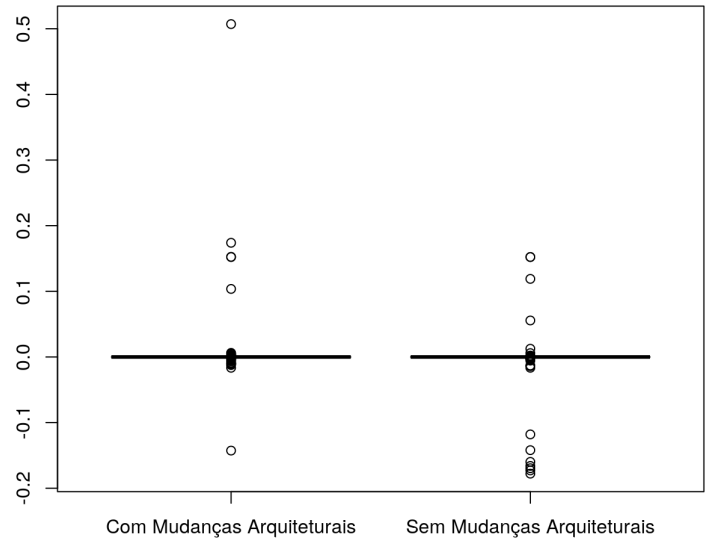
Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações negativas** (redução dos valores da métrica) que os *commits* sem mudanças na arquitetura (51 *commits versus* 26 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica DiT. Nesse conjunto, é preciso destacar que, na maioria dos casos (tanto em *commits* com mudanças arquiteturais quanto nos *commits* sem alterações na arquitetura - 149 x 185), a profundidade da árvore de herança não mudou (delta médio = 0). A Figura 4.33 mostra a quantidade de *commits*, onde a métrica DiT, por versão, aumentou ou diminuiu no projeto KDELibs. Conforme descrito, para a métrica DiT, a **manutenção** dos valores apurados foi **predominante**. Nesse sentido, **não se registrou diferença** estatisticamente **significativa** (teste qui-quadrado, *p-value* de 0,619) na proporção de aumento/diminuição dos valores da métrica (onde houve variação) entre os conjuntos de *commits* (arquiteturais e não-arquiteturais).

4.4.3.4 Número de Classes Filhas (NOC)

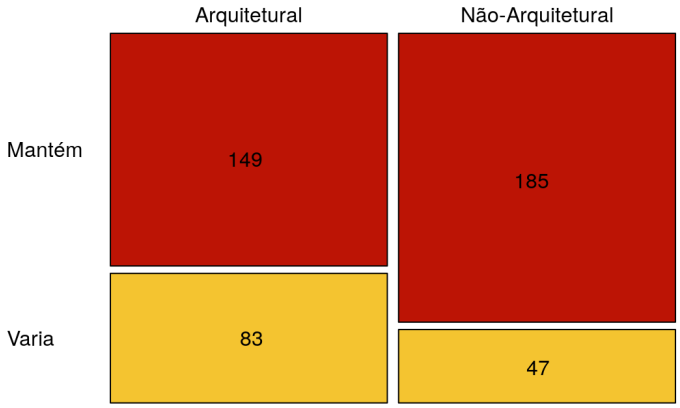
A Tabela 4.19 apresenta os dados da estatística descritiva obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica NOC para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos milésimos, indicando que o número de classes filhas variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 4 classes filhas adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 1 classe filha a menos para *commits* sem mudanças na arquitetura. As variações registradas nos valores obtidos para a métrica ΔNOC entre os dois conjuntos de *commits* avaliados (arquiteturais e não-arquiteturais) são estatisticamente significativas (teste U de Mann-Whitney-Wilcoxon, *p-value* de $1{,}4e^{-3}$).

A Figura 4.34(a) apresenta um *boxplot*, contendo as variações dos valores obridos para a métrica **Número de Classes Filhas** em *commits* com mudanças arquiteturais e os *commits* em que houveram mudanças de outra natureza. Não foram observados *commits* com mudanças significativas no número de classes filhas. Já a parte b dessa figura mostra que, a proporção de *commits* nos quais há variação nos valores da métrica NOC, é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. Essa diferença é significativa estatisticamente (teste qui-quadrado, *p-value* = 0,044).

Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram



(a) ΔDiT – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔDiT – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.32: Métrica ΔDiT para *commits* com e sem mudanças na arquitetura.

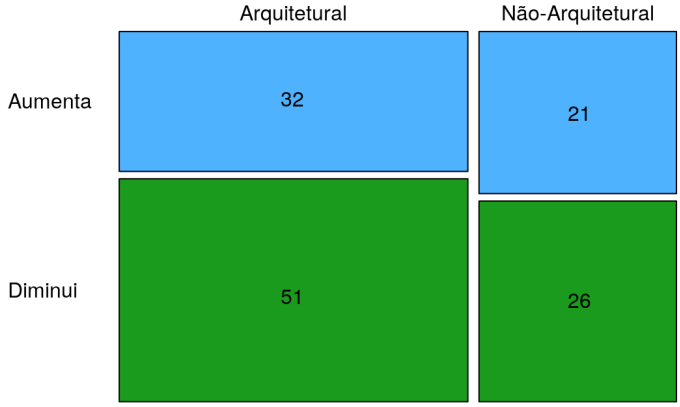
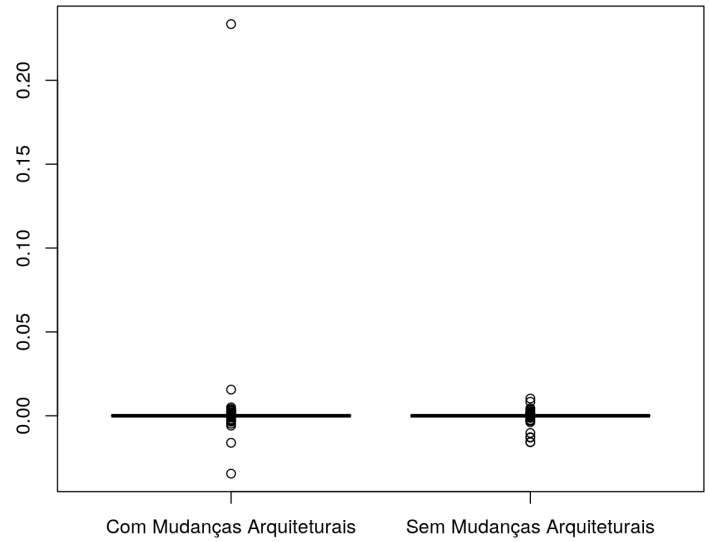


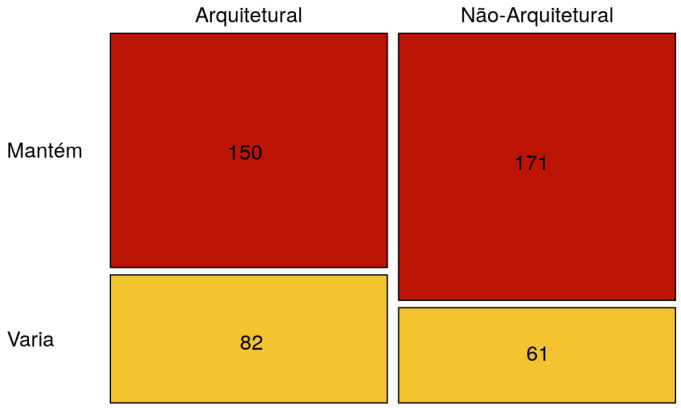
Figure 4.33: ΔDiT: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

Table 4.19: Estatística Descritiva obtida para as variações da métrica NOC para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,001	-0,0002
Mediana	0,00	0,00
Amplitude	0,257	0,026
Mínimo	-0,034	-0,016
Máximo	0,233	0,010



(a) ΔNOC – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔNOC – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.34: Métrica ΔNOC para *commits* com e sem mudanças na arquitetura.

mais **variações negativas** (redução dos valores da métrica) que os *commits* sem mudanças na arquitetura (43 *commits versus* 37 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica NOC. Nesse conjunto, é preciso destacar que, na maioria dos casos, (tanto em *commits* com mudanças arquiteturais quanto nos *commits* sem alterações na arquitetura - 150 x 171), o número de classes filhas não mudou (delta médio = 0). A Figura 4.35 mostra a quantidade de *commits*, na qual a média dos valores da métrica NOC, por versão, aumentou ou diminuiu no projeto KDELibs. Conforme descrito nessa seção, a **manutenção** dos valores apurados para a métrica ΔNOC foi **predominante**. Nesse sentido, **não se registrou diferença** estatisticamente **significativa** (teste qui-quadrado, *p-value* de 0,419) na proporção de aumento/diminuição dos valores da métrica (onde houve variação) entre os conjuntos de *commits* (arquiteturais e não-arquiteturais).

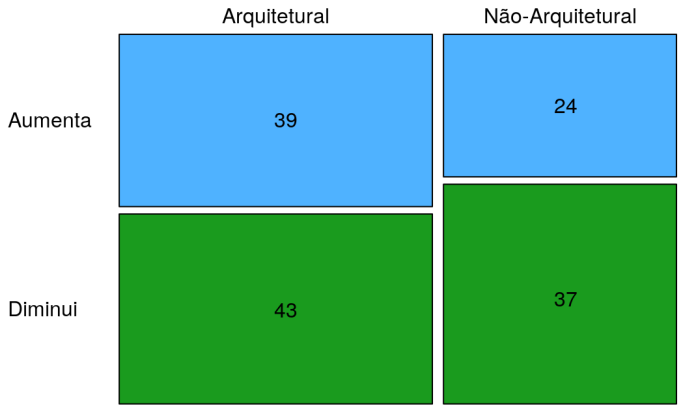


Figure 4.35: ΔNOC: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

4.4.3.5 Resposta por Classe (RFC)

A Tabela 4.20 apresenta os dados da estatística descritiva obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica RFC para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos décimos para os *commits* com alteração na arquitetura e dos centésimos para os *commits* não-arquiteturais, indicando que o número de respostas por classe variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada

versão, a variação verificada foi de 496 respostas adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 255 respostas adicionais para *commits* sem mudanças na arquitetura. As variações registradas entre valores obtidos para a métrica ΔRFC nos dois conjuntos de *commits* avaliados (arquiteturais e não-arquiteturais) são estatisticamente significativas (teste U de Mann-Whitney-Wilcoxon, *p-value* de $2{,}2e^{-16}$).

Table 4.20: Estatística Descritiva obtida para as variações da métrica RFC para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

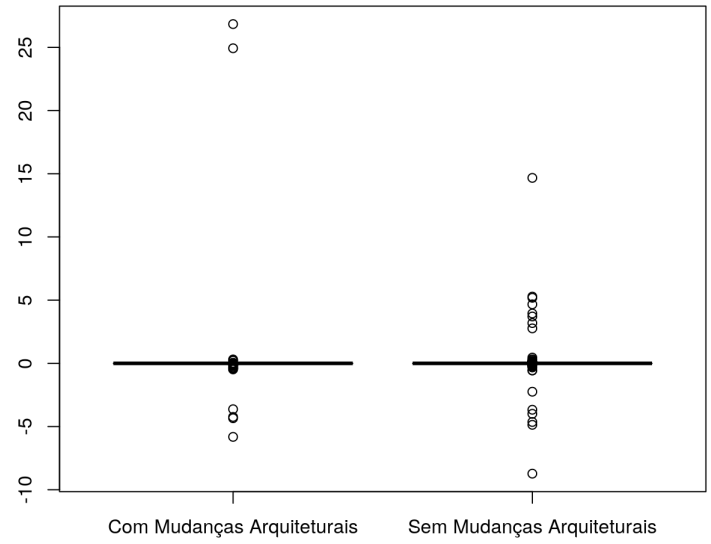
	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,134	0,069
Mediana	0,00	0,00
Amplitude	32,656	23,400
Mínimo	-5,818	-8,728
Máximo	26,838	14,672

A Figura 4.36(a) apresenta um *boxplot*, contendo as variações dos valores coletados para a métrica **Resposta por Classe** em *commits* em que houveram mudanças arquiteturais e os *commits* que registraram mudanças de outra natureza. Foi observado também que em poucos *commits* houve mudança aguda na quantidade de respostas por classe — a exemplo da versão δ (de 18/09/2010), onde se verificou uma variação média de 4,583 e da versão θ (de 08/11/2011), com uma variação média de 4,039. Não foi observado *commit*, em meio ao conjunto selecionado, com alterações de outra natureza que apresentasse alterações significativas nessa métrica. A parte b da Figura 4.36, por sua vez, ilustra que, a proporção de *commits* nos quais há variação nos valores da métrica RFC, é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença é estatisticamente significativa (teste qui-quadrado, *p-value* = $8{,}3e^{-5}$).

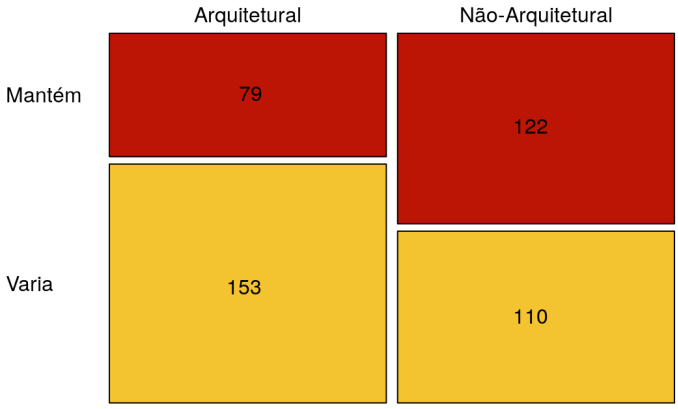
Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações positivas** (aumento dos valores da métrica) que os *commits* sem mudanças na arquitetura (77 *commits versus* 56 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica RFC. Nesse conjunto, é preciso destacar que, na maioria dos casos dos *commits*, onde não houve mudanças arquiteturais (122) o valor da métrica resposta por classe não mudou (delta médio = 0). Para os *commits* com mudanças arquiteturais, em 79 versões, o valor de RFC não sofreu modificações. A Figura 4.37 mostra a quantidade de versões de*commits* onde a métrica RFC por versão aumentou ou diminuiu no projeto KDELibs. Embora a **variação** dos valores apurados para a métrica ΔRFC tenha sido predominante, **não se registrou diferença** estatisticamente **significativa** (teste qui-quadrado, *p-value* de 1,0) na proporção de aumento/diminuição dos valores da métrica entre os conjuntos de *commits* (arquiteturais e não-arquiteturais).

4.4.3.6 Discussão sobre as métricas de acoplamento coletadas

Ao analisar os valores das métricas de acoplamento obtidas, observa-se, usando o



(a) ΔRFC – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔRFC – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.36: Métrica ΔRFC para *commits* com e sem mudanças na arquitetura.

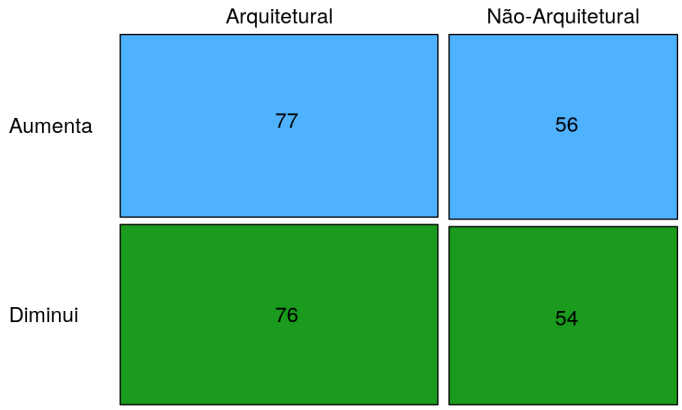


Figure 4.37: ΔRFC: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

teste U Mann-Whitney-Wilcoxon, significância estatística nos dados obtidos sobre todas as métricas de acoplamento coletadas. Dessa forma, pode-se considerar que *commits* com modificações na arquitetura produzem variações diferentes de outros tipos de alterações no projeto para os valores das métricas de acoplamento obtidas. Analisando a Tabela 4.21, pode-se observar que, nos *commits* estudados, a tendência majoritária variou de métrica para métrica. Enquanto para as métricas **Acoplamento entre Objetos**, **Profundidade da Árvore de Herança** e **Número de Classes Filhas**, a maioria dos *commits* estudados apresentou manutenção nos valores obtidos, para as métricas **Conexões Aferentes por Classe** e **Resposta por Classe** predominou *commits* com alterações nos valores das métricas e registrando aumento em seus valores.

As Figuras 4.38, 4.39 e 4.40 apresentam um conjunto de *mosaic plots*, propondo a comparação individual para cada métrica de acoplamento avaliada. As proporções obtidas (aumento/diminuição/manutenção) para os valores de cada métrica de acoplamento analisada são, **estatisticamente significativas**. A Tabela 4.21 apresenta os *p-values* obtidos pelo teste qui-quadrado para os valores de cada métrica apresentada nas Figuras 4.38, 4.39 e 4.40.

Sobre as métricas de acoplamento repousavam as principais expectativas em relação aos impactos no código a partir de mudanças na arquitetura, uma vez que o projeto arquitetural deveria interferir diretamente no grau de acoplamento do código produzido. Esperava-se que o **Acoplamento entre Objetos**, as **Conexões Aferentes por Classe** e as **Respostas por Classe** fossem afetadas por mudanças na arquitetura. Apesar de serem métricas que medem detalhes de design, esperava-se que o projeto arquitetural interferisse diretamente no grau de acoplamento medido por essas métricas. Dessas,

apenas o **Acoplamento entre Objetos**, de forma surpreendente, obteve manutenção nos valores de métrica como tendência majoritária. Ainda de forma surpreendente, as variações observadas nas métricas **Acoplamento entre Objetos** e **Respostas por Classe** foram de acréscimos em seus valores. Esperava-se por uma redução em seus valores, visto que, em um projeto arquitetural, que preza pelo reuso e modularidade, esses valores deveriam ser sucessivamente reduzidos. As métricas, que medem o acoplamento a partir da herança (**Profundidade da Árvore de Herança** e **Número de Classes Filhas**), referem-se a detalhes de implementação e obtiveram o resultado esperado: manutenção de seus valores como tendência majoritária, pois, como já afirmado, mudanças arquiteturais não devem tratar desse tipo de detalhe.

Table 4.21: Resumo dos testes estatísticos para os valores das métricas de Acoplamento obtidos frente ao projeto KDELibs considerando todos os arquivos de cada versão comparando *commits* com modificações arquiteturais e *commits* sem mudanças arquiteturais.

Métrica	**Mann-Whitney-Wilcoxon**			**Teste Qui-Quadrado**	
	P-value	**Significância Estatística**	**Tendência Verificada**	***P-value***	**Significância Estatística**
ACC	$0,017$	SIM	Variação (Aumento)	$2,0e^{-4}$	SIM
CBO	$2,2e^{-16}$	SIM	Manutenção	$8,3e^{-3}$	SIM
DiT	$2,2e^{-16}$	SIM	Manutenção	$2,9e^{-4}$	SIM
NOC	$1,4e^{-3}$	SIM	Manutenção	$0,044$	SIM
RFC	$2,2e^{-16}$	SIM	Variação (Aumento)	$8,3e^{-5}$	SIM

Lançando mão dos tópicos arquiteturais, definidos na Subseção 3.3.4.3, a análise individualizada de cada um perante os resultados obtidos para as métricas de acoplamento coletadas, pode-se verificar que para cada métrica se sobressaíram os seguintes tópicos:

- ACC: A maior parte dos *commits* arquiteturais, que registrou **aumento** de seus valores, se referia aos tópicos (VIII) Mecanismos de Sincronização, (XI) Relação entre Elementos e (XIII) Requisitos Não-Funcionais (42 *commits* de um total de 73 – 58%). Nesse grupo de *commits* foram registrados 14 tópicos com esse tipo de comportamento.

- CBO: A maior parte dos *commits* arquiteturais, que registrou a **manutenção** de seus valores, se referia aos tópicos (XI) Relação entre Elementos, (VIII) Mecanismos de Sincronização, (XIII) Requisitos Não-Funcionais e (VII) Protocolos de Comunicação (76 de 121 - 63%). Nesse grupo de *commits* foram registrados 13 tópicos com esse tipo de comportamento.

- DiT: A maior parte dos *commits* arquiteturais, que registrou a **manutenção** de seus valores, se referia aos tópicos (XI) Relação entre Elementos, (VIII) Mecanismos de Sincronização, (XIII) Requisitos Não-Funcionais, (VI) Estruturas do Sistema e (VII) Protocolos de Comunicação (103 de 149 - 69%). Nesse grupo de *commits* foram registrados 14 tópicos com esse tipo de comportamento.

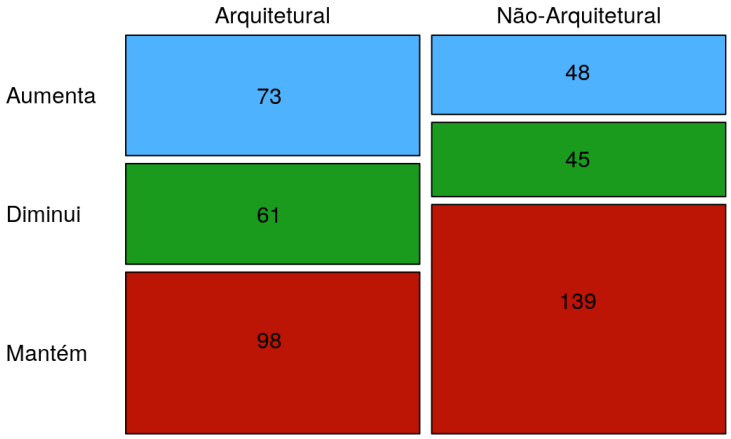
- NOC: A maior parte dos *commits* arquiteturais, que registrou a **manutenção** de seus valores, se referia aos tópicos (XI) Relação entre Elementos, (VIII) Mecanismos de Sincronização, (XIII) Requisitos Não-Funcionais, (VI) Estruturas do Sistema e (VII) Protocolos de Comunicação (104 *commits* de um total de 150 - 69%). Nesse grupo de *commits* foram registrados 14 tópicos com esse tipo de comportamento.

- RFC: A maior parte dos *commits* arquiteturais, que registrou o **aumento** de seus valores, se referia aos tópicos (VIII) Mecanismos de Sincronização, (XI) Relação entre Elementos e (XIII) Requisitos Não-Funcionais (43 *commits* de um total de 77 - 56%). Nesse grupo de *commits* foram registrados 14 tópicos com esse tipo de comportamento.

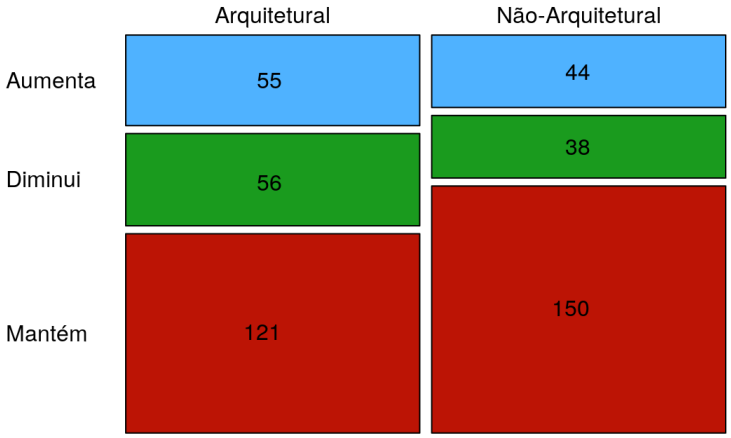
Comparando as tendências majoritárias obtidas por cada métrica de acoplamento, em *commits* com mudanças arquiteturais, e as tendências obtidas por cada tópico arquitetural nessas métricas, é possível perceber que não há um comportamento uniforme em todos eles. Para a métrica **ACC**, a tendência geral verificada foi de aumento dos valores em *commits* com mudança arquitetural. No entanto, os tópicos, que se destacaram, individualmente, apresentaram tendências que divergem da tendência geral, já que os tópicos Relação entre Elementos e Mecanismos de Acesso a Dados indicaram, como tendência majoritária, a manutenção dos valores obtidos para a métrica ACC enquanto o tópico Motivações, Objetivos e Restrições sinalizou a diminuição nos valores dessa métrica. As demais métricas de acoplamento tiveram comportamento diferente. A métrica **CBO**, por exemplo, teve indicados os tópicos Relação entre Elementos, Requisitos Não-Funcionais, Protocolos de Comunicação, Elementos Arquiteturais de Conexão, Mecanismos de Acesso a Dados e Motivações, Objetivos e Restrições com tendências majoritárias no mesmo sentido que a tendência geral dessa métrica (manutenção de valores). Já as métricas **DiT** e **NOC** apresentaram, através dos valores obtidos, convergência com a tendência geral (manutenção dos valores) em quase todos os tópicos arquiteturais estudados. Estão incluídos nesse conjunto Relação entre Elementos, Mecanismos de Sincronização, Requisitos não-funcionais, Estruturas do Sistema, Protocolos de Comunicação, Elementos Arquiteturais de Conexão, Mecanismos de Acesso à Dados, Motivações, Objetivos e Restrições, Estruturas Globais de Controle e Organização Fundamental. A métrica **RFC**, por sua vez, apresentou convergência entre a tendência majoritária e apenas os tópicos Requisitos Não-Funcionais e Estruturas Globais de Controle. Ressalte-se que os tópicos Motivações, Objetivos e Restrições e Relação entre Elementos sugeriram tendências majoritárias contrárias à tendência geral verificada para essa métrica. Para a primeira registrou-se a diminuição de valores, enquanto a segunda registrou a manutenção dos valores.

Dessa forma, os dados aqui apresentados sugerem:

> Os valores de métricas de Acoplamento estudadas não se alteram de forma generalizada em mudanças arquiteturais, sejam eles analisados de forma geral, sejam através dos tópicos arquiteturais definidos nessa pesquisa.

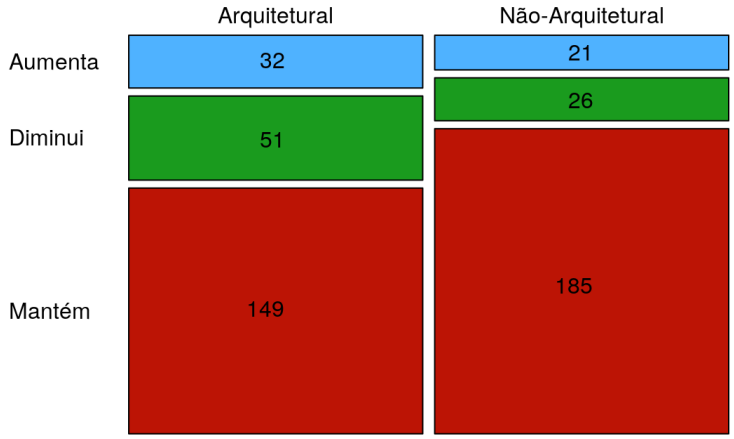


(a) Comparação das variações nos valores da métrica ΔACC entre *commits* com e sem mudanças na arquitetura.

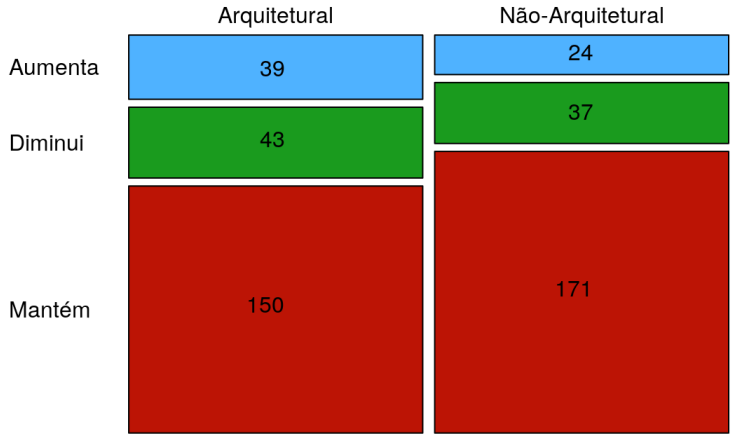


(b) Comparação das variações nos valores da métrica ΔCBO entre *commits* com e sem mudanças na arquitetura.

Figure 4.38: Comparação das variações médias para os valores das métricas de acoplamento consideradas nesse estudo para *commits* com e sem mudanças na arquitetura (parte 1).

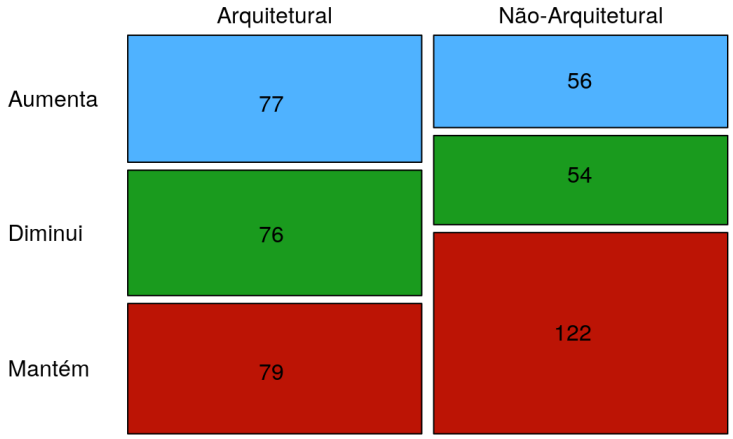


(a) Comparação das variações nos valores da métrica ΔDiT entre *commits* com e sem mudanças na arquitetura.



(b) Comparação das variações nos valores da métrica ΔNOC entre *commits* com e sem mudanças na arquitetura.

Figure 4.39: Comparação das variações médias para os valores das métricas de acoplamento consideradas nesse estudo para *commits* com e sem mudanças na arquitetura (parte 2).



(a) Comparação das variações nos valores da métrica ΔRFC entre *commits* com e sem mudanças na arquitetura.

Figure 4.40: Comparação das variações médias para os valores das métricas de acoplamento consideradas nesse estudo para *commits* com e sem mudanças na arquitetura (parte 3).

Ao se comparar os resultados obtidos por este e outros estudos, que envolveram a relação entre métricas de acoplamento de software com esforço de manutenção, bugs e necessidade de *refactoring*, pode-se observar alguns aspectos. Em primeiro lugar, RFC aumenta nas versões onde ocorreram mudanças da arquitetura, quando comparadas versões com outros tipos de mudança, seguindo a mesma lógica da predição de defeitos (MISRA; BHAVSAR, 2003; JIN; LIU, 2010) e, em conjunto com outras métricas (LI; HENRY, 1993; TANG; KAO; CHEN, 1999; THWIN; QUAH, 2005), é preciso destacar que Bengtsson (1998) informou que a RFC, é uma candidata a adaptação, num processo de produção de métricas específicas para arquitetura de software. Em segundo lugar, os valores das métrica CBO, DiT e NOC, registraram a manutenção dos valores nos *commits* em que ocorreram mudanças na arquitetura, assim como nos outros tipos de mudança, ao contrário dos estudos de Li e Henry (1993), Misra e Bhavsar (2003), Thwin e Quah (2005) e Jin e Liu (2010), que apontam o aumento dessas métricas como sendo indicadores da presença de bugs e preditoras de esforço de manutenção. Não foram encontrados estudos comparáveis para a métrica ACC.

A Análise de Componente Principal (HOTELLING, 1933) realizada, considerando todos os valores das variações das métricas de acoplamento estudadas, apontou que os valores dessas métricas não são dependentes entre si, uma vez que os coeficientes que as

representam (*PC1* e *PC2*) têm valores distantes entre si[18] (ver Tabela 4.22), refletindo na posição de seus vetores conforme retrata o gráfico *PC1* x *PC2* presente na Figura 4.41.

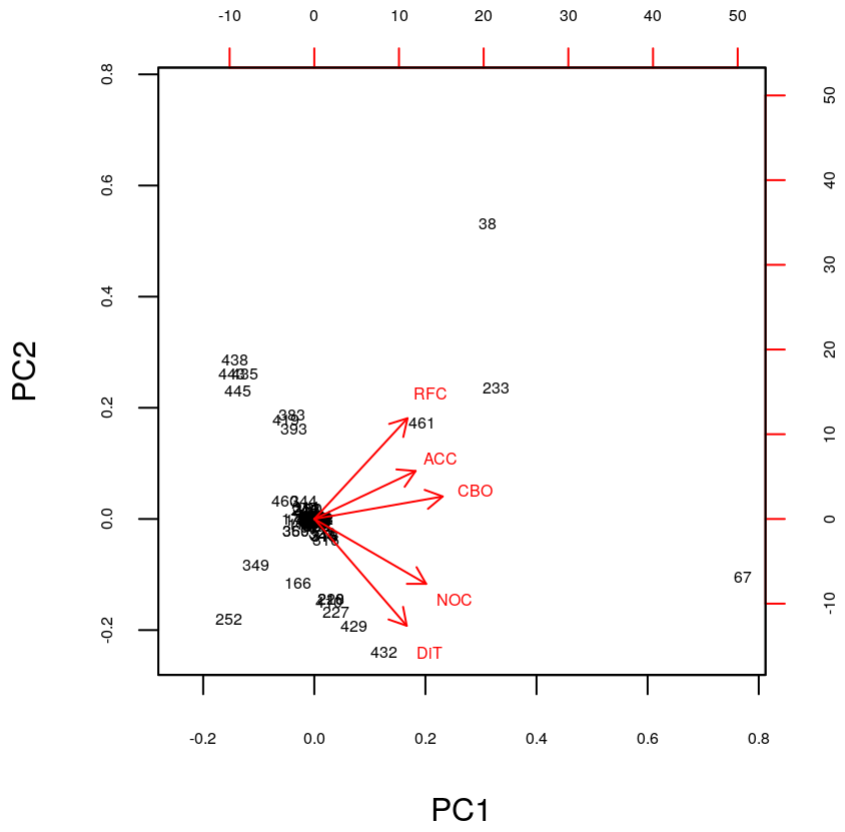


Figure 4.41: Análise de Componente Principal - Variações nos valores das Métricas de Acoplamento (normalizadas) por *commit*.

Table 4.22: Tabela de Rotação da Análise de Componente Principal para variações nas métricas de acoplamento do projeto KDELibs.

Métrica	**PC1**	**PC2**	**PC3**	**PC4**	**PC5**
ACC	0.4262264	0.2836557	-0.85803263	0.005776423	-0.04021385
CBO	0.5402504	0.1329600	0.33230230	-0.567586796	-0.50781053
DiT	0.3888500	-0.6319805	-0.04531816	-0.326638555	0.58365211
NOC	0.4704767	-0.3841004	0.12647746	0.716757784	-0.31840270
RFC	0.3923110	0.5957588	0.36783814	0.239535790	0.54633437

[18]Considerando o método dos mínimos quadrados

Em linhas gerais, os coeficientes apresentados na Tabela 4.22 não sugerem uma ligação entre a variação das métricas de acoplamento estudadas. A Figura 4.41, apresenta os vetores com as variações das métricas de acoplamento consideradas, a partir dos coeficientes de *PC1* e *PC2*. Essas dimensões da análise foram adotadas, pois juntas explicam 80,0% da variância encontrada nesse conjunto. Dessa forma, é possível afirmar que, em função da caracterização mudanças arquiteturais, a variação das métricas DiT e NOC seguem o sentido inverso de ACC, CBO e RFC, mas ainda assim é necessário observar as variações nos valores das métricas de acoplamento apresentadas.

4.4.4 Alterações arquiteturais em um projeto induzem a maiores variações na coesão dos módulos do projeto que outros tipos de alterações?

A **RQ2.4** questiona se as mudanças arquiteturais produzem maiores alterações na coesão dos módulos do projeto que outros tipos de alterações. Para responder a essa pergunta, foram analisadas as métricas de coesão definidas na Seção 4.3.2.1 a partir dos valores obtidos com o estudo das versões selecionadas no projeto KDELibs (Seção 4.3.2.2). As subseções, a seguir, apresentam os resultados obtidos para cada métrica estudada e, ao final, é apresentada uma discussão acerca do conjunto de métricas de coesão analisadas.

4.4.4.1 Falta de Coesão dos Métodos (LCOM4)

A Tabela 4.23 apresenta os dados da estatística descritiva obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica LCOM4 para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos centésimos para os *commits* com alterações na arquitetura e dos milésimos para os demais *commits*, indicando que a falta de coesão dos métodos variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 41 unidades de coesão adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 1 unidade de coesão adicionais para *commits* sem mudanças na arquitetura. No entanto, é preciso registrar que **não houve significância estatistica** nas variações obtidas pelos conjuntos de *commits* avaliados (arquiteturais e não-arquiteturais) para a métrica ΔLCOM4 (teste U de Mann-Whitney-Wilcoxon, *p-value* =0,677).

A Figura 4.42(a) apresenta um *boxplot*, contendo as variações dos valores coletados para a métrica **Falta de Coesão por Método** no conjunto de *commits* com mudanças arquiteturais e os *commits* com mudanças de outra natureza. Foi observado também em poucos *commits* houve mudança aguda na falta de coesão entre os métodos — a exemplo da versão δ (de 18/09/2010), onde se verificou uma variação média de 1,635 e da versão θ (de 08/11/2011), com uma variação média de 1,256. Em relação aos *commits* que não modificaram a arquitetura do projeto, as variações mais significativas nessa métrica

foram nos *commit* ρ (de 24/10/2016), com uma variação de -2,737 e no *commit* τ[19] (de 01/02/2012), onde foi registrada uma variação de 1,566. Já a parte b dessa figura mostra que, a proporção de *commits* nos quais há variação nos valores da métrica LCOM4, é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença é estatisticamente significativa (teste qui-quadrado, *p-value* = $6{,}8e^{-4}$).

Table 4.23: Estatística Descritiva obtida para as variações da métrica LCOM4 para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

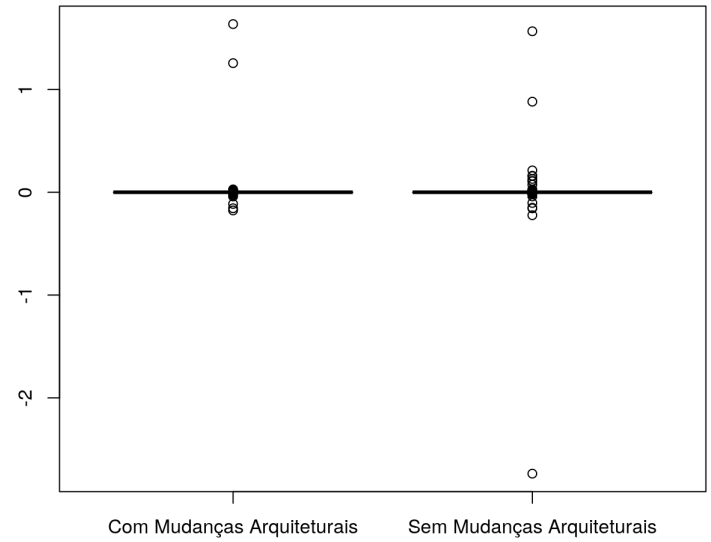
	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,011	-0,0001
Mediana	0,00	0,00
Amplitude	1,812	4,303
Mínimo	-0,177	-2,737
Máximo	1,635	1,566

Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações positivas** (aumento dos valores da métrica) que os *commits* sem mudanças na arquitetura (59 *commits versus* 43 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica LCOM4. Nesse conjunto, é preciso destacar que, na maioria dos casos, (tanto em *commits* com mudanças arquiteturais quanto nos *commits* sem alterações na arquitetura - 118 x 155), a falta de coesão dos métodos não mudou (delta médio = 0). A Figura 4.43 mostra a quantidade de *commits*, por versão, na qual a métrica LCOM4 aumentou ou diminuiu no projeto KDELibs. Seguindo o resultado obtido com a avaliação de significância estatística obtida na comparação entre as proporções expressas na Figura 4.42(b), também **não se registrou significância estatística** (teste qui-quadrado, *p-value* de 0,683) na proporção de aumento/redução nos valores da métrica ΔLCOM4 entre os dois conjuntos de *commits* analisados (arquiteturais e não-arquiteturais).

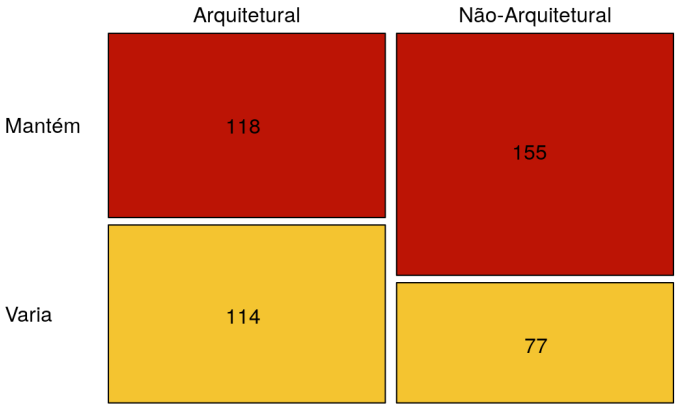
4.4.4.2 Complexidade Estrutural (SC)

A Tabela 4.24 apresenta os dados da estatística descritiva obtida com os conjuntos de *commits* analisados. Esses valores representam as médias das variações obtidas (diferenças de valores da métrica SC para os pares de conjuntos A_w e A_{w-1}, e N_w e N_{w-1}) em cada *commit* analisado. Como se vê, a variação média ficou posicionada na casa dos décimos para os *commits* com alterações arquiteturais e dos centésimos para os demais *commits*, indicando que a complexidade estrutural variou pouco tanto para *commits* arquiteturais quanto para *commits* não-arquiteturais. Considerando que o número médio de arquivos por versão é de, aproximadamente, 3.700, para cada versão, a variação verificada foi de 751 unidades de complexidade estrutural adicionais na soma de todos os arquivos modificados para *commits* com mudanças na arquitetura e de 59 unidades de complexidade estrutural adicionais para *commits* sem mudanças na arquitetura. As variações registradas nos valores obtidos para a métrica ΔSC entre os dois conjuntos de

[19]Git key 233075d84dddd4183aac131e49674eac9685efdf



(a) ΔLCOM4 – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔLCOM4 – Quantidade de *commits* com alteração no valor da métrica: *commits* com mudanças arquiteturais x *commits* sem mudanças arquiteturais

Figure 4.42: Métrica ΔLCOM4 para *commits* com e sem mudanças na arquitetura.

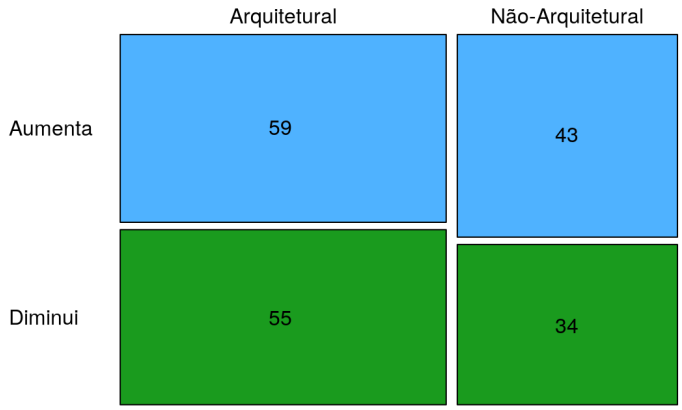


Figure 4.43: ΔLCOM4: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

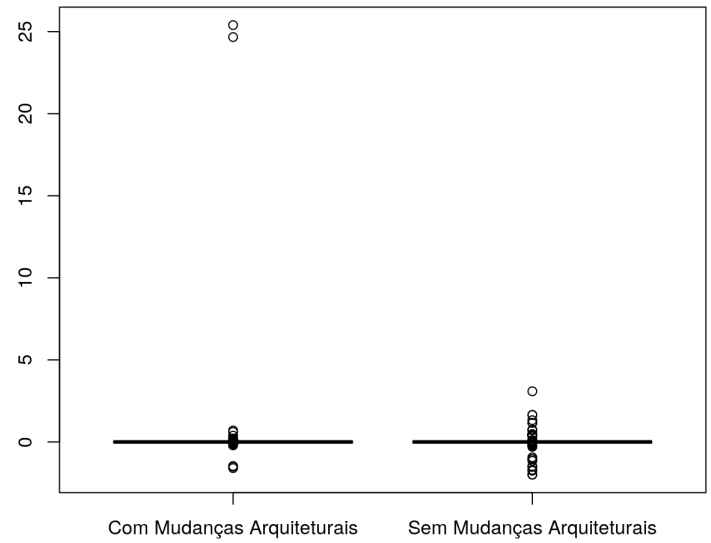
commits estudados (arquiteturais e não-arquiteturais) são estatisticamente significativas (teste U de Mann-Whitney-Wilcoxon, *p-value* =$2{,}2e^{-16}$).

Table 4.24: Estatística Descritiva obtida para as variações da métrica SC para o projeto KDELibs, considerando os *commits* arquiteturais x *commits* não-arquiteturais.

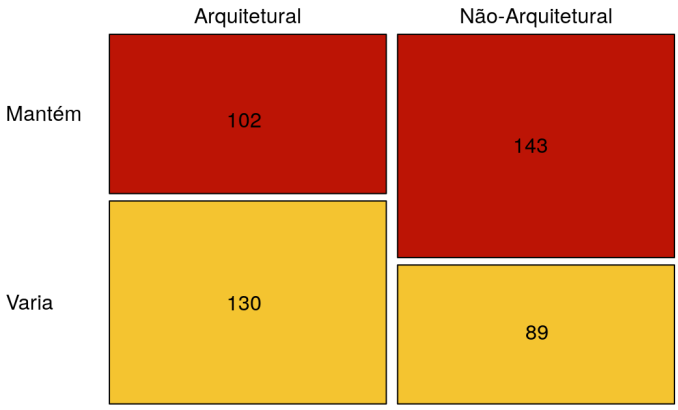
	***Commits* com Mudanças Arquiteturais**	***Commits* sem Mudanças Arquiteturais**
Média	0,203	0,016
Mediana	0,00	0,00
Amplitude	26,994	5,421
Mínimo	-1,593	-1,993
Máximo	25,401	3,085

A Figura 4.44(a) apresenta um *boxplot*, contendo as variações dos valores obtidos para a métrica **Complexidade Estrutural** em *commits* com mudanças arquiteturais comparando-os com os *commits* em que houveram mudanças de outra natureza. Foi observado também que em poucos *commits* houve mudança acentuada na complexidade estrutural medida — a exemplo da versão δ (de 18/09/2010), onde se verificou uma variação média de 25,401 e da versão θ (de 08/11/2011), com uma variação média de 24,671 —, quando comparadas aos *commits* que entregaram outros tipos de modificações no projeto — a variação mais significativa nessa métrica foi no *commit* τ (de 01/02/2012), com uma variação de 3,085. A parte b dessa figura, por sua vez, apresenta a proporção de *commits* nos quais há variação nos valores da métrica SC. Essa proporção é maior dentre os *commits* arquiteturais do que dentre os *commits* não-arquiteturais. A diferença

registrada é estatisticamente significativa (teste qui-quadrado, *p-value* = $1{,}9e^{-4}$).



(a) ΔSC – Variação dos valores da métrica por versão: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais



(b) ΔSC – Quantidade de *commits* com alteração no valor da métrica: *Commits* com Mudanças Arquiteturais x *Commits* sem Mudanças Arquiteturais

Figure 4.44: Métrica ΔSC para *commits* com e sem mudanças na arquitetura.

Observou-se que, em geral, os *commits* com modificações arquiteturais apresentaram mais **variações positivas** (aumento dos valores da métrica) que os *commits* sem mudanças na arquitetura (71 *commits versus* 48 *commits* sem mudanças na arquitetura) para os valores obtidos para a métrica SC. Nesse conjunto, é preciso destacar que, na maioria dos

casos de *commits* sem mudanças na arquitetura (143 *commits*) a complexidade estrutural média não mudou (delta médio = 0). Para *commits* com mudanças arquiteturais, verificaram-se que 102 deles, também não registraram alterações no valor da métrica SC. A Figura 4.45 mostra a quantidade de *commits*, por versão, onde a métrica SC aumentou ou diminuiu no projeto KDELibs. Embora tenha sido registrada a **predominância** da **variação** nos valores obtidos com a métrica ΔSC, **não se registrou diferença** estatisticamente **significativa** (teste qui-quadrado, *p-value* de 1,0) na proporção de aumento/redução da métrica entre os dois conjuntos de *commits* (arquiteturais e não-arquiteturais).

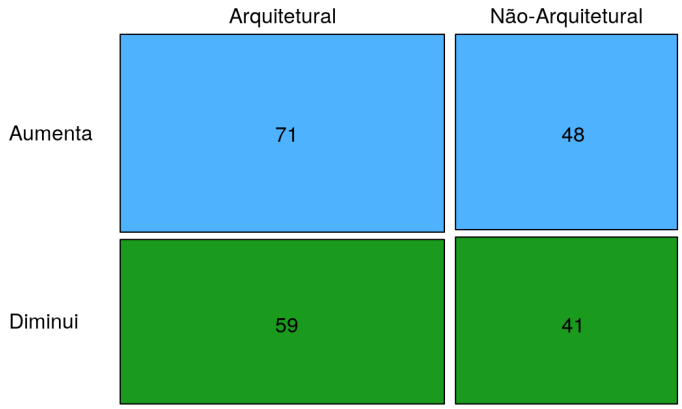


Figure 4.45: ΔSC: Comparação quantidade de *commits* com mudanças arquiteturais x *commits* sem mudança arquitetural – aumento x redução.

4.4.4.3 Discussão sobre as métricas de coesão coletadas

Ao analisar os valores das métricas de coesão coletadas, deduz-se, usando o teste U de Mann-Whitney-Wilcoxon, significância estatística nos dados obtidos sobre a medida de complexidade estrutural. Dessa forma, pode-se considerar que os *commits* com mudanças na arquitetura produzem variações diferentes nessa métrica (SC) que outros tipos de alterações no projeto a partir da avaliação dos valores dessa métrica. Analisando a Tabela 4.25, pode-se visualizar a tendência de variação expressa acima.

A Figura 4.46 apresenta um conjunto de *mosaic plots*, apresentando a comparação individual para cada métrica de coesão avaliada. As proporções obtidas (aumento/diminuição/manutenção) para os valores de cada métrica de coesão analisada são estatisticamente significativas. A Tabela 4.25 apresenta os *p-values* obtidos pelo teste qui-quadrado para as distribuições de valores de cada métrica apresentada na Figura 4.46.

Desconsiderando-se a significância estatística não obtida pela métrica LCOM4, pode-

Table 4.25: Resumo dos testes estatísticos realizados com os valores das métricas de Coesão obtidos frente ao projeto KDELibs considerando todos os arquivos de cada versão comparando *commits* com modificações arquiteturais (CCMA) e *commits* sem mudanças arquiteturais (CSMA).

Métrica	**Mann-Whitney-Wilcoxon**			**Teste Qui-Quadrado**	
	P-value	**Significância Estatística**	**Tendência Verificada**	***P-value***	**Significância Estatística**
LCOM4	$0,677$	NÃO	Manutenção	$6,8e^{-4}$	SIM
SC	$2,2e^{-16}$	SIM	Variação (Aumento)	$1,9e^{-4}$	SIM

se afirmar que os resultados obtidos com as métricas de coesão surpreendem, pois se esperava a redução em seus valores (aumento de coesão) para versões com mudanças na arquitetura. Por outro lado, a tendência esperada para os valores das métricas de coesão coletadas, junto às versões com mudanças na arquitetura, ocorreu no outro conjunto: Nas versões onde **não houveram alterações na arquitetura** registrou-se o **aumento da coesão** dos módulos/classes do projeto.

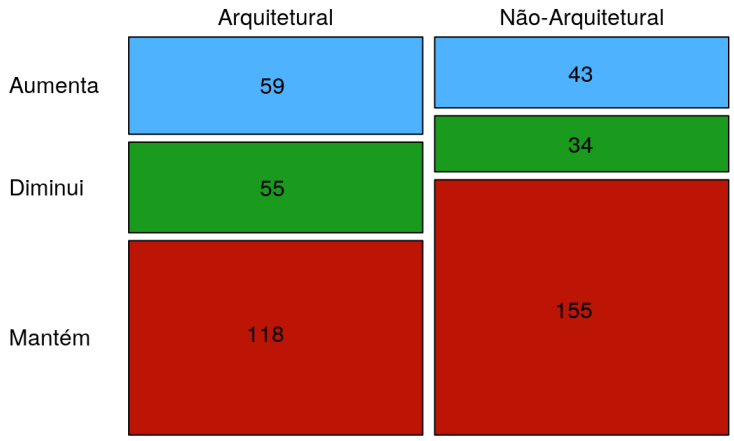
Considerando os tópicos definidos no capítulo anterior deste estudo e, ao analisar-se, individualmente, os resultados obtidos para cada métrica a partir desses tópicos, pode-se destacar para cada métrica de coesão:

- LCOM4: A maior parte dos *commits* arquiteturais, que registrou a **manutenção** de seus valores, se referia aos tópicos (XI) Relação entre Elementos, (VIII) Mecanismos de Sincronização e (VII) Protocolos de Comunicação (68 *commits* de um total de 118 - 58%). Nesse grupo de *commits* foram registrados 14 tópicos com esse tipo de comportamento.

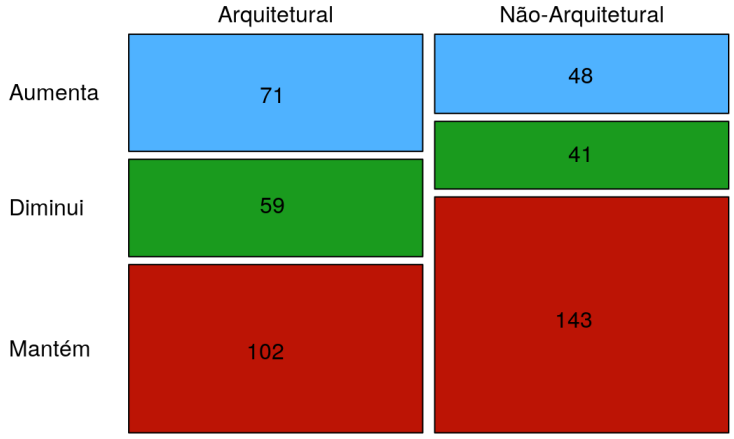
- SC: A maior parte dos *commits* arquiteturais, que registrou o **aumento** de seus valores, se referia aos tópicos (VIII) Mecanismos de Sincronização, (XI) Relação entre Elementos (44 de 71 - 62%). Nesse grupo de *commits*, foram registrados 14 tópicos com esse tipo de comportamento.

A análise destes resultados mostra que para a métrica **LCOM4**, os tópicos Relação entre Elementos, Protocolos de Comunicação, Elementos Arquiteturais de Conexão, Mecanismos de Acesso a Dados e Estruturas Globais de Controle apresentaram uma trajetória de manutenção dos valores obtidos para essa métrica. Já para a métrica **SC**, por sua vez, apresentou apenas o tópico Relação entre Elementos com tendência "divergente" (manutenção de valores) em relação à tendência majoritária verificada junto aos valores obtidos para essa métrica.

Ao se comparar os resultados obtidos por este e outros estudos envolvendo a relação entre métricas de coesão de software com esforço de manutenção, bugs e necessidade de *refactoring*, vê-se que tanto as métricas LCOM4 quanto SC diminuíam, um efeito diferente do aqui observado. A predição de defeitos, no entanto, seguiu a mesma lógica: para *commits* com alteração na arquitetura, o valor médio de SC aumenta. Os trabalhos de Li e Henry (1993), Thwin e Quah (2005) e Jin e Liu (2010), apontaram que quando essas



(a) Comparação das variações nos valores da métrica ΔLCOM4 entre *commits* com mudanças na arquitetura e *commits* sem mudanças na arquitetura.



(b) Comparação das variações nos valores da métrica ΔSC entre *commits* com mudanças na arquitetura e *commits* sem mudanças na arquitetura.

Figure 4.46: Comparação das variações médias para os valores das métricas de coesão estudadas para *commits* com e sem mudanças na arquitetura.

métricas aumentam os defeitos e os esforços de manutenção aumentam. É preciso destacar que Bengtsson (1998) apontou a métrica LCOM (não a LCOM4) como adaptável, para que seja capaz de medir a qualidade arquitetural. A Figura 4.46 ilustra esses números.

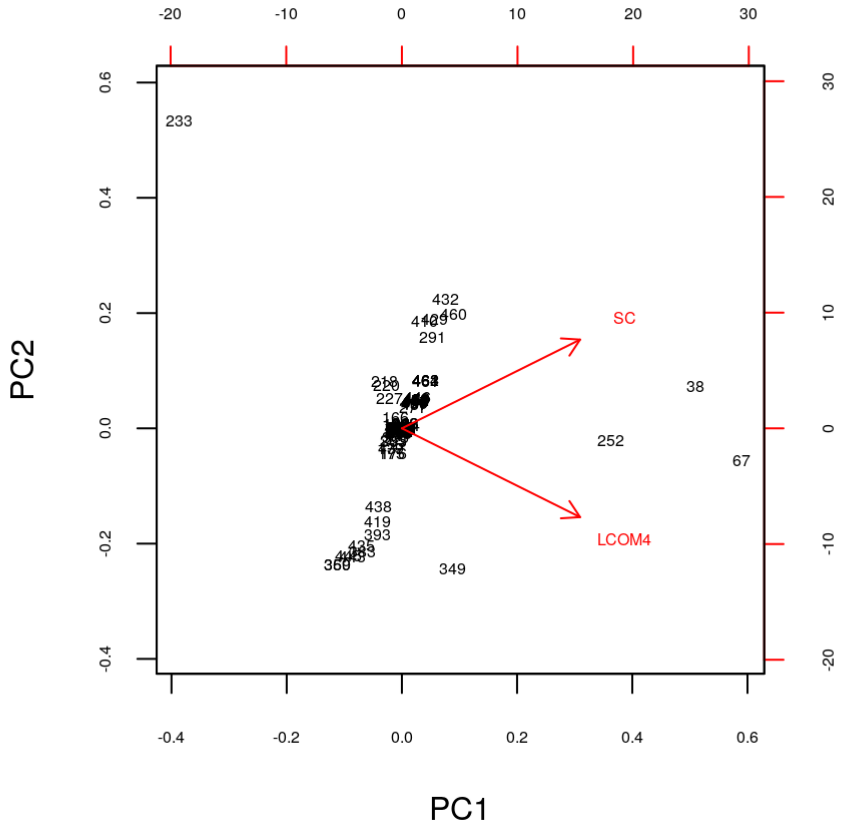


Figure 4.47: Análise de Componente Principal - Variações nos valores das Métricas de Coesão (normalizadas) por *commit*.

A Análise de Componente Principal (HOTELLING, 1933) realizada, considerando todos os valores das variações das métricas de coesão estudadas, apontou que os valores dessas métricas são dependentes entre si, uma vez que os coeficientes que as representam têm valores idênticos para a *PC1* e simétricos para *PC2* (ver Tabela 4.26), refletindo na posição de seus vetores conforme retrata o gráfico presente na Figura 4.47.

Em linhas gerais, os coeficientes apresentados na Tabela 4.26 sugerem uma ligação entre a variação das métricas de coesão estudadas (embora em sentidos opostos). A Figura 4.47, apresenta os vetores com as variações das métricas de coesão consideradas, a partir dos coeficientes de *PC1* e *PC2*. Dessa forma, é possível afirmar que, para caracterizar mudanças arquiteturais, é necessário observar as variações nos valores das métricas de coesão apresentadas neste trabalho.

Table 4.26: Tabela de Rotação da Análise de Componente Principal para variações nas métricas de coesão do projeto KDELibs.

Métrica	**PC1**	**PC2**
LCOM4	0.7071068	-0.7071068
SC	0.7071068	0.7071068

4.5 AMEAÇAS À VALIDADE

Nessa seção, foram elencados os fatores que podem comprometer a validade deste estudo. Essas ameaças foram organizadas em 4 conjuntos: validade de construto, validade interna, validade externa e confiabilidade.

Validade de Construto. Foi utilizada a avaliação do estudo de Motta, Souza e Sant'Anna (2018) para a definição do conjunto de *commits* com modificações arquiteturais. Para a construção do conjunto de *commits* sem mudança na arquitetura, foi realizado um sorteio parametrizado (dentro do mesmo período em que foram identificados os *commits* arquiteturais). Para as comparações que foram realizadas, avaliaram-se os *commits* sorteados se de fato não continham descrições de mudanças na arquitetura. Para os *commits* avaliados como contendo modificações arquiteturais, os *commits* apontados pelo estudo anterior foram reavaliados. Como o conceito de arquitetura é amplo e abarca vários aspectos do software, é possível que haja divergência de algum pesquisador da área de Engenharia de software quanto à classificação realizada. No entanto, o trabalho de Heesch, Avgeriou e Hilliard (2012) reforça a inexistência de consenso sobre o que deve ser descrito na documentação arquitetural.

Validade externa. A primeira ameaça identificada é o tipo de projeto de software. Foi adotado um projeto complexo. Outro ponto é o domínio: é um software central para o ecossistema do KDE. Um software de um domínio diferente pode apresentar menos preocupações sobre a arquitetura de software. No entanto, pode-se argumentar que o fenômeno pode ser estudado em qualquer tipo de software. Outra ameaça à validade externa é o fato que foi utilizado um software diferente da prática industrial. Para atenuar essa ameaça, usou-se o KDELibs, software que tem um longo período evolutivo e interage com vários outros projetos. Finalmente, ao usar apenas um projeto, reduz-se a generalização dos resultados. No entanto, não se pode afirmar que essas descobertas possam ser aplicadas para outras linguagens ou projetos de escala diferente.

Validade interna. A ameaça interna a ser considerada é a seleção dos *commits* analisados neste estudo, uma vez que eles foram identificados como contendo mensagens mencionando modificações na arquitetura por um estudo anterior. Minimizou-se essa ameaça revisando essas mensagens de *commit* e aplicando os critérios utilizados no estudo anterior.

Confiabilidade Para a coleta das métricas utilizou-se uma ferramenta livre, já testada e utilizada em vários outros estudos, mas não testada no âmbito desta pesquisa. Para mitigar essa ameaça, coletou-se métricas através do Analizo em versões do projeto escolhidas aleatoriamente, mas que não participaram deste estudo. Em seguida, fez-se o mesmo (com

as versões processadas pelo Analizo) usando uma versão de teste de um software comercial e ao final comparou-se os resultados obtidos por ambas as ferramentas.

4.6 CONCLUSÕES

Essa pesquisa mostrou que *commits* com mudança(s) na arquitetura de um projeto provocam variações nos valores de um conjunto de métricas de tamanho, complexidade, acoplamento e coesão diferentes das variações provocadas por mudanças onde a arquitetura não é modificada. Em relação às métricas de complexidade, apenas a métrica NPM não registrou diferenças estatisticamente significativas. Tanto os valores obtidos para ACCM quanto para NPA passaram pelo crivo do teste estatístico a que foram submetidos. Dessa forma, os resultados nos mostram que as métricas de código de tamanho e complexidade colaboram na caracterização de *commits* onde foram realizadas mudanças na arquitetura do projeto.

Viu-se em seguida que todas as métricas de acoplamento estudadas registraram essa diferenciação em relação aos *commits*, em que a arquitetura não foi modificada. Os valores obtidos para as métricas de acoplamento estudadas e para a métrica SC (também vista como métrica de acoplamento) foram estatisticamente significativas, o que não foi verificado com a métrica LCOM4. Os resultados obtidos também mostram que, as métricas de código de acoplamento e coesão podem colaborar na caracterização de *commits*, nos quais foram realizadas mudanças na arquitetura do projeto.

Essa pesquisa, ao trazer os resultados sobre o efeito de mudanças na arquitetura sob a óptica das métricas de código, ajuda a firmar que mudanças arquiteturais provocam diferentes efeitos sobre projetos de software. Foram registrados aumentos no (i) tamanho médio dos métodos, (ii) tamanho do código, (iii) número de métodos, (iv) tamanho do maior método, (v) complexidade ciclomática média dos métodos, (vi) número de conexões aferentes, (vii) resposta por classe e, (viii) complexidade estrutural. Por outro lado, foram registradas reduções no número médio de parâmetros. Percebeu-se também que não houve mudança no (i) número de atributos, (ii) número de atributos públicos, (iii) número de métodos públicos, (iv) grau de acoplamento entre objetos, (v) profundidade da árvore de herança e, (vi) número de classes filhas. Não foi possível apresentar conclusões sobre a falta de coesão dos métodos.

Chapter 5

AVALIANDO MODIFICAÇÕES ARQUITETURAIS SOB A PERSPECTIVA DE DESENVOLVEDORES E PESQUISADORES

5.1 INTRODUÇÃO

A literatura aponta que ter acesso à evolução arquitetural de um projeto de software e lógica dessa evolução melhora a rastreabilidade e a compreensão do mesmo (DUTOIT; PAECH, 2001; BACHMANN et al., 2005). Além disso, disponibilidade e documentação de informações sobre mudanças na arquitetura são fatores importantes para tornar eficaz a atividade de manutenção e para a avaliação de mudanças no projeto (BURGE; BROWN, 2008). Como os desenvolvedores tomam decisões, ao longo do desenvolvimento de software, ter conhecimento sobre a evolução da arquitetura do projeto é importante e pode evitar resultados indesejados (GURP; BOSCH, 2002; FARID; AZAM; IQBAL, 2011). Behnamghader et al. (2017) apontaram que as mudanças na arquitetura ocasionam de forma proeminente a decadência arquitetural, pois, aliada à ausência de dados empíricos, dificultam a compreensão dos desenvolvedores. Ding et al. (2014) já haviam sinalizado sobre a ausência de informações sobre a arquitetura de software e sua evolução (modificações).

Os estudos apresentados nos capítulos anteriores mostram que é possível atenuar o cenário de ausência de informações, uma vez que apresentam um estudo exploratório sobre a disponibilidade dessas informações em mensagens de *commit* e sobre o tipo de impacto no código causado por mudanças na arquitetura através de métricas de tamanho, complexidade, acoplamento e coesão. Esses estudos trazem informações sobre mudanças arquiteturais, tais como o perfil dos colaboradores de projetos de Software Livre que modificam a arquitetura do projeto, o tópico arquitetural mais modificado, o período quando mais ocorrem as mudanças na arquitetura, o tamanho das mensagens que descrevem essas mudanças e o nível de alteração dos arquivos/módulos em comparação com mudanças sem alteração na arquitetura.

Sobre as métricas, viu-se que, em geral, os valores das métricas de tamanho aumentam nas mudanças arquiteturais, de forma mais incisiva, que as demais mudanças (exceção

para **Número Médio de Parâmetros** e **Número de Atributos**); a mesma tendência se percebeu na métrica de **Complexidade Ciclomática Média por Método**. O **Número de Métodos Públicos** e **Número de Atributos Públicos** foram as outras métricas de complexidade analisadas, mas não apresentaram tendência de mudanças em seus valores. As métricas de coesão e acoplamento registraram uma tendência oposta. À exceção das **Conexões Aferentes por Classe** e das **Respostas por Classe**, a tendência das métricas de acoplamento foi de não variação nos valores de métricas. As métricas de coesão indicaram uma tendência de aumento nos seus valores registrados.

Assim, emerge a necessidade de apresentar os resultados desses estudos à comunidade de pesquisa em Engenharia de Software e à indústria, para investigar como as informações obtidas poderão ser utilizadas por esses pesquisadores e desenvolvedores, assim como se visualizam esses *insigths* nas suas práticas de pesquisa e desenvolvimento. Com esse fim, propõe-se um estudo quanti-qualitativo para caracterizar a percepção de pesquisadores, desenvolvedores e arquitetos frente aos resultados obtidos até aqui e aos problemas e desafios enfrentados por eles na pesquisa/prática de Engenharia de Software. A próxima seção (Seção 5.2) apresenta um conjunto de trabalhos relacionados este estudo.

5.2 TRABALHOS RELACIONADOS

Buscar a visão de desenvolvedores frente a tópicos relacionados à Arquitetura de Software não é uma proposta de pesquisa inédita. No entanto, a pesquisa em Engenharia de Software dedicou-se, até aqui, a outras nuances da Arquitetura de Software. Tratam-se de temas diferentes do estudo de mudanças na Arquitetura ao longo da evolução de projetos de Software Livre. A seguir, apresentamos alguns desses estudos.

Melo et al. (2016) estudaram as percepções de desenvolvedores sobre Arquitetura de Software. Através da aplicação de um *survey*, os pesquisadores concluíram que: (i) a arquitetura de software não conta com uma definição única, (ii) os desenvolvedores consideram importante documentar a arquitetura (embora não o façam), e (iii) a academia tem direcionado seus esforços num sentido que não contribui com a adesão dos desenvolvedores em seu uso. Rost et al. (2013) realizaram um *survey* envolvendo desenvolvedores, no qual trataram sobre a documentação da Arquitetura de Software. Como conclusão, viu-se que a documentação arquitetural disponível não é suficiente, pois não evolui com o software, frequentemente é contraditória em si mesma e não oferece uma interface consistente para a pesquisa de informações.

Ozkaya (2016) executou um *survey* com 50 participantes - desenvolvedores e pesquisadores - acerca da utilização dos conceitos de Arquitetura de Software em seus projetos de software. Ele apontou que os participantes reconheceram a Arquitetura de Software como uma importante fonte de informações frente à necessidade de compreensão de projetos de software, mas por outro lado eles visualizam apenas o aspecto estrutural da Arquitetura como sendo importante nesse processo, esquecendo-se dos demais tópicos arquiteturais. Parte deste estudo de doutorado, o uso de métricas também foi avaliado por desenvolvedores e contratantes de software usando um *survey* em estudo proposto por Kasunic (2006). Ele analisou, junto a desenvolvedores e contratantes de software, qual a relevância de métricas na tomada de decisões. Nas conclusões, ele relatou que

as métricas ainda são subutilizadas, pois menos da metade dos entrevistados coletam "frequentemente" medidas sobre produtos e serviços nos quais contribui. Mesmo entre os participantes que coletam "frequentemente" métricas, não há clareza sobre os uso dos dados coletados em prol da melhoria dos processos e produtos.

Como se percebe, embora a pesquisa em Engenharia de Software tenha colocado em pauta a percepção dos desenvolvedores sobre Arquitetura de Software e uso de métricas, não se tem disponível um estudo sobre a percepção desse público sobre mudanças Arquiteturais. A próxima seção apresenta o *design* do estudo realizado para coletar a percepção de desenvolvedores e pesquisadores sobre esse assunto.

5.3 DESIGN DO ESTUDO

Nesta subseção, é apresentado o design deste estudo. Inicialmente, é apresentada uma visão geral dele. Em seguida, são apresentadas as subquestões de pesquisa. Depois, explica-se quais os critérios para a definição do conjunto de participantes. A seguir é apresentado o questionário aplicado, mostrando como cada questão se relaciona com as subquestões de pesquisa que norteiam este trabalho. Em seguida, apresenta-se como foi realizada a coleta dos dados e por fim, é apresentado como os dados obtidos serão analisados.

5.3.1 Visão Geral

Este estudo está calcado na aplicação de um *survey* construído para coletar as impressões de pesquisadores da área de Engenharia de Software e desenvolvedores acerca da compreensão de projetos de software durante seu processo evolutivo, sob o aspecto arquitetural. Para isso foram apresentadas informações sobre mudanças arquiteturais obtidas através de mineração de repositórios - analisando mensagens de *commit* -, variações de métricas de tamanho, complexidade, acoplamento e coesão observadas durante essas mudanças, além de caracterizar a prática natural para o entendimento de projetos de software em sua atividade profissional: pesquisa ou desenvolvimento.

O processo de construção deste *survey* seguiu as ideias apresentadas por Kitchenham e Pfleeger (2002a, 2002b). Enquanto o primeiro trabalho (KITCHENHAM; PFLEEGER, 2002a) orientou o caráter do estudo aqui descrito, o segundo (KITCHENHAM; PFLEEGER, 2002b) guiou a construção do questionário utilizado neste *survey*. Trata-se de uma abordagem exploratória e de confirmação. Exploratória porque deseja revelar os métodos utilizados para a compreensão de projetos de software, sob o aspecto arquitetural, em cenários onde a documentação arquitetural é ausente ou desatualizada. Confirmacional porque deseja-se apresentar os resultados obtidos por esta pesquisa até aqui aos participantes e verificar se eles concordam com esses resultados.

Esse método foi escolhido porque tem sido largamente utilizado em estudos empíricos (PUNTER et al., 2003) e porque é importante a validação por parte de pesquisadores e desenvolvedores, potenciais consumidores dos resultados obtidos. Assim, foram apresentados, nesse *survey*, os resultados obtidos pela mineração de informações sobre mudanças arquiteturais e os impactos causados no código-fonte a partir dessas mudanças. Este *survey* apresentou

6 seções com 26 perguntas elaboradas segundo as recomendações de Kitchenham e Pfleeger (2002a, 2002b) e foi disponibilizado via Google Forms. Enquanto a definição do tamanho da amostra, e o design do estudo seguiram as recomendações presentes em (KITCHENHAM; PFLEEGER, 2002a), o formato, as questões abordadas e a disposição das questões foram influenciadas por (KITCHENHAM; PFLEEGER, 2002b). Foram convidados 50 participantes, entre desenvolvedores e pesquisadores. A esses participantes foi solicitado que eles convidassem novos participantes para que respondessem a esta pesquisa. Esse formulário ficou disponível por um período de 3,5 meses. Os dados obtidos foram analisados de forma quantitativa e qualitativa, buscando responder às subquestões de pesquisa deste estudo. Ao todo, o estudo contou com 75 participantes entre desenvolvedores e pesquisadores. As próximas subseções detalham cada aspecto deste estudo.

5.3.2 Questões de Pesquisa

As subquestões de pesquisa deste estudo foram projetadas para apresentar a pesquisadores da Engenharia de Software e desenvolvedores os resultados alcançados até aqui por esta pesquisa de doutorado e coletar suas impressões sobre tais resultados. Elas foram projetadas para (i) identificar as estratégias utilizadas pelos participantes no processo de compreensão de projetos de software, quando não há documentação adequada disponível, (ii) coletar a percepção dos participantes quanto ao impacto mensurável no código-fonte (através de métricas), após mudanças na arquitetura de um projeto e, (iii) coletar a visão dos participantes quanto ao uso de mensagens de *commit* contendo traços arquiteturais como artefatos auxiliares no processo de compreensão de mudanças na arquitetura do projeto.

Com o objetivo de analisar essas questões, foi formulada a seguinte questão de pesquisa: **RQ3: Desenvolvedores e Pesquisadores da Engenharia de Software podem se beneficiar no processo de compreensão de software ao usar informações sobre mudanças arquiteturais obtidas através de mensagens de *commit*?** Essa questão foi desmembrada em três subquestões de pesquisa, apresentadas a seguir:

RQ3.1: Quais as estratégias utilizadas por desenvolvedores e pesquisadores para compreender a arquitetura de um software durante sua prática profissional? Nesta subquestão, pergunta-se sobre quais as estratégias utilizadas por desenvolvedores e pesquisadores para compreender o projeto arquitetural em vigor de um projeto de software. Para isso, destinou-se uma seção do *survey* ao questionamento de como são recuperadas as informações arquiteturais de um projeto de software e sobre a suficiência dessas informações para a compreensão do projeto.

RQ3.2: Qual a percepção dos desenvolvedores e pesquisadores sobre o impacto no código após mudanças na arquitetura de um projeto? Nesta subquestão, foram apresentadas as variações de código identificadas através de métricas de tamanho, complexidade, coesão e acoplamento obtidas no estudo apresentado no Capítulo 4. Os participantes foram questionados, em uma seção do *survey*, sobre a expectativa de variação nas métricas de código provocadas por mudanças na arquitetura do projeto e, em seguida, confrontados com os resultados obtidos no estudo supramencionado (Capítulo 4).

RQ3.3: Qual a percepção dos desenvolvedores e pesquisadores frente aos traços arquiteturais presentes em mensagens de *commit* como artefato que auxilie na compreensão de mudanças na arquitetura?
A partir dessa subquestão, os participantes foram perguntados sobre os traços arquiteturais obtidos durante esta pesquisa de doutorado (estudo apresentado no Capítulo 3), relacionados a informações sobre mudanças na arquitetura do projeto. Destinou-se uma seção do *survey* para questionar a utilidade dessas informações como auxiliares na compreensão de mudanças arquiteturais, assim como a disponibilidade dessa informação nos projetos onde os desenvolvedores atuam ou atuaram. Para os pesquisadores, a questão se referiu aos projetos utilizados como objeto de estudo recentemente.

Na próxima subseção serão apresentados os critérios de escolha dos participantes.

5.3.3 Participantes

Para este estudo, foram escolhidos como participantes profissionais brasileiros e estrangeiros com experiência em desenvolvimento de software maior que 5 anos. Eles foram convidados por e-mail, com texto padrão cuja única variação é o nome do participante. Estimou-se que a maior parte dos participantes seriam brasileiros, pois os convites foram enviados a partir das interações construídas ao longo da trajetória profissional e acadêmica do autor desse trabalho. Buscando diversificar ainda mais o perfil dos participantes, foi solicitado aos respondentes que convidassem outros desenvolvedores para que participassem desta pesquisa[1].

Quanto ao convite enviado aos pesquisadores, foram escolhidos aqueles que atuam na área de Engenharia de Software e estudam fenômenos ligados à evolução de software e/ou à arquitetura de software. O público foi limitado a pesquisadores com experiência mínima de 5 anos de atuação, ou que já tenham atuado com desenvolvimento de software por um período maior que 5 anos ou que estejam em processo final de doutoramento, estudando arquitetura de software ou elementos a ela relacionados.

Foram convidados, diretamente, 25 pesquisadores e 41 desenvolvedores. Trata-se de uma amostra não-estratificada e escolhida dentro dos grupos de interação do autor deste trabalho (amostra por conveniência).

5.3.4 Questionário

Para atender a esta pesquisa, foi construído um questionário buscando responder às subquestões de pesquisa definidas neste estudo e descritas na Seção 5.3.2. O processo de construção seguiu as recomendações de Kitchenham e Pfleeger (2002b). A definição do instrumento de coleta, o formato do questionário, a definição do tipo de questão aplicada, a presença da seção de perfil dos participantes, e a proposição e avaliação das perguntas (antes da aplicação do *survey*) foram inspiradas nessa referência.

Após a construção da versão preliminar do questionário, foi realizado um estudo piloto com dois desenvolvedores e dois pesquisadores dentro do perfil desejado para a pesquisa, buscando verificar se as perguntas foram apresentadas de forma clara e objetiva,

[1]Para os participantes convidados indiretamente, não foi exercido o critério de experiência mínima.

além de medir o tempo médio de participação. Durante o piloto, o tempo de resposta para cada pergunta foi monitorado e ao final contabilizou-se o tempo total. Após o estudo piloto, os respondentes foram questionados sobre a clareza de cada questão, e sobre o entendimento obtido. Como resultado, foi necessário adaptar perguntas e mudar a ordem das seções. Por limitação da ferramenta utilizada quanto à tabulação e marcações no texto, foram inseridas figuras em locais onde tivemos relatos de dificuldade de compreensão de textos necessários para respostas em perguntas a eles relacionadas. Os questionários são direcionados para cada público: pesquisadores e desenvolvedores. Eles contêm perguntas parecidas, mas adaptadas às diferentes atuações profissionais.

Apesar de preocupados com questões ligadas à fadiga que poderiam ser ocasionadas por um questionário extenso, estimou-se que a participação dos desenvolvedores e pesquisadores exija entre 20 e 30 minutos. Essa informação consta no convite para participação no estudo, uma orientação necessária para que obtenhamos uma maior adesão, segundo Smith et al. (2013). Outra medida, que se tomou para diminuir a influência de fadiga sobre os participantes, durante a participação, foi inserir a seção de perfil do respondente como última a ser respondida (SEAMAN, 1999).

A Tabela 5.1 relaciona as subquestões de pesquisa às perguntas contidas no questionário produzido. Para cada pergunta, é apresentada a questão de pesquisa relacionada, a seção onde foi apresentado, o modelo de questionário (Pesquisador - P ou Desenvolvedor D) e o tipo de pergunta: Múltipla Escolha (ME), Seleção Múltipla (SM), Resposta Curta (RC) e Resposta Longa (RL). As perguntas de múltipla escolha (ME) são fechadas, onde o participante deve escolher apenas uma opção de resposta dentre as disponíveis, enquanto as perguntas de Seleção Múltipla (SM) oferecem várias respostas disponíveis e o participante pode escolher mais que uma delas. As perguntas com respostas curtas (RC) devem ser respondidas com períodos textuais curtos, enquanto as perguntas de respostas longas (RL) suportam respostas com a construção de várias frases organizadas em parágrafos.

Para permitir a participação de pesquisadores e desenvolvedores não-brasileiros, foram disponibilizadas versões em inglês dos dois modelos de questionários. Para facilitar a identificação das perguntas em cada questionário, as perguntas marcadas em cinza claro na Tabela 5.1 constam no questionário de pesquisadores, em cinza escuro constam no questionário de desenvolvedores e as linhas sem marcação de cor referem-se às perguntas presentes em ambos os questionários.

O questionário está organizado em 6 seções. A primeira seção possui apenas o Termo de Livre Consentimento Assistido. A sua aceitação é condição essencial para a participação na pesquisa. Já a segunda seção (caracterização) apresenta as perguntas onde o participante irá relatar como, durante a sua experiência profissional, lida com a evolução de projetos de software, sua documentação e como observa a arquitetura desses projetos. Essa seção reúne quatro perguntas, personalizadas para cada tipo de participante (desenvolvedor ou pesquisador). A Seção 3, por sua vez, reúne quatro perguntas relacionadas à **RQ3.1**. Através delas, busca-se saber sobre a documentação disponível em projetos que ele participou, quais estratégias e ferramentas utilizadas nesse processo e quais aspectos (tópicos arquiteturais) são observados nesse processo

Table 5.1: Descrição das Perguntas presentes no questionário: Seção; Questão de Pesquisa (RQ); Pergunta; Tipo de Pergunta (TP) e Modelo de Questionário (MQ).

Seção	RQ	Pergunta	TP	MQ
Apresentação	–	Termo de Livre Consentimento Assistido	ME	P,D
Caracterização	–	É uma prática comum em seus projetos de pesquisa estudar a evolução dos projetos de software?	ME	P[1]
Caracterização	–	É uma prática comum na sua empresa/equipe inserir desenvolvedores ao longo da evolução dos projetos (considere sua empresa atual ou última experiência, caso não estejas atuando com desenvolvimento nesse momento)?	ME	D[2]
Caracterização	–	Nos projetos utilizados em sua pesquisa, existe algum tipo de treinamento/ capacitação (mesmo on-line) sobre o projeto para novos desenvolvedores/pesquisadores? Se sim, fale sobre ele.	RL	P[1]
Caracterização	–	Existe um treinamento/capacitação sobre o projeto em que os desenvolvedores estão sendo inseridos? Se sim, fale sobre como ele ocorre.	RL	D[2]
Caracterização	–	Caso não haja treinamento/capacitação sobre o projeto no ato da inserção de novos participantes, como você procedeu para compreender o projeto? Você tem um processo sistemático para isso? Fale sobre ele.	RL	P[1]
Caracterização	–	Caso não haja treinamento/capacitação sobre o projeto no ato da inserção de novos participantes, como se dá o processo de compreensão do projeto?	RL	D[2]
Caracterização	–	Nos projetos que você utiliza em seus estudos, há documentos especificamente usados para registrar a arquitetura do software (e sua evolução)? Quais?	RL	P[1]
Caracterização	–	Há documentos usados no projeto para registrar a arquitetura do software (e sua evolução)? Quais?	RL	D[2]
3	RQ3.1	Na sua pesquisa, houve projetos onde você precisou entender o código por não ter acesso à documentação de análise? Se sim, fale sobre sua experiência.	RL	P[1]
3	RQ3.1	Houve projetos onde você precisou entender o código por não ter participado da análise e ter ingressado na equipe durante a produção/evolução? Se sim, fale sobre sua experiência.	RL	D[2]
3	RQ3.1	Quais as estratégias adotadas por você para compreender o projeto nessas situações?	RL	P,D
3	RQ3.1	Você utiliza alguma ferramenta no processo de compreensão dos projetos utilizados em sua pesquisa? Fale rapidamente sobre essa ferramenta.	RL	P[1]
3	RQ3.1	Existe alguma ferramenta utilizada por você no processo de compreensão? Fale rapidamente sobre essa ferramenta.	RL	D[2]
3	RQ3.1	Esse método/ferramenta revela quais aspectos arquiteturais do projeto?	SM	P,D
4	RQ3.2	Os resultados obtidos por nossa pesquisa correspondem às suas expectativas (sim/não)? Justifique.	RL	P,D
4	RQ3.2	Você concorda que esses resultados se repetem na maioria dos projetos (sim/não)? Justifique.	RL	P,D
4	RQ3.2	Você concorda que esses resultados se repetem nos projetos em que você pesquisa (sim/não)? Justifique.	RL	P[1]
4	RQ3.2	Você concorda que esses resultados se repetem nos projetos em que você desenvolve (sim/não)? Justifique.	RL	D[2]
5	RQ3.3	Acima, encontram-se 5 mensagens com informações sobre mudanças na arquitetura de um projeto. Quais elementos arquiteturais podem ser percebidos nessas mensagens?	SM	P,D
5	RQ3.3	Essas mensagens lhe ajudariam a compreender as mudanças arquiteturais realizadas (sim/não)? Justifique.	RL	P,D
5	RQ3.3	Esse tipo de mensagem existe em seu projeto/controlador de versão? Por favor, nos dê um exemplo.	RL	P,D
5	RQ3.3	Você concorda que localizar esse tipo de mensagem ajuda a entender as mudanças na arquitetura do projeto ao longo de sua evolução (sim/não)? Justifique.	RL	P,D
Perfil	–	Qual é o seu nome?	RC	P,D
Perfil	–	Em qual país você trabalha atualmente?	RC	P,D
Perfil	–	Você gostaria que entrássemos em contato para uma devolutiva dos resultados dessa pesquisa?	ME	P,D
Perfil	–	Em quais linguagens de programação você tem experiência?	RL	P,D
Perfil	–	Qual a sua formação acadêmica (marque todas as que considerar necessárias)?	SM	P,D
Perfil	–	Qual a sua função atual na empresa onde atua?	SM	P,D
Perfil	–	Há quanto tempo você desenvolve/pesquisa software e em quais funções você atuou? Exemplos: Programador (5 anos), pesquisador (10 anos).	RL	P[1]
Perfil	–	Há quanto tempo você desenvolve software e em quais funções você atuou? Exemplo: Programador (5 anos), pesquisador (10 anos).	RL	D[2]

[1] Pergunta presente apenas no questionário de **pesquisadores**.

[2] Pergunta presente apenas no questionário de **desenvolvedores**.

de compreensão. Essa seção conta com o suporte de dois *links* de apoio. O primeiro[2] indica um artigo que define tópico arquitetural e fala sobre a mineração de informações arquiteturais (MOTTA; SOUZA; SANT'ANNA, 2018) e o outro apresenta um resumo sobre mudanças arquiteturais e os tópicos relacionados no estudo supracitado.

A Seção 4 traz consigo perguntas relacionadas à **RQ3.2**. Essa seção conta com três perguntas, onde são apresentadas as variações obtidas nos valores das métricas estudadas no Capítulo 4 e o participante é instigado a comparar com os resultados que ele possui no seu dia-a-dia de trabalho e com a sua percepção de uma forma geral. Já a Seção 5 traz consigo quatro perguntas relacionadas à **RQ3.3**. Nessas perguntas, os participantes foram questionados acerca das informações sobre arquitetura de software mineradas por esta pesquisa e apresentadas no Capítulo 3. Ainda nessa seção, eles foram convidados a refletir sobre o uso desse tipo de técnica como suporte à compreensão de projetos de software ao longo de sua evolução. Por fim, a Seção de Perfil apresenta as perguntas necessárias à caracterização dos participantes desta pesquisa. O Apêndice E contém *printscreens* de cada uma das versões do formulário desenvolvido. A próxima subseção apresenta o processo de coleta de dados.

5.3.5 Coleta de Dados

Conforme afirmado na subseção 5.3.3, os participantes foram convidados através de um e-mail padrão, onde foram instados a preencher o formulário disponibilizado na ferramenta Google Forms o qual reflete as seções descritas na subseção 5.3.4. Essa ferramenta foi escolhida por facilitar tanto a coleta (está disponível na web e não requer muitos recursos adicionais dos participantes) quanto à extração dos dados obtidos (por permitir a extração em planilhas em formatos de dados conhecidos).

Esse formulário aceitou respostas por 105 dias divididos em dois períodos: Entre 7 de Abril de 2020 e 6 de Julho de 2020 e entre 15 de Dezembro de 2020 e 30 de Dezembro de 2020. Ele voltou a receber respostas porque durante o primeiro período de coleta, foram recebidas poucas respostas de participantes envolvidos em projetos de Software Livre. Assim sendo, durante o segundo período, foram convidados apenas desenvolvedores que atuam em projetos de Software Livre atualmente.

5.3.6 Análise

As subquestões de pesquisa apresentadas na Seção 5.3.2 foram analisadas por meio de estatística descritiva e análise qualitativa. Frente aos resultados apresentados, os dados foram comparados entre os diferentes perfis de respondentes. Para as perguntas abertas (respostas curtas e longas), as respostas obtidas foram codificadas observando técnicas de Teoria Fundamentada nos Dados (CHARMAZ, 2006; CORBIN; STRAUSS, 2008) para a codificação inicial e focalizada.

Durante a codificação inicial, são identificadas as expressões que respondem ao questionamento realizado ("o que os dados sugerem ou afirmam?"(GLASER, 1978)). Essa etapa da codificação é realizada com a análise de "palavra por palavra", "linha a linha". Já a

[2]https://motta-tiago.github.io/doctorate/

codificação focalizada busca identificar os códigos iniciais mais significativos. Durante esse processo, decide-se quais códigos permitem uma compreensão analítica de modo a se juntarem com outros de forma a compartilharem uma categoria.

5.4 RESULTADOS

5.4.1 Visão Geral

Conforme explicado na subseção de Coleta de dados (5.3.5), foram realizadas duas coletas utilizando o mesmo formulário em períodos distintos. No primeiro período, foram disparados 66 convites por e-mail (e solicitado aos respondentes que estendessem o convite para pessoas com o perfil desejado). O autor desta tese recebeu (em cópia) a extensão desse convite a mais 60 pessoas - totalizando 126 convites. Nesse primeiro momento, o formulário disponibilizado registrou 65 participações.

O formulário foi reaberto, por um período de 15 dias, visando a participação de desenvolvedores de projetos de Software Livre, uma vez que na primeira coleta, poucos participantes registraram esse perfil. Assim, na segunda coleta foram convidados 12 desenvolvedores de projetos de Software Livre e foram registradas 9 participações. Ressalte-se que os desenvolvedores convidados são participantes de projetos de Software Livre espalhados pelo mundo e que já interagiram academicamente ou profissionalmente com algum dos membros do grupo de pesquisa aSide[3].

Em meio aos participantes, foi registrada a participação de pessoas com os mais distintos perfis profissionais relacionados ao desenvolvimento de software. Participaram desta pesquisa analistas, arquitetos, engenheiros de software, gerentes de projetos, líderes técnicos, pesquisadores, professores, programadores e testadores. As próximas seções detalham os resultados obtidos nessa pesquisa.

5.4.2 Caracterização dos Participantes

Conforme já informado, o questionário registrou 74 participações, sendo 19 pesquisadores e 55 desenvolvedores. Eles estão espalhados por onze países de 3 continentes. A maioria deles se concentrou no Brasil (57), sendo 14 pesquisadores e 43 desenvolvedores, mas também foram registrados participantes dos seguintes países: Alemanha (1 desenvolvedor), Bélgica (1 pesquisador e 1 desenvolvedor), Canadá (1 pesquisador e 2 desenvolvedores), Colômbia (1 pesquisador), Hungria (1 desenvolvedor), Irã (1 desenvolvedor), Portugal (1 desenvolvedor), República Tcheca (1 desenvolvedor), Tailândia (1 pesquisador) e EUA (1 pesquisador e 4 desenvolvedores). Dentre os participantes há 15 mulheres (10 desenvolvedoras e 5 pesquisadoras) e 59 homens (45 desenvolvedores e 14 pesquisadores). Conforme afirmado, na seção de design do estudo, não se trata de uma amostra estratificada.

Apenas 1 pesquisador declarou não programar em nenhuma linguagem. Dentre os demais, todos programam em Java. Também destaca-se o uso de C (12 pesquisadores). Também foram citadas por esse grupo as linguagens C++, Python, Php, JavaScript, C#, Pascal, Objective C, R, Basic, Cobol, Haskell, Object Pascal, Visual Basic, Ada,

[3]http://aside.dcc.ufba.br/

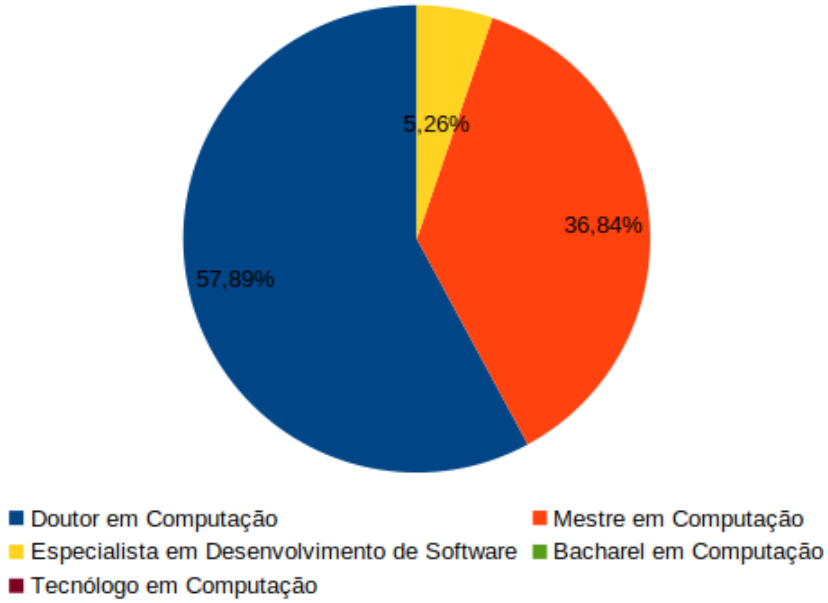
Clipper, Delphi, Fortran, Lisp, ML e Prolog. Dentre os desenvolvedores, as preferências mudaram pouco: Java é a linguagem utilizada por 39 deles, enquanto C tem a adesão de 22 desenvolvedores. Nesse grupo de participantes também foram citadas as seguintes linguagens: JavaScript, Python, PHP, C#, C++, Ruby, SQL, Dart, Delphi, Typescript, Visual Basic, NodeJS, R, Shell Script, ADVPL, Android, Angular, Arduino, ASP, Centura, .Net, Go, Objective-C, Object Pascal, Perl, React, Scala, Swift, Windev. Percebe-se que há uma diversidade de linguagens utilizadas entre os pesquisadores e desenvolvedores participantes desta pesquisa. A Tabela 5.2 apresenta o panorama das linguagens utilizadas pelos participantes desta pesquisa.

Table 5.2: Lista de linguagens de programação utilizadas pelos participantes (desenvoolvedores e pesquisadores) desta pesquisa. Nesta questão, cada participante poderia escolher mais que uma linguagem em sua resposta.

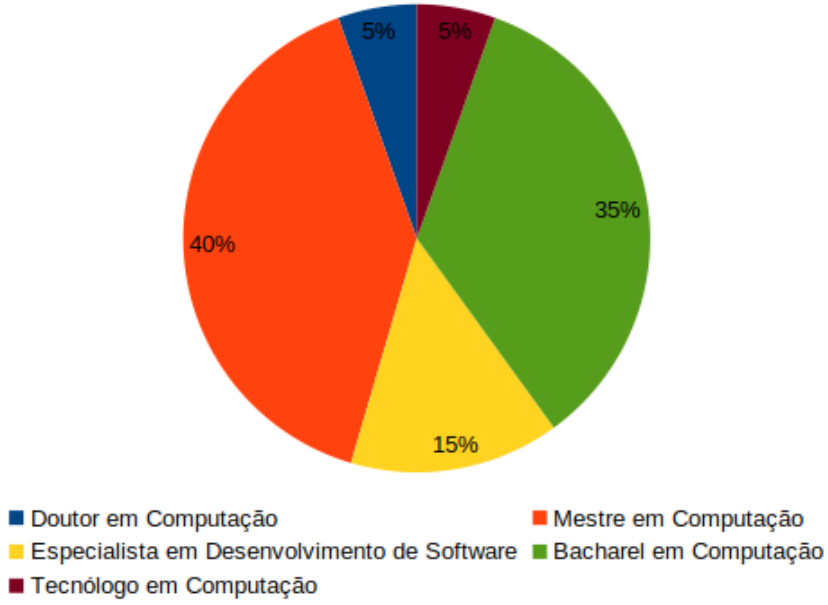
Linguagem(ens)	**Total**	**Desenvolvedores**	**Pesquisadores**
Java	58	39	19
C	33	23	10
Python	30	22	8
Javascript	24	19	5
C++	23	14	9
PHP	21	14	7
C#	18	13	5
Ruby	12	12	0
Shell	5	5	0
VB	5	3	2
Delphi	4	3	1
Objective-C	4	1	3
R	4	2	2
SQL, Typescript	4	4	0
Dart	3	3	0
Pascal	3	0	3
Android, Go, NodeJS	2	2	0
Cobol, Haskell, SysML	2	0	2
Object Pascal	2	1	1
.NET, ADVPL, Angular, Arduino, ASP, Centura, ETL, Perl, React, Scala, Swift, Windev	1	1	0
AADL, ADA, Clipper, Fortran, LISP, Prolog	1	0	1

Em relação à titulação dos participantes, entre os pesquisadores todos possuem mestrado ou doutorado em Ciência da Computação. Dentre os desenvolvedores que participaram desta pesquisa, por outro lado, apenas 4 possuem doutorado em Ciência da Computação, 22 possuem mestrado em Ciência da Computação e 10 possuem especialização *lato sensu* em Desenvolvimento de Software. Nesse grupo apenas 3 participantes não possuem graduação na área de Ciência da Computação e afins (tratam-se de tecnólogos em desenvolvimento de sistemas). Como se vê, tratam-se de participantes titulados na área de Ciência da Computação e/ou Desenvolvimento de Software. A Figura 5.1 apresenta de forma sumarizada a titulação máxima declarada pelos participantes deste estudo.

Em relação à(s) função(ões) em que os participantes atualmente exercem, 15 pesquisadores relatam que também são professores (além de pesquisadores). Nesse grupo de participantes, também foram citadas como área de atuação: programador, arquiteto de software, analista de sistemas, engenheiro de software e gerente de projetos. Para o grupo de desenvolvedores,



(a) Titulação máxima declarada pelos pesquisadores participantes deste estudo.



(b) Titulação máxima declarada pelos desenvolvedores participantes deste estudo.

Figure 5.1: Comparação da titulação máxima dos participantes deste estudo, comparando pesquisadores (19 participantes) e desenvolvedores (55 participantes).

além as ocupações citadas pelos pesquisadores, foram citados: testador de software e líder técnico. Como se percebe, tratam-se de profissionais que atuam no desenvolvimento de software em vários papéis. Mais da metade dos desenvolvedores (30) exercem apenas um papel nos projetos em que trabalham (em funções de engenheiro de software, analista ou gerente de projetos). Um cenário diverso do verficado entre os pesquisadores, onde apenas 7 exercem apenas uma função em seu ambiente de trabalho (professor ou pesquisador). A Figura 5.2 apresenta as ocupações declaradas por cada participante deste estudo. Na parte a, são apresentadas as funções "exclusivas" daqueles participantes que declararam exercer apenas uma função atualmente. Já a parte b apresenta o quantitativo geral de funções exercidas por todos os participantes.

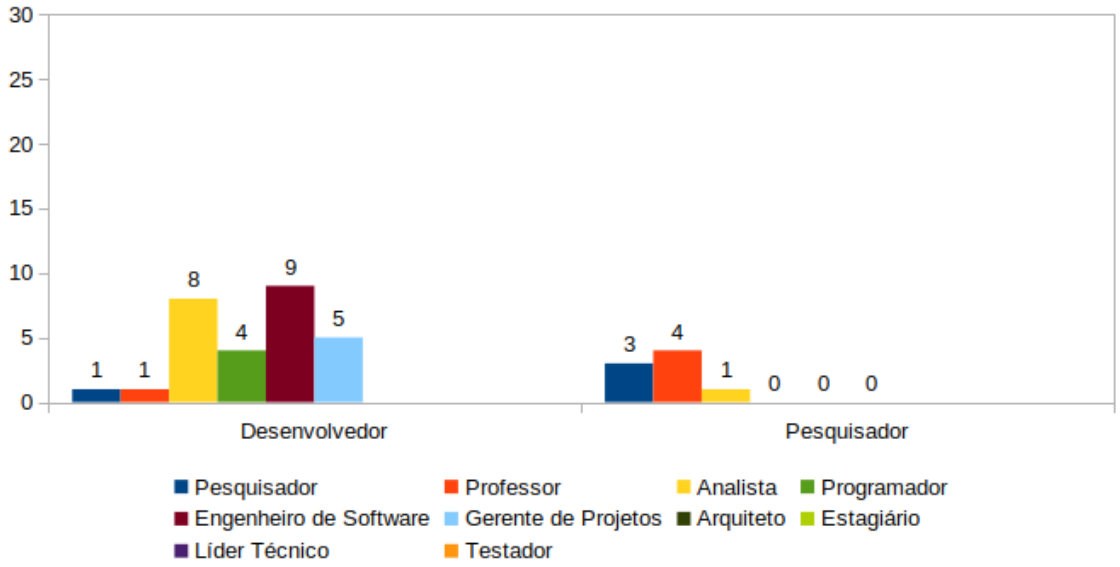
Sobre a experiência relatada entre os participantes, elaborou-se uma escala intervalar, com 3 pontos: menor que 5 anos, entre 5 e 10 anos e maior que 10 anos. No grupo de pesquisadores que participaram da pesquisa, o participante com maior tempo de pesquisa em Engenharia de Software traz consigo a experiência de 40 anos. Nesse grupo registrou-se apenas um participante com experiência menor que 5 anos. Os demais respondentes foram organizados em dois grupos com 9 participantes: aqueles com experiência entre 5 e 10 anos e aqueles com experiência superior a 10 anos. O tempo médio de experiência entre os pesquisadores foi de 11,9 anos.

Dentre os desenvolvedores, por outro lado, obteve-se um cenário mais heterogêneo. O desenvolvedor mais experiente possui 30 anos de desenvolvimento de software profissional. Entre os intervalos definidos, obteve-se a participação de 14 desenvolvedores com experiência menor que 5 anos, 27 desenvolvedores com experiência profissional em programação entre 5 e 10 anos e, por fim, 14 desenvolvedores possuem tempo de experiência maior que 10 anos. O tempo médio de experiência entre os desenvolvedores participantes foi de 8,32 anos. Percebe-se dois grupos distintos entre os participantes. Entre os desenvolvedores há um número considerável (29% do total) de desenvolvedores com experiência inferior a cinco anos de desenvolvimento, por outro lado, contou-se com desenvolvedores com até 30 anos de experiência entre os respondentes. Entre os pesquisadores, no entanto, vê-se um tempo médio de experiência mais elevado (3,6 anos superior em relação aos desenvolvedores) e apenas um pesquisador com tempo de experiência menor que 5 anos. A Figura 5.3 apresenta o tempo de experiência dos participantes deste estudo, comparando pesquisadores e desenvolvedores. A próxima seção apresenta os cenários relatados pelos participantes desta pesquisa em relação aos projetos que desenvolvem ou utilizam em suas pesquisas.

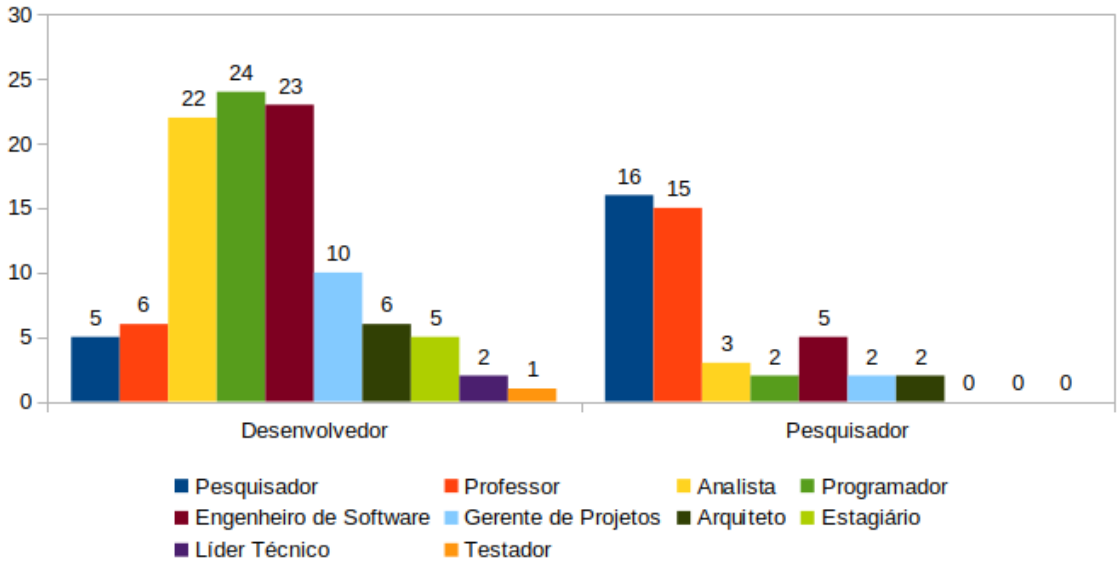
5.4.3 Caracterização dos Projetos desenvolvidos/pesquisados pelos participantes

A segunda seção do questionário utilizado por este estudo versa sobre a caracterização dos projetos desenvolvidos/pesquisados pelos participantes desta pesquisa. Foram realizadas 4 perguntas no questionário aplicado, sobre as quais obteve-se os resultados a seguir.

Para o grupo de pesquisadores que participaram desta pesquisa, apenas 4 pesquisadores (21%) reportaram não estudarem corriqueiramente a evolução de projetos de software. Para os outros 15 pesquisadores, esse é um assunto recorrente na pauta de seus projetos. Para os desenvolvedores, perguntou-se sobre a adesão de novos desenvolvedores ao longo

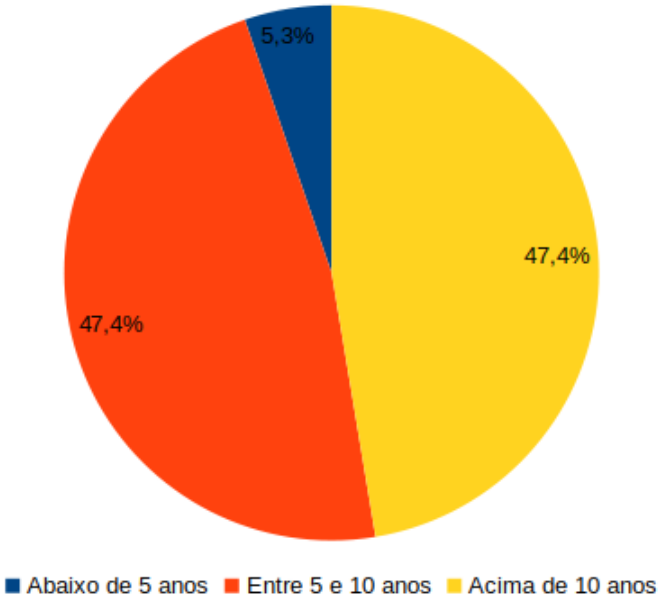


(a) Funções exercidas pelos participantes (desenvolvedores x pesquisadores) deste estudo que exercem apenas uma função profissional.

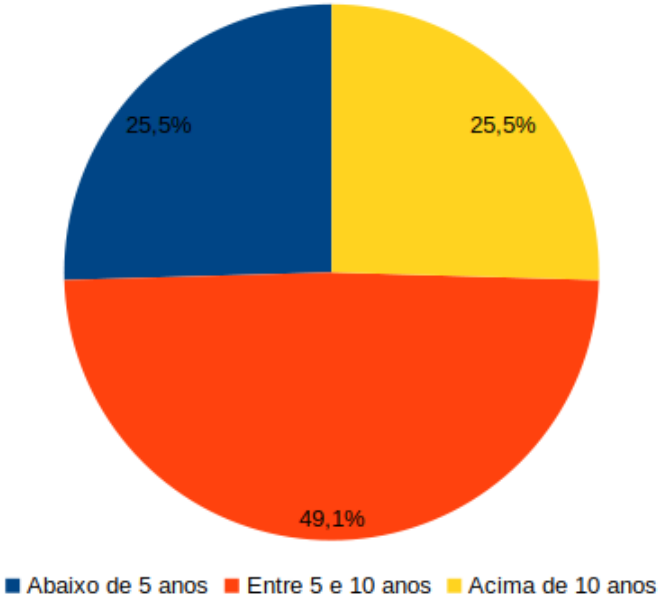


(b) Funções exercidas pelos participantes deste estudo (desenvolvedores x pesquisadores).

Figure 5.2: Funções exercidas pelos participantes deste estudo.



(a) Tempo de experiência declarado pelos pesquisadores participantes deste estudo.



(b) Tempo de experiência declarado pelos desenvolvedores participantes deste estudo.

Figure 5.3: Comparação entre o tempo de experiência dos participantes deste estudo, comparando pesquisadores (19 participantes) e desenvolvedores (55 participantes).

da evolução dos projetos enquanto prática comum na sua equipe/empresa. O relato desses participantes é que a adesão de desenvolvedores ao longo do projeto é corriqueira, pois 47 participantes desse grupo relataram esse cenário (85%). Apenas 8 desenvolvedores reportaram um cenário diferente.

Quando questionados sobre a existência de treinamento para ingressantes, 16 pesquisadores informaram que não há treinamento. Os demais (três) citaram a apresentação do projeto e a apresentação dos casos de teste como sendo a estratégia de treinamento adotada. Para os desenvolvedores, 25 relataram não haver nenhum tipo de treinamento. Um número considerável de desenvolvedores relatou a apresentação do projeto (24) como estratégia de treinamento. Também foram citadas como estratégias de treinamento a apresentação da equipe do projeto, apresentação do padrão de desenvolvimento utilizado, as tecnologias utilizadas e a atribuição de tarefas de baixa complexidade. Percebe-se um cenário diferente entre os dois grupos. Enquanto para os pesquisadores o treinamento não é priorizado, para os desenvolvedores alguma estratégia de apresentação do projeto é adotada, segundo o relato da maioria dos participantes. A Tabela 5.3 apresenta um sumário do tipo de treinamento oferecido a ingressantes nos projetos (pesquisadores e desenvolvedores).

Table 5.3: Estratégias de treinamento adotadas para novos desenvolvedores e pesquisadores.

Grupo	**Estratégia**	**Quantidade**	**% Grupo**
Pesquisadores	Não há	16	84,2%
	Apresentação do Projeto	2	10,5%
	Apresentação de Casos de Teste	1	5,3%
Desenvolvedores	Não há	25	45,5%
	Apresentação do Projeto	13	23,7%
	Apresentação da Visão Geral do projeto	7	12,7%
	Apresentação de Padrões de Desenvolvimento	6	10,9%
	Apresentação da Equipe	1	1,8%
	Apresentação das Tecnologias utilizadas	1	1,8%
	Atribuição de Tarefas de baixa complexidade	1	1,8%
	Uso de Ferramentas	1	1,8%

Em seguida foi questionado aos participantes como eles fizeram para compreender os projetos que pesquisam/desenvolvem nos cenários onde eles não participaram desde o início do projeto. Nessa pergunta, o participante poderá indicar mais que uma estratégia utilizada para a compreensão do projeto, caso deseje. Os pesquisadores citaram como a principal fonte de informação o código-fonte. Cinco pesquisadores analisaram o código-fonte, enquanto três recorreram a engenharia reversa. As outras estratégias citadas foram a programação em par (com desenvolvedores mais experientes), a mineração de artefatos e o estudo de documentação - uma citação cada uma. Seis pesquisadores revelaram não utilizar nenhuma estratégia (32%).

Para os desenvolvedores as estratégias foram parecidas. Novamente a estratégia mais recorrente estava relacionada ao código-fonte: Análise de código (12). Outras estratégias apontadas pelos participantes desse grupo foram pedir ajuda a desenvolvedores experientes, apresentação do projeto, análise de *bug report*, adesão à programação em

pares, fazer reunião com toda a equipe, busca por documentação disponível, simulação de desenvolvimento e aprender sozinho. Observa-se pelas respostas coletadas que o código-fonte permanece como principal fonte de informações para desenvolvedores/pesquisadores ingressantes em equipes de projetos de software. Isso denota uma grande preocupação com os aspectos estruturais da arquitetura dos projetos de software, em detrimento das demais visões arquiteturais. Em busca dessas informações, eles utilizam-se de todo o tipo de informação disponível no projeto, incluído aí o conhecimento não documentado sobre a arquitetura que emerge na programação em pares, reuniões com a equipe e no pedido de ajuda a desenvolvedores mais experientes. Vê-se também a busca em fontes de informações não-estruturadas tais como *bug reports*, mineração de artefatos e estudo da documentação disponível. Quanto ao grupo de desenvolvedores que não utilizam nenhuma estratégia, o resultado foi parecido com os pesquisadores: 17 desenvolvedores registraram essa opção (31% dos desenvolvedores). A Tabela 5.4 apresenta as estratégias utilizadas por pesquisadores e desenvolvedores para compreender os projetos que pesquisam/desenvolvem.

Table 5.4: Estratégias adotadas por desenvolvedores e pesquisadores para a compreensão de projetos de software.

Grupo	**Estratégia**	**Quantidade**	**% Grupo**
Pesquisadores	Sem estratégia	6	31,6%
	Análise de Código	5	26,3%
	Engenharia Reversa	3	15,8%
	Mineração de Artefatos	2	10,5%
	Ajuda de Desenvolvedor experiente	1	5,2%
	Análise de Comentários de Código	1	5,2%
	Análise de *Feature model*	1	5,2%
	Estudo de Documentação	1	5,2%
	Informalmente	1	5,2%
	Leitura de artigos sobre a tecnologia empregada	1	5,2%
	Programação em par	1	5,2%
Desenvolvedores	Sem estratégia	17	30,9%
	Análise de Código	12	21,8%
	Ajuda de Desenvolvedor experiente	10	18,2%
	Apresentação do Projeto	8	14,5%
	Programação em par	7	12,7%
	Busca por documentação disponível	3	5,5%
	Reunião da Equipe	3	5,5%
	Consulta ao *bug report*	1	1,8%
	Atribuição de tarefas com baixa complexidade	1	1,8%
	Simulação de desenvolvimento	1	1,8%

Por fim, os participantes foram perguntados sobre a disponibilidade de documentação nos projetos que desenvolvem/realizam suas pesquisas. No grupo de pesquisadores, 8 não dispõem de nenhum tipo de documentos relacionados à arquitetura dos projetos (42% dos pesquisadores). Por outro lado, foram reportadas a existência de descrição arquitetural usando AADL, diagramas estruturais da UML, documentos descrevendo a arquitetura (informalmente), informações não-estruturadas em mensagens de *commit*, lista de requisitos não-funcionais e documentos de casos de uso.

Table 5.5: Documentação arquitetural disponível para consulta por desenvolvedores e pesquisadores.

Grupo	Documento	Quantidade	% Grupo
Pesquisadores	Não há	8	42,1%
	Diagramas de UML	3	15,8%
	Documento de Arquitetura	2	10,5%
	Dados de *commit*	1	5,2%
	Documentação informal	1	5,2%
	Documento de Casos de Uso	1	5,2%
	Documento de Requisitos	1	5,2%
	Documentos escritos em AADL	1	5,2%
	Feature Model	1	5,2%
	Notas técnicas	1	5,2%
	Rascunho de Desenho estrutural	1	5,2%
	Registros de solicitação de mudanças	1	5,2%
	Requisitos não-funcionais	1	5,2%
Desenvolvedores	Não há	17	30,9%
	Diagramas de UML	10	18,1%
	Documento de Requisitos	6	10,9%
	Documento de Arquitetura	5	9,1%
	Página *Wiki*	5	9,1%
	Modelo de Dados	4	7,2%
	Código	3	5,5%
	Diagrama de Fluxo de Dados	3	5,5%
	Comentários de Código	2	3,6%
	Diagrama de Componentes	2	3,6%
	Documento de Casos de Uso	2	3,6%
	Documento de README	2	3,6%
	ApplicationMap	1	1,8%
	Desenho Arquitetural	1	1,8%
	Diagrama Arquitetural	1	1,8%
	Diagramas de Interação	1	1,8%
	Documentação dos Projetos	1	1,8%
	Documentação Geral	1	1,8%
	Documento de *Powerpoint*	1	1,8%
	Documento de Primeira Visão	1	1,8%
	Documento Escrito em ADL	1	1,8%
	Emails	1	1,8%
	Estrutura de pastas do código	1	1,8%
	Fluxogramas	1	1,8%
	Issue Trackers	1	1,8%
	Mensagens de *commit*	1	1,8%
	Páginas internas	1	1,8%
	Pipelines	1	1,8%
	Quadro Kanban	1	1,8%
	Registros no JIRA	1	1,8%
	Representações Gráficas de Topologia	1	1,8%

Quanto aos desenvolvedores que responderam ao questionário, 17 participantes reportaram não possuírem nenhuma documentação disponível (31% dos desenvolvedores) . Além desses, houveram citações de documentos disponíveis que remetem à arquitetura de software: código-fonte, diagramas da UML, documento arquitetural, documento de primeira visão, documento de requisitos, documentos de README, e-mails, fluxogramas, *issue trackers*, mensagens de *commit*, quadro *Kanban*, comentários de código, modelo de dados, estruturas de pastas, histórico do Git, páginas de wiki e apresentações em *Powerpoint*.

Após a análise dessas respostas, percebe-se que os desenvolvedores necessitam das informações arquiteturais, uma vez que apenas uma pequena parcela dos projetos possui

documentação arquitetural (utilizando AADL). Eles buscam essas informações em todo tipo de fonte de dados não-estutrurada em que acreditam haver informações sobre a arquitetura, tais como mensagens de *commit*, quadros *Kanban*, documentos de requisitos e documentos de primeira visão, dentre outros. A Tabela 5.5 apresenta a lista sumarizada de documentos onde pesquisadores e desenvolvedores relatam buscar informações sobre a arquitetura do projeto. A próxima subseção apresenta as estratégias adotadas, por desenvolvedores e pesquisadores, para preencher as lacunas trazidas pela escassez de documentação arquitetural.

5.4.4 RQ3.1: Quais as estratégias utilizadas por desenvolvedores e pesquisadores para compreender a arquitetura de um projeto de software durante sua prática profissional?

Conforme detalhado na seção de design deste estudo, a terceira seção do questionário relacionado se dedica à coletar junto aos participantes as estratégias utilizadas para a compreensão da arquitetura de um projeto de software durante a sua prática profissional.

Ao serem convidados a relatar as estratégias adotadas quando precisaram entender o projeto sem ter à disposição documentação satisfatória sobre o mesmo, os grupos de participantes apresentaram cenários diferentes. Quinze pesquisadores (de um total de 19) apontaram a análise de código como sendo a estratégia utilizada para compreender o projeto. Também foram citadas como estratégias a engenharia reversa e pedir ajuda aos desenvolvedores mais experientes da equipe. O Pesquisador 01 relatou em sua resposta o porquê da adoção do código-fonte como origem das informações arquiteturais recuperadas: "Em muitos casos de manutenção em sistema legado, a análise do código é o único instrumento para entend Em todos os casos, a experiên- cia foi desafiadora e custosa" (grifo nosso).

No grupo de participantes desenvolvedores, as estratégias utilizadas foram diversificadas. Embora a análise de código tenha sido a estratégia adotada pela maior parte dos desenvolvedores (35 de um total de 55 desenvolvedores), outras estratégias também tiveram adesão. Nessa questão, dois desenvolvedores afirmaram não acreditar na utilidade de se produzir documentação. As demais estratégias registradas foram: ajuda de desenvolvedor experiente, utilizar a javadoc[4], fazer engenharia reversa, estudar a documentação disponível, programar em par, analisar logs, além de analisar casos e códigos de teste. O Desenvolvedor 09 descreveu assim sua experiência no ingresso em projetos em evolução e sem documentação adequada: "Na maioria das vezes fui inserido em projetos em anda- mento e realmente utilizei o código-fonte para compreender sobre o domínio, regras de negócio e arquitetura do software. Alguns dos projetos apresentavam uma padronização de código-fonte e arquitetura coerente, no entanto, outros projetos apresentaram dificuldade de compreensão devido a falta de uma padronização. Essa falta de padronização dificultou a compreensão, e portanto, demandou mais tempo para manutenção/evolução do projeto."

Os desenvolvedores também apontaram algumas consequências enfrentadas diante desse cenário, como baixa produtividade, retrabalho, documentação tardia, aumento

[4]Conjunto de anotações disponíveis na linguagem Java que permitem comentar o código-fonte e gerar documentação em formato HTML a partir desses comentários.

do esforço e aparecimento de *code smells*. O Desenvolvedor 26 relata de forma direta problemas enfrentados em consequência da ausência de documentação adequada: "(...) Em todas as empresas que precisei atuar na evolução de um software tive uma barreira inicial, o que ocasionou na maior parte do tempo uma baixa produtividade, além do alto índice de *bugs*". Como se percebe, na ausência da documentação adequada, a principal origem de informações durante o processo de compreensão dos projetos é o código-fonte. Talvez isso explique por que a maioria das estratégias desenvolvidas, até aqui, para a recuperação de informações arquiteturais, é baseada nesse artefato.

Ao serem questionados sobre o uso de alguma ferramenta nesse processo de compreensão do projeto e quais tópicos arquiteturais são abordados, obtiveram-se cenários parecidos entre os dois grupos. Dentre os pesquisadores, 11 participantes (58%) relataram o uso de alguma ferramenta para a exploração do projeto de software; elas vão desde ferramentas de gerenciamento de configuração a ferramentas de análises de interesses. As ferramentas indicadas foram "ferramenta de visualização", AKS, Doxygen, DPduSS, "IDE própria com *plugins*", "ferramenta de modelagem arquitetural", Jude, GitHub, R, Understand, Sonar, Javadoc, KcacheGrind e Eclipse Debugger.

Para o grupo de desenvolvedores, 32 participantes (58%) indicaram o uso de alguma ferramenta como suporte à compreensão de projetos. Assim como no grupo de pesquisadores participantes, um amplo leque de ferramentas foi indicado pelos desenvolvedores. Destaque para o uso de IDEs (não especificadas) com o uso de *plugins* para o mapeamento arquitetural. Além das IDEs, foram citadas as seguintes ferramentas: Enterprise Architecture, leitores de PDF (para a documentação disponível), navegadores de internet (para a leitura de wikis), fluxogramadores (como o Draw.io e Papyrus), Redmine (gerenciador de projeto), Astah (modelador UML), Trello, GitHub, Sonar, ambientes de testes e de validações próprios e o editor de textos "vim". A Tabela 5.6 apresenta a lista de ferramentas citadas por pesquisadores e desenvolvedores como resposta a essa questão.

Apenas os tópicos arquiteturais "Mecanismos de Sincronização" e "Estilos Arquiteturais" não foram citados como identificáveis pelas ferramentas utilizadas pelos pesquisadores. A maioria dos respondentes desse grupo indicou "Estruturas do Sistema" e "Relação entre Elementos" como tópicos arquiteturais indicados por essas ferramentas. Para os desenvolvedores, todos os tópicos arquiteturais foram citados como identificáveis. A maioria dos desenvolvedores indicaram além dos tópicos arquiteturais "Estruturas de Sistema" e "Relação entre Elementos", os tópicos "Elementos Arquiteturais de Processamento", "Elementos Arquiteturais de Dados", "Motivações, Objetivos e Restrições" e "Mecanismos de Acesso à Dados" como sendo os principais, dentre os identificáveis pelas ferramentas que utilizam. A Tabela 5.7 apresenta quais tópicos arquiteturais são identificados com o uso das ferramentas citadas na Tabela 5.6 juntamente com o quantitativo de respondentes que identificam esses tópicos.

Table 5.6: Ferramentas utilizadas por desenvolvedores e pesquisadores no processo de compreensão dos projetos.

Grupo	**Ferramenta**	**Quantidade de indicações**
Pesquisadores	Nenhuma	8
	Doxygen	2
	AKS - Análise de interesses	1
	DPduSS	1
	Eclipse Debugger	1
	Ferramentas de Detecção de clones	1
	Ferramentas de Modelagem Arquitetural	1
	Ferramentas de UML	1
	Ferramentas de Visualização	1
	Ferramentas de Análise de Comentários	1
	Github	1
	IDE	1
	Javadoc	1
	KcacheGrind	1
	R	1
	Understand	1
Desenvolvedores	Nenhuma	23
	IDE	11
	Draw.io	4
	Wiki (navegador de internet)	4
	Ambiente de Teste	3
	Github	3
	Redmine	2
	Trello	2
	Ambiente de Validação próprio	1
	Astah	1
	Enterprise Architecture	1
	Ferramentas de Edição de texto	1
	Leitor de PDF	1
	Papyrus	1
	Reengenharia	1
	Scrum	1
	Sonar	1
	Vim (editor de texto)	1
	Webconferencia	1
	Whiteboards	1

A rigor, poucos participantes indicaram ferramentas diretamente relacionadas à arquitetura de software, a exemplo da "Understand" ou do "Enterprise Architecture". A maioria deles lançam mão de ferramentas de propósito geral, como gerenciadores de configuração ou fluxogramadores para construir algo como diagramas de componentes (em UML). Em outras situações, utilizam navegadores de internet ou leitores de PDF e de arquivos de texto para explorar a documentação e o código disponível e então constroem seu entendimento sobre os projetos. É importante destacar que não foram apontadas ferramentas para a extração de informações relevantes sobre a arquitetura a partir das fontes de dados não-estruturadas apontadas como fonte de informação arquitetural, tais como os Wikis, histórico de versões, emails, *issue trackers*, dentre outros. Outra importante constatação é que no grupo de desenvolvedores houve participantes que usam a mesma ferramenta mas não identificam os mesmos tópicos arquiteturais.

Em síntese, o que se extrai das respostas oferecidas pelos participantes é que o código-fonte tem sido a principal fonte de informações no processo de compreensão da arquitetura

Table 5.7: Tópicos arquiteturais identificados pelas ferramentas utilizadas por pesquisadores e desenvolvedores.

Tópico Arquitetural	**Pesquisadores**	**Desenvolvedores**
Elementos Arquiteturais de Processamento	3	28
Elementos Arquiteturais de Dados	3	30
Elementos Arquiteturais de Conexão	6	25
Motivações, Objetivos e Restrições	4	28
Estruturas Globais de Controle	3	27
Estruturas do Sistema	12	34
Protocolos de Comunicação	2	22
Mecanismos de Sincronização	0	21
Mecanismos de Acesso a Dados	1	33
Atribuições de Recursos	3	20
Relação entre Elementos	11	27
Organização Fundamental	4	22
Requisitos Não-Funcionais	4	24
Estilos Arquiteturais	0	21
Outros elementos relacionados à arquitetura	3	21

de projetos de software. As estratégias adotadas, o tipo de ferramenta utilizado e os tópicos arquiteturais obtidos reforçam essa afirmação. Assim, a resposta para **RQ3.1** é:

> A análise do projeto se dá a partir do código-fonte sob diversos ângulos: seja por análise de código, seja por análise de códigos de teste, seja por questionamentos à participantes experientes do projeto quanto ao código, seja pelo uso de ferramentas de engenharia reversa a partir do código-fonte ou ainda pelo desenvolvimento em par com outros membros do projeto.

Na próxima subseção serão apresentadas as percepções desses participantes sobre como mudanças na arquitetura são refletidas no código sob o ponto de vista de métricas.

5.4.5 RQ3.2: Qual a percepção dos desenvolvedores e pesquisadores sobre o impacto no código após mudanças na arquitetura de um projeto?

Nas perguntas relacionadas à **RQ3.2** foram apresentados os resultados sobre as variações de métricas provocadas por mudanças arquiteturais obtidos por esta pesquisa de doutorado. Frente a essas variações apresentadas, os participantes foram convidados a refletir sobre as suas expectativas acerca dessas mudanças, assim como sobre sua experiência pregressa, seja nos projetos em que participaram, seja em outros projetos sobre os quais têm conhecimento. Os resultados obtidos por esta pesquisa no tocante à variação dos valores de métricas em mudanças que envolvem a arquitetura do projeto se resumem à tabela apresentada na Figura 5.4. Ela consta no questionário produzido e foi referenciada em todas as perguntas da seção 4 deste questionário.

Quando confrontados quanto ao resultado apresentado e a expectativa dos participantes, houve uma distinção entre os dois grupos participantes. Considere-se o quantitativo de 19 pesquisadores e 55 desenvolvedores. Para o grupo de pesquisadores, a maioria nutria expectativa semelhante aos resultados apresentados (11 participantes) ou concorda com

parte desses resultados (4 participantes). Os pesquisadores que não concordaram com o resultado (4 participantes) relataram que não é possível generalizar uma tendência, uma vez que cada tópico arquitetural (ou interesse) provoca um tipo de variação diferente nas métricas. O Pesquisador 02 ilustra essa constatação em sua resposta: "É importante eu ressaltar algo em relação à minha resposta: como venho pesquisando sobre interesses em software, eu sempre vinculo o que venho medindo a cada interesse. Desta forma, o que obtenho de valores de métricas é sensível a cada interesse que observo. Estratégias dos desenvolvedores para injetar o interesse 'Logging' é bem diferente de outras que eles aplicam para implementar 'Test' ou 'Mathematical Processing'".

MÉTRICAS DE TAMANHO	
Nome da Métrica	**Tendência Verificada**
Tamanho Médio dos Métodos **(AMLOC)**	**AUMENTO**
Número Médio de Parâmetros **(ANPM)**	**REDUÇÃO**
Linhas de Código **(LOC)**	**AUMENTO**
Número de Atributos **(NOA)**	**MANUTENÇÃO**
Número de Métodos **(NOM)**	**AUMENTO**
Tamanho do Maior Método **(MMLOC)**	**AUMENTO**
MÉTRICAS DE COMPLEXIDADE	
Complexidade Ciclomática Média **(ACCM)**	**REDUÇÃO**
Número de Atributos Públicos **(NPA)**	**MANUTENÇÃO**
Número de Métodos Públicos **(NPM)**	**MANUTENÇÃO**
MÉTRICAS DE ACOPLAMENTO	
Conexões Aferentes por Classe **(ACC)**	**AUMENTO**
Acoplamento entre Objetos **(CBO)**	**MANUTENÇÃO**
Profundidade da Árvore de Herança **(DiT)**	**MANUTENÇÃO**
Número de Classes Filhas **(NOC)**	**MANUTENÇÃO**
Respostas por Classe **(RFC)**	**AUMENTO**
MÉTRICAS DE COESÃO	
Perda de Coesão por Método **(LCOM4)**	**MANUTENÇÃO**
Complexidade Estrutural **(SC)**	**AUMENTO**

Figure 5.4: Resumo das variações obtidas para cada métrica estudada neste estudo de doutorado em *commits* com alterações na arquitetura de software.

Para o grupo dos desenvolvedores, por outro lado, obteve-se a correspondência de expectativa para 32 participantes. Nesse grupo houve os desenvolvedores que se disseram indiferentes às métricas apresentadas (e por isso discordavam dos valores), os que acreditavam que o acoplamento diminuiria e a coesão aumentaria, e um desenvolvedor que apontou o comportamento das métricas como sendo não-generalizável. O Desenvolvedor 30 ilustra em sua resposta a indiferença reportada por alguns: "Honestamente a informação apresentada acima afeta em absolutamente nada o meu processo de desenvolvimento". Já o Desenvolvedor 45 declarou em sua resposta: "Eu acho que depende da mudança. Para pequenas

mudanças, talvez" (as variações de métricas correspondam às expectativas dele).

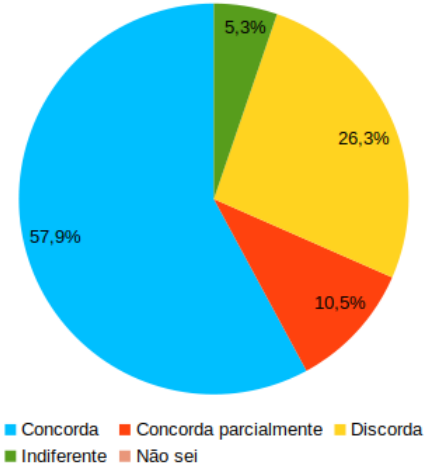
Como se vê, a maioria dos participantes (58% dos participantes de ambos os grupos) concordam com os resultados apresentados. No entanto, é preciso considerar a aplicação desse estudo em outros projetos de software com perfil parecido, uma vez que um dos participantes apontou a possibilidade de não-generalização dos resultados. Houveram também desenvolvedores que relacionaram mudanças arquiteturais às atividades de refatoração, uma vez que esperavam melhoria nos atributos de qualidade interno (manutenibilidade foi um aspecto apontado por mais de um participante). A Figura 5.5 apresenta o grau de concordância de pesquisadores e desenvolvedores frente ao resultado obtido em comparação com sua própria expectativa em relação a essas mudanças. Para efeito de lembrança, considere-se o quantitativo de participantes: 19 pesquisadores e 55 desenvolvedores.

Quando perguntados sobre a generalização desses resultados para todos os projetos, a tendência apresentada nas respostas da questão anterior se reforçou. A maior parte dos participantes (52% dos pesquisadores e 49% dos desenvolvedores) relataram que sua experiência aponta para resultados similares nos demais projetos. Da mesma forma, entre os dois grupos de participantes, houve pesquisadores e desenvolvedores que discordaram da generalização desses resultados. Em ambos os grupos, as justificativas apresentadas estão relacionadas ao tópico abordado na modificação, às particularidades de cada projeto e ao estilo arquitetural utilizado. O Pesquisador 02 descreveu dessa forma o motivo de sua discordância: "Essa é outra dimensão da minha pesquisa. O que venho percebendo é que os resultados também são sensíveis à forma como você escolhe agrupar projetos de software para análise. Quando você agrupa projetos considerando que eles se destinam a automatizar a mesma coisa (softwares de um mesmo domínio), os resultados são mais uniformes em relação à forma como evoluem.". No grupo de desenvolvedores, o Desenvolvedor 20 foi na mesma linha, porém de forma sucinta: "Não concordo, pois creio que dependa do tipo de modificação na arquitetura.".

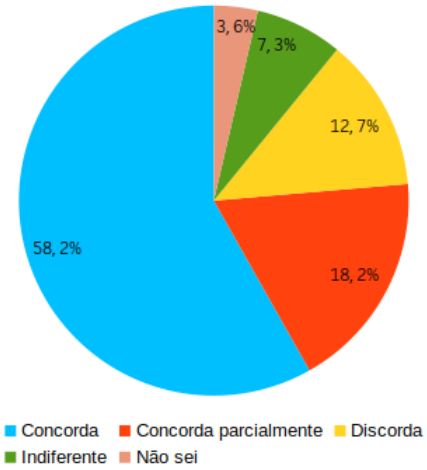
Assim como na questão anterior, embora haja a concordância da maioria dos participantes deste estudo, há espaço para um trabalho futuro onde se coloque à prova os resultados obtidos durante esta pesquisa de doutorado. Nesse trabalho futuro, considerar o tipo da mudança arquitetural e o domínio do projeto utilizado na pesquisa, é imprescindível, pois tratam-se de variáveis que podem trazer respostas mais detalhadas a essa questão. A Figura 5.6 apresenta esse comparativo em um gráfico de pizza para cada grupo participante desta pesquisa.

Quando questionados sobre a similaridade dos resultados obtidos com os projetos em que os participantes desta pesquisa desenvolvem ou pesquisam, o número de desenvolvedores e pesquisadores que atesta a repetição dos resultados apresentados diminui para os pesquisadores (42% dos participantes) e aumenta para os desenvolvedores (54% dos participantes) quando comparado à generalização dos resultados para qualquer projeto. Embora, em ambos os grupos, um número significativo de participantes reconheça que os resultados apresentados nesta pesquisa se repete em seus projetos, o número de discordâncias (principalmente de concordâncias parciais) cresce em relação às perguntas anteriores. As justificativas para isso permanecem as citadas nas perguntas anteriores: tópicos diferentes provocam mudanças diferentes, cada projeto tem suas particularidades, o estilo arquitetural interfere no impacto das mudanças sobre o código, além das questões

relacionadas à melhoria dos atributos de qualidade internos. O Pesquisador 12 anotou, curiosamente, uma divergência entre seus projetos utilizados atualmente como objeto de estudo e outros utilizados no passado: "Não nos projetos que estou investigando, mas nos quais participei anteriormente sim!". O Desenvolvedor 10, por sua vez, reconheceu alguns dos resultados apresentados e refutou outros: "Alguns projetos em que observei refatorações em partes pontuais no código percebi uma diminuição do número de métodos e aumento no tamanho dos métodos que foram criados. A conexão entre as classes também aumentou. Existiu uma maior dependência entre elas".



(a) Grau de concordância de pesquisadores com as variações de métricas verificadas em mudanças que envolvam a arquitetura do projeto.

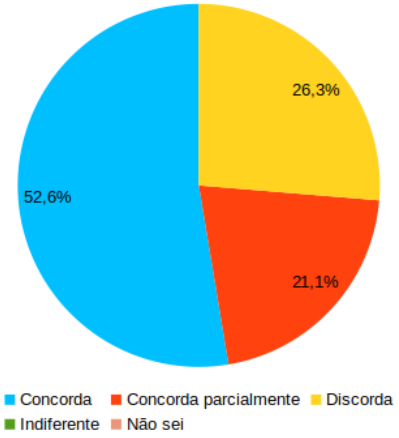


(b) Grau de concordância de desenvolvedores com as variações de métricas verificadas em mudanças que envolvam a arquitetura do projeto.

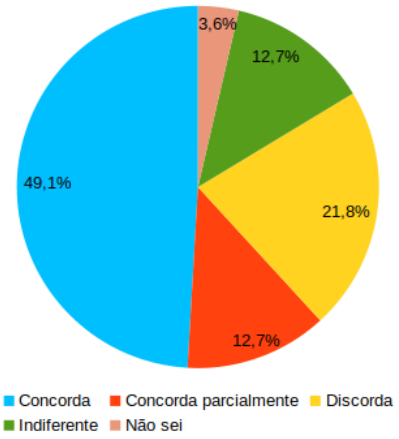
Figure 5.5: Comparação do grau de concordância dos participantes deste estudo com as variações de métricas obtidas em mudanças onde a arquitetura do projeto foi modificada.

A Figura 5.7 apresenta o grau de alinhamento entre os resultados obtidos por este estudo de doutorado no tocante à variação de valores de métricas em mudanças que

envolvem modificações na arquitetura do projeto e os projetos que eles colaboram (desenvolvedores) ou pesquisam (pesquisadores).



(a) Grau de concordância de pesquisadores com as variações de métricas verificadas em mudanças que envolvam a arquitetura do projeto em comparação com a maioria dos projetos.

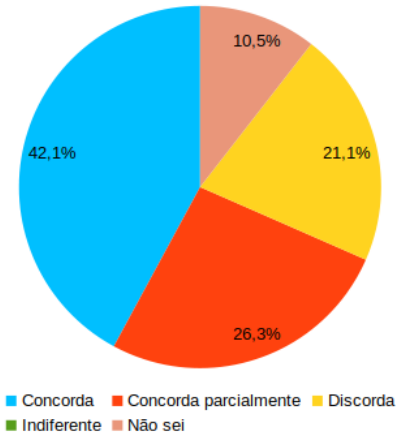


(b) Grau de concordância de desenvolvedores com as variações de métricas verificadas em mudanças que envolvam a arquitetura do projeto em comparação com a maioria dos projetos.

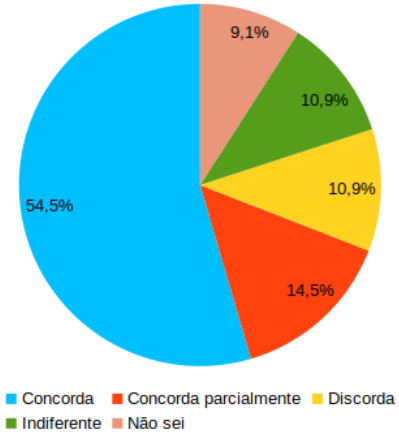
Figure 5.6: Comparação do grau de concordância dos participantes deste estudo com as variações de métricas obtidas em mudanças onde a arquitetura do projeto foi modificada em comparação com a maioria dos projetos.

A Figura 5.8 apresenta a comparação entre as respostas obtidas para cada questão relacionada a esta **RQ3.2**. Observe-se que há similaridade nos índices de aceitação dos resultados apresentados por esta tese quando a comparação está relacionada às próprias expectativas e aos projetos em geral no tocante à variação das métricas analisadas quando a alteração no projeto atinge a arquitetura do mesmo. No entanto, embora o grau de aceitação desses resultados como sendo generalizáveis para os projetos em geral e para os projetos em que atuam fique por volta de 50% dos participantes (até 8% a mais ou a menos), o grau de rejeição para aceitação/generalização dos resultados apresentados

se mantém estável para o grupo de pesquisadores, mas aumenta para os desenvolvedores quando comparados aos projetos em geral. Outro fenômeno destacável é o aumento da incidência de respostas inconclusivas ("não sei") tanto para desenvolvedores quanto para pesquisadores quando a comparação é estabelecida com os projetos com os quais trabalha (pesquisando ou desenvolvendo).



(a) Grau de concordância de pesquisadores com as variações de métricas verificadas em mudanças que envolvam a arquitetura do projeto em comparação com projetos que utiliza nas pesquisas.



(b) Grau de concordância de desenvolvedores com as variações de métricas verificadas em mudanças que envolvam a arquitetura do projeto em comparação com projetos que desenvolveu/desenvolve.

Figure 5.7: Comparação do grau de concordância dos participantes deste estudo com as variações de métricas obtidas em mudanças onde a arquitetura do projeto foi modificada em comparação com os projetos que usa em suas pesquisas (pesquisadores) ou é colaborador (desenvolvedores).

Dessa forma, a resposta para **RQ3.2** é:

A percepção dos desenvolvedores e pesquisadores sobre o impacto no código após as mudanças na arquitetura do projeto é que se repetem os resultados obtidos nesta tese. No entanto, cabe a ressalva para o comportamento de outros projetos de Software Livre, onde se possa estudar os impactos causados por cada tópico arquitetural individualmente e comparar os resultados obtidos com projetos concebidos a partir de diferentes estilos arquiteturais.

Na próxima subseção são apresentadas as percepções sobre a importância de mensagens de *commit* contendo traços arquiteturais na compreensão de mudanças arquiteturais.

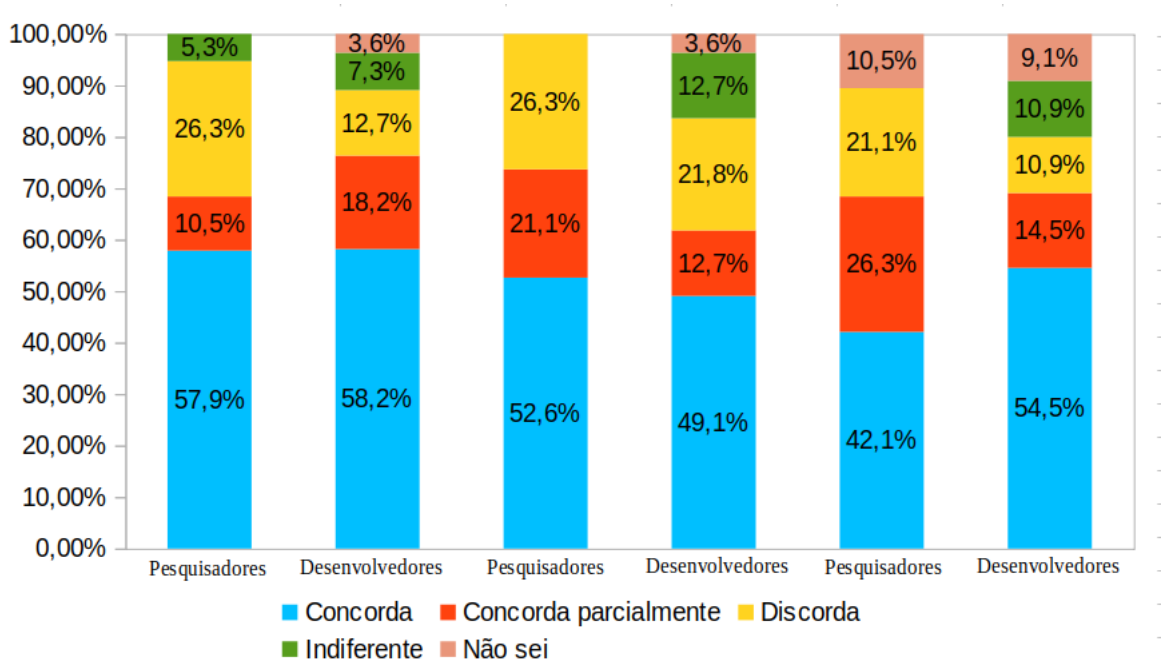


Figure 5.8: Avaliação das métricas obtidas neste estudo doutorado em *commits* com alterações na arquitetura de software: **Expectativa dos participantes** x **Repetição dessas variações em todos os projetos** x **Repetição dessas variações nos projetos que desenvolve/pesquisa**.

5.4.6 RQ3.3: Qual a percepção dos desenvolvedores e pesquisadores frente aos traços arquiteturais presentes em mensagens de commit como artefato que auxilie na compreensão de mudanças na arquitetura?

Este trabalho de doutorado apresentou o conceito de traço arquitetural. Ele relacionou a existência dessa informação ao registro de mudanças na arquitetura de um projeto de software em mensagens de *commit*. A **RQ3.3** questiona se essas mensagens contendo traços arquiteturais auxiliam na compreensão de mudanças na arquitetura de software. Assim, os participantes foram apresentados a cinco mensagens de *commit* que contêm traços arquiteturais (Figura 5.9). A partir dessas mensagens, eles foram questionados quanto à utilidade dessas mensagens para compreender (i) as mudanças realizadas na

arquitetura e a consequente evolução arquitetural do projeto. Também foi questionada (ii) a existência desse tipo de mensagem em seu projeto e (iii) os tópicos arquiteturais que os participantes identificam nessas mensagens.

Mensagem 1: "O kio agora usa o nepomuk em algum lugar, portanto, é necessário incluir o soprano por extensão"

Mensagem 2: "garantir que o slotchanged seja chamado em todas as instâncias, mova a chamada slotchanged para o slot broadcastdone para que seja chamado em todas as instâncias do dispositivo; de fato, broadcastdone é chamado pelo dbus. Isso garante que o estado do dispositivo seja atualizado corretamente quando modificado por alguma rotina fora de processo, por exemplo quando montada por meio de ações kde-runtime/soliduiserver"

Mensagem 3: "(...) código usando o projeto iso-codes como fonte de validação e tradução está sendo desenvolvido no kdepimlibs e no kdebase e será movido para o kdelibs no 4.7 uma vez que os requisitos e a dependência da API implicações são melhor compreendidas"

Mensagem 4: "use o algoritmo fisher-yates para randomizar listas, isso garante que o tempo necessário para o embaralhamento dependa linearmente do comprimento da lista, e não quadraticamente. Além disso, evita alocações de memória durante o embaralhamento e economiza tempo, mesmo em listas muito curtas"

Mensagem 5: "além dos protocolos locais, não procure informações de proxy se um URL não tiver componente de host, por exemplo o protocolo de dados"

(a) Mensagens apresentadas na Seção 5 do formulário utilizado nesta pesquisa (traduzidas e disponíveis na versão em Português).

Message 1: "kio now uses nepomuk somewhere, so needs soprano includes by extension"

Message 2: "Ensure slotchanged is called in all instances move slotchanged call to the broadcastdone slot so that it is called in all instances of the device; in fact broadcastdone is called by dbus. this makes sure that the device state is correctly updated when modified by some out-of-process routine, e.g. when mounted via kde-runtime/soliduiserver actions."

Message 3: "(...) code using the iso-codes project as a source for validation and translation is being developed in kdepimlibs and kdebase and will be moved into kdelibs in 4.7 once the api requirements and dependency implications are better understood."

Message 4: "Use the fisher-yates algorithm to randomize lists this ensures that the time required for the shuffling depends linearly on the length of the list, rather than quadratically. Moreover, it prevents memory allocations during the shuffling and thus saves time even for very short lists."

Message 5: "In addition to local protocols, don't look up proxy information if a url has no host component, e.g. the data protocol."

(b) Mensagens apresentadas na Seção 5 do formulário utilizado nesta pesquisa (na versão em Inglês).

Figure 5.9: Mensagens, contendo traços arquiteturais, apresentadas a pesquisadores e desenvolvedores na Seção 5 do formulário utilizado nesta pesquisa (versões em Português e em Inglês).

A primeira pergunta desta seção foi direcionada para o reconhecimento dos tópicos arquiteturais presentes nas mensagens expressas na Figura 5.9. Foram oferecidos aos participantes os quinze tópicos apresentados na Seção 3.3.4.3 e solicitado que fossem apontados quais daqueles elementos arquiteturais (tópicos) estavam contidos nestas mensagens. Os resultados foram interpretados de duas formas: Primeiro observou-se entre os grupos de participantes (desenvolvedores e pesquisadores) quais os tópicos reconhecidos pela maioria. A maioria dos pesquisadores apontou os tópicos "**Elementos Arquiteturais de Processamento**", "**Elementos Arquiteturais de Conexão**" e "**Estruturas do Sistema**". Também tiveram uma quantidade expressiva de indicações os tópicos "Protocolos de Comunicação" e "Requisitos Não-Funcionais". No grupo de desenvolvedores, o resultado obtido foi parecido. Os tópicos reconhecidos pela maioria desse grupo foram "**Elementos Arquiteturais de Processamento**", "**Elementos Arquiteturais de Dados**", "**Elementos Arquiteturais de Conexão**" e "**Protocolos de Comunicação**". Assim como no grupo de pesquisadores, houve outros tópicos que se destacaram em número de citações por parte dos desenvolvedores que participaram desta pesquisa: "Estruturas do Sistema", "Relação entre Elementos" e "Requisitos Não-Funcionais". A Tabela 5.8 apresenta a quantidade de citações registradas em relação à cada tópico disponível.

Table 5.8: Tópicos arquiteturais identificados por pesquisadores e desenvolvedores nas mensagens presentes na Figura 5.9.

Tópico Arquitetural	**Pesquisadores**	**Desenvolvedores**
Elementos Arquiteturais de Processamento	12	33
Elementos Arquiteturais de Dados	8	28
Elementos Arquiteturais de Conexão	12	35
Motivações, Objetivos e Restrições	3	18
Estruturas Globais de Controle	5	16
Estruturas do Sistema	11	25
Protocolos de Comunicação	8	28
Mecanismos de Sincronização	2	17
Mecanismos de Acesso a Dados	5	19
Atribuições de Recursos	7	18
Relação entre Elementos	5	20
Organização Fundamental	1	8
Requisitos Não-Funcionais	7	24
Estilos Arquiteturais	2	7
Outros elementos relacionados à arquitetura	2	3

A segunda interpretação dos dados obtidos levou em conta a experiência dos participantes. Assim como na caracterização dos participantes, seccionou-se pesquisadores e desenvolvedores em 3 grupos: (i) com experiência registrada menor que cinco anos (formado por 14 desenvolvedores e 1 pesquisador); (ii) com experiência registrada entre cinco e dez anos (formado por 27 desenvolvedores e 9 pesquisadores), e; (iii) com experiência registrada maior que dez anos (formado por 14 desenvolvedores e 9 pesquisadores). Como pode-se observar na Tabela 5.9, a experiência de desenvolvedores e pesquisadores interfere diretamente na interpretação das mensagens indicadas para avaliação. Note-se que os tópicos "Mecanismos de Acesso a Dados" e "Atribuições de Recursos" não são maioria dentre os desenvolvedores em geral, mas se destacam no grupo de desenvolvedores menos

experientes. Por outro lado, o tópico "Requisitos não-funcionais", identificado pelos desenvolvedores mais experientes, não figura entre aqueles apontados pela maioria dos desenvolvedores como presente nas mensagens avaliadas (apesar de obter destaque não representou maioria no agrupamento geral de desenvolvedores). Destaque-se o tópico "Mecanismos de Sincronização" presente na mensagem 2 que não foi identificado pela maioria de desenvolvedores e pesquisadores, nem de uma forma geral nem por nenhum grupo formado por tempo de experiência.

Table 5.9: Tópicos arquiteturais identificados por pesquisadores e desenvolvedores nas mensagens presentes na Figura 5.9 organizados por tempo de experiência.

Tópico Arquitetural	**Experiência**	**Pesquisadores**	**Desenvolvedores**
Elementos Arquiteturais de Processamento	< 5 anos	1	6
	Entre 5 e 10 anos	7	14
	> 10 anos	5	13
Elementos Arquiteturais de Dados	< 5 anos	1	9
	Entre 5 e 10 anos	3	13
	> 10 anos	5	7
Elementos Arquiteturais de Conexão	< 5 anos	1	10
	Entre 5 e 10 anos	6	16
	> 10 anos	6	9
Motivações, Objetivos e Restrições	< 5 anos	0	5
	Entre 5 e 10 anos	2	8
	> 10 anos	1	4
Estruturas Globais de Controle	< 5 anos	0	6
	Entre 5 e 10 anos	2	6
	> 10 anos	4	5
Estruturas do Sistema	< 5 anos	1	5
	Entre 5 e 10 anos	6	14
	> 10 anos	5	7
Protocolos de Comunicação	< 5 anos	1	8
	Entre 5 e 10 anos	5	16
	> 10 anos	3	5
Mecanismos de Sincronização	< 5 anos	0	5
	Entre 5 e 10 anos	2	9
	> 10 anos	1	4
Mecanismos de Acesso a Dados	< 5 anos	1	7
	Entre 5 e 10 anos	4	7
	> 10 anos	1	4
Atribuições de Recursos	< 5 anos	0	7
	Entre 5 e 10 anos	4	6
	> 10 anos	3	4
Relação entre Elementos	< 5 anos	1	5
	Entre 5 e 10 anos	3	8
	> 10 anos	1	6
Organização Fundamental	< 5 anos	0	1
	Entre 5 e 10 anos	0	5
	> 10 anos	1	2
Requisitos Não-Funcionais	< 5 anos	0	6
	Entre 5 e 10 anos	3	9
	> 10 anos	4	8
Estilos Arquiteturais	< 5 anos	0	1
	Entre 5 e 10 anos	0	3
	> 10 anos	2	3
Outros elementos relacionados à arquitetura	< 5 anos	0	0
	Entre 5 e 10 anos	0	1
	> 10 anos	2	2

O autor desta tese identificou os seguintes tópicos nestas mensagens: "**Elementos Arquiteturais de Processamento**", "**Elementos Arquiteturais de Dados**", "**Elementos Arquiteturais de Conexão**", "**Estruturas do Sistema**", "**Protocolos de Comunicação**", "**Mecanismos de Sincronização**", "**Relação entre Elementos**" e "**Requisitos Não-Funcionais**". Embora apenas os cinco últimos tópicos estejam sendo alvo de modificação, os três primeiros (tópicos estruturais) também podem ser identificados nessas mensagens.

Sobre a utilidade de mensagens de *commit* contendo traços arquiteturais como auxílio na compreensão de mudanças na arquitetura do projeto de software, no grupo de pesquisadores que participaram da pesquisa, a maioria (11 participantes, 58% dos pesquisadores) concorda com a utilidade dos traços arquiteturais, destacando, porém, o seu caráter complementar (pois não se tratam de informações completas e direcionadas). Curiosamente, parte dos pesquisadores que não consideram úteis os traços arquiteturais citaram que os traços arquiteturais podem colaborar na compreensão do projeto, desde que haja outras informações. O Pesquisador 19 fala sobre a utilidade dos traços arquiteturais, mas levanta questão sobre a arquitetura atual: "Com certeza, elas ajudam a compor o conhecimento sobre o projeto. Mas fica a pergunta: e o estado arquitetural atual?". Já o Pesquisador 01 enfatiza a natureza complementar dessas informações: "Elas são complementares à documentação formal e análise de código". Por fim, o Pesquisador 07 sublinhou a necessidade de contextualização do projeto, em conjunto com essas informações: "Isoladamente não são úteis. Mas com a apresentação de um contexto ajudaria sim".

No grupo de desenvolvedores esse cenário se repetiu. Trinta e cinco participantes (63% dos desenvolvedores) concordam que os traços arquiteturais colaboram na compreensão do projeto. A maioria deles, assim como no grupo dos pesquisadores, também apontaram a necessidade de mais informações para que se compreenda o cenário de mudança arquitetural por completo. Dentre os desenvolvedores que discordam da utilidade dos traços arquiteturais, também se registraram justificativas sobre a não-completude das informações disponíveis nos traços arquiteturais. Enquanto o Desenvolvedor 12 confirmou a utilidade dos traços arquiteturais — "Elas me parecem ser bem detalhadas sobre o que foi implementado em cada versão do sistema" —, o Desenvolvedor 30 afirmou: "Sim. Mas não isoladamente. Dentro de um contexto, com mais informações, elas se encaixam como um quebra-cabeça, ajudando a construir uma imagem geral da arquitetura construída.". Num outro sentido, o Desenvolvedor 15 afirmou "Não, minha empresa não foca em arquitetura do software".

Sobre a existência de mensagens contendo traços arquiteturais nos projetos em que os participantes desta pesquisa trabalham (desenvolvendo ou pesquisando), no grupo de pesquisadores a maioria refutou a afirmação (12 pesquisadores, 63%). Dentre os que confirmaram a presença dos traços arquiteturais (7 pesquisadores), apenas dois registraram exemplos dessas mensagens. O Pesquisador 05 transcreveu uma mensagem relativa à mudança em um "Mecanismo de Sincronização": "*ExpressionTransformer will go before all other RecordTransformers, so that every other transformer has the real values*". O Pesquisador 11 relatou em sua resposta a dificuldade de encontrar esse tipo de informação dentre as demais: "Não atuo como desenvolvedor em um projeto específico. Atuo como pesquisador, analisando muita informação em repositório de sw. Neste caso, o tempo gasto para selecionar um tópico tornaria a resposta muito custosa."

No outro sentido, dentre os desenvolvedores que participaram desta pesquisa, 29 relataram haver esse tipo de informação em seus projetos (52% dos desenvolvedores) e dezessete deles apresentaram mensagens contendo traços arquiteturais. Dentre os desenvolvedores que negaram a existência de traços arquiteturais em meio às mensagens de *commit*, apenas um relatou que faz apenas o registro das mudanças acompanhado de um link para o número do *ticket* do *issue tracker* utilizado no projeto. O Desenvolvedor 22 apontou uma mudança em requisito não-funcional como exemplo de mensagem com traço arquitetural: "*improving performance by using YAML::XS*". O Desenvolvedor 51 apontou uma mudança em um "Elemento Arquitetural de Conexão": "*move to repository pattern instead of ActiveRecord pattern*". O Desenvolvedor 40, por outro lado apontou que esse tipo de informação não fica apenas nas mensagens de *commit*: "Evitamos deixar informações tão relevantes apenas ao comentário do *commit*. O uso do git, como controlador de versão, entrega um conjunto de ferramentas que facilita a documentação dessas alterações. Temos utilizado cada vez mais a documentação no git através de markdown e controle de resolução de problemas através das issues."

Sobre o uso de mensagens com traços arquiteturais para ajudar na compreensão da evolução de projetos de software, os participantes dos dois grupos (pesquisadores e desenvolvedores) registraram percepções parecidas. Em ambos os grupos a larga maioria concorda que na ausência de informações sobre a evolução arquitetural, a disponibilidade de traços arquiteturais ajuda a explicar como a arquitetura foi modificada ao longo do projeto – 16 pesquisadores (84% do total) e 45 desenvolvedores (82% do total). No entanto, em ambos os grupos houveram os participantes que não concordam com essa utilidade. Entre os pesquisadores, dois indicaram que se tratam de informações complementares sobre as modificações no projeto e, portanto, sozinhas não substituem a documentação formal. O Pesquisador 03 informa: "Acredito que essas mensagens ajudam a compreender as mudanças, em conjunto com outras fontes". Já o Pesquisador 04 ressalta a dificuldade em encontrar tais informações: "Acredito que os filtros devem ser ajustados para detectá-las com o menor esforço possível.". Por outro lado, o Pesquisador 07 relembrou que: "Precisamos associá-las a outros tipos de informação.".

Dentre os desenvolvedores, oito apresentaram discordância quanto à utilidade de traços arquiteturais para o entendimento da evolução arquitetural dos projetos. Dentre as justificativas, está a dificuldade de identificação de tais informações, a necessidade de conexão dessas informações com o contexto do projeto. Um desenvolvedor apontou a lista de tarefas pendentes como possuindo informações mais reveladoras que traços arquiteturais. O Desenvolvedor 30 ilustra a preocupação com o uso dessas informações isoladamente: "Essas informações devem estar relacionadas ao contexto das decisões.". Já o Desenvolvedor 50 relembra a dificuldade em encontrar esse tipo de informação: "É dificil identificar esse tipo de mensagem em meio às demais.". Por outro lado o Desenvolvedor 07 celebra a existência desse tipo de mensagem: "Quando disponíveis, ajudam na manutenção e evolução.". Na mesma linha, o Desenvolvedor 24 aponta uma outra utilidade para esse tipo de informação: "Ajuda a rastrear a origem de bugs a partir de mudanças na arquitetura.". Ainda nesse sentido, o Desenvolvedor 17 simplifica sua percepção sobre esse tipo de mensagem: "Ajudam a compreender a evolução do projeto."

Assim, a percepção para **RQ3.3** é que:

Em geral tanto desenvolvedores quanto pesquisadores reconhecem nas mensagens de *commit* contendo traços arquiteturais uma fonte de informações que auxilia na compreensão de mudanças na arquitetura do projeto, principalmente em um contexto de ausência de documentação adequada que apresente a arquitetura do projeto e sua evolução. No entanto, nos dois grupos de participantes há céticos quanto à possibilidade de disponibilidade e recuperação de tais informações e quanto à sua completude, embora esta pesquisa deixe claro que os traços arquiteturais trazem consigo informações incompletas sobre a arquitetura.

A próxima seção apresenta as ameaças à validade deste estudo.

5.5 AMEAÇAS À VALIDADE

As ameaças à validade deste estudo foram identificadas, tiveram medidas para sua mitigação tomadas as quais são descritas a seguir.

Validade de Construto. Em se tratando de um questionário que trata de conceitos e crenças quanto a mudanças de arquitetura, identificou-se uma ameaça social relacionada ao comportamento dos participantes, onde o participante pode se sentir pressionado por achar que está sendo avaliado (WOHLIN et al., 2012). Outra ameaça detectada é a compreensão dos participantes quanto ao que foi perguntado no questionário.
Para diminuir o viés causado pela apreensão trazida pela ameaça social, e obter informações diretamente relacionadas à experiência de cada participante, o convite para a participação reforçou a necessidade de que apenas a experiência profissional seria suficiente para responder às perguntas e, assim, atingir os objetivos da pesquisa. Para garantir que os participantes compreendam exatamente o que está sendo perguntado em cada pergunta do questionário, foi realizado uma aplicação piloto com dois desenvolvedores e dois pesquisadores onde um dos aspectos observados foi a compreensão das perguntas. Como resultado desta aplicação piloto, cinco perguntas foram reformuladas. Ainda para garantir a compreensão das perguntas por parte dos participantes, foi oferecido, a cada seção do formulário, links para artigos relacionados ao tema Arquitetura de Software, caso os participantes não se sentissem familiarizados com os termos utilizados.

Validade de Conclusão. Segundo Wohlin et al. (2012), é preciso ter atenção com a codificação das respostas oferecidas pelos participantes, pois é necessário garantir que os dados e as análises sejam as mesmas independente dos pesquisadores que a realizarem. Para mitigar essa ameaça, a codificação foi revisada por completo por um pesquisador independente. Esse processo se deu com o autor desta tese fazendo a codificação de cada resposta. Paralelamente, um pesquisador independente procedeu a codificação destas respostas. Ao final, os dois pesquisadores se reuniram e fizeram o alinhamento dos códigos, onde para cada código atribuído de forma divergente sofreu nova avaliação e foi realizada uma atividade de consenso.

Validade Externa. As ameaças externas estão relacionadas à generalização dos dados obtidos. Para este estudo, foi identificada como ameaça externa o tamanho da amostra obtida, assim como a diversidade dos respondentes. Foram convidados participantes dos cinco continentes, no entanto obtiveram-se respostas de apenas onze diferentes países de três continentes. Além disso, a grande maioria dos participantes atua no Brasil.

Apesar de atuarem profissionalmente em diferentes organizações/centros e institutos de pesquisa e, de possuírem experiência acumulada diversa entre si, não se pode garantir que os resultados obtidos representam qualquer cenário onde a pesquisa seja replicada.

Validade Interna. A primeira ameaça interna que se deve considerar é a seleção de participantes, porque eles foram escolhidos, inicialmente, por meio de amostragem de conveniência. No entanto, um terço dos participantes foram convidados indiretamente, reduzindo essa ameaça.

Outra ameaça é que os participantes poderiam ser afetados negativamente pelo tédio e pelo cansaço. Para minimizar essa ameaça, ofereceu-se um formulário online, de preenchimento entre 20 e 30 minutos. Realizou-se, também, um estudo piloto onde dois pesquisadores e dois desenvolvedores responderam a uma versão preliminar do questionário e foram monitorados durante esse processo. A ordenação das seções e a versão final das perguntas sofreu alterações a partir das lições aprendidas com o estudo piloto.

A seguir, são apresentadas as conclusões deste estudo.

5.6 CONCLUSÕES

Esse capítulo mostrou que os desenvolvedores e pesquisadores, em sua maioria, acreditam na existência e na possibilidade de exploração de informações sobre mudanças na arquitetura registradas em mensagens de *commit*. Também foi relatado por esse público a dificuldade em ter acesso a esse tipo de informação nos artefatos que "deveriam" ser mantidos durante a evolução do projeto. A documentação da arquitetura de projetos de software é um dos documentos citados como indisponíveis ou desatualizados. A partir disso, é preciso que se considere a mineração de mensagens de *commit* como um método válido na busca por informações sobre a arquitetura de software.

Esse capítulo também mostrou que os participantes reconhecem os impactos produzidos no código-fonte por mudanças na arquitetura do projeto. Embora tenham sido registradas controvérsias em relação a eles, a maioria dos desenvolvedores reportaram a ocorrência de resultados parecidos em projetos onde participam seja como desenvolvedores seja como pesquisadores. Trata-se de mais uma informação que pode ajudar a desenvolvedores e pesquisadores que buscam retratos mais próximos do que aconteceu ao longo da evolução dos projetos no tocante à mudanças na arquitetura. Quanto às controvérsias, elas abrem espaço para pesquisas semelhantes com diferentes projetos onde se possa comparar os resultados obtidos e assim dirimi-las.

Chapter 6

CONSIDERAÇÕES FINAIS

Conforme já afirmado ao longo deste texto, os desenvolvedores tomam decisões ao longo do ciclo de vida dos projetos de software. Essas decisões estão relacionadas aos vários elementos do projeto, tais como design (e especificamente à arquitetura), implementação e requisitos. No entanto, essas decisões podem mudar durante a evolução do projeto. O conhecimento sobre essas mudanças é de grande importância durante a manutenção e evolução do software, pois fazer alterações sem o devido cuidado pode resultar em resultados indesejáveis (GURP; BOSCH, 2002; FARID; AZAM; IQBAL, 2011). Nesse sentido, várias pesquisas foram destinadas a buscar informações sobre mudanças na arquitetura (e em outras visões e artefatos) do projeto, usando diferentes técnicas e diferentes fontes de informação (OREIZY; MEDVIDOVIC; TAYLOR, 1998; MANCORIDIS et al., 1999; DUTOIT; PAECH, 2001; RIVA et al., 2004; PINZGER, 2005; YING; WRIGHT; ABRAMS, 2005; KNODEL et al., 2006; KO; DELINE; VENOLIA, 2007; BURGE; BROWN, 2008; HASSAN, 2008; SHIHAB; ZHEN; HASSAN, 2009; GARCIA et al., 2012; BRUNET et al., 2014; STOREY et al., 2014; HESSE et al., 2016; ALKADHI et al., 2017; BHAT et al., 2017; ALKADHI et al., 2018; KLEEBAUM et al., 2018).

Dessa forma, este trabalho se propôs a caracterizar mudanças arquiteturais em projetos de longa duração utilizando a mineração de mensagens de *commit* com informações sobre mudanças. Para isso, foram realizados 3 estudos empíricos. Inicialmente, foi realizada a mineração no repositório de um grande projeto de software (KDELibs), em busca de informações sobre mudanças na arquitetura do projeto. Nesse trabalho, foram utilizadas como fontes de informação as mensagens de *commit* registradas ao longo da evolução deste projeto. As mensagens encontradas foram analisadas e permitiram caracterizar: (i) o perfil dos colaboradores de projetos de Software Livre que modificam a arquitetura do projeto, (ii) os tópicos arquiteturais mais abordados, (iii) o período do projeto quando mais ocorrem as mudanças na arquitetura, (iv) o tamanho das mensagens que descrevem mudanças na arquitetura, (v) a quantidade de arquivos/módulos alterados, e (vi) a extensão das alterações sobre o código-fonte.

De posse desses *commits* minerados, foi realizado o segundo estudo que comparou *commits* onde a arquitetura do projeto KDELibs não foi alterada com os *commits* obtidos

no primeiro estudo realizado no âmbito desta pesquisa de doutorado (que continham mudanças na arquitetura do projeto). Os *commits* sem mudanças na arquitetura foram escolhidos aleatoriamente, buscando, no entanto, *commits* em períodos semelhantes àqueles em que foram encontrados os *commits* com mudanças na arquitetura. Como resultado, o estudo mostra, empiricamente, que os valores dessas métricas variam de forma diferente, quando comparados os *commits* dos dois conjuntos analisados. Além disso, ele mostra que, nas mudanças arquiteturais, quando comparadas com mudanças sem modificações na arquitetura, em geral, o tamanho do código tem uma tendência maior a aumentar, a complexidade ciclomática tem uma tendência maior a aumentar (embora a complexidade, considerando todas as métricas de complexidade estudadas, tenha uma tendência maior a se manter). Na mesma lógica, as conexões aferentes entre classes, a complexidade estrutural e as respostas entre classes também apresentaram tendência maior a aumentar ante a comparação estabelecida.

Esses resultados foram referendados pelo terceiro estudo empírico realizado. Esse estudo empírico se materializou através de um *survey* onde desenvolvedores e pesquisadores da engenharia de software puderam descrever seu cenário atual de trabalho, sob o ponto de vista da arquitetura de software e da compreensão global do projeto, e opinar sobre os resultados obtidos nesta pesquisa de doutorado. Nesse estudo foram apresentados os resultados obtidos nas fases anteriores da pesquisa, tais como mensagens de *commit* contendo informações sobre mudanças na arquitetura e a variação nos valores de métricas (de tamanho, complexidade, acoplamento e coesão) para *commits* com alterações na arquitetura e *commits* sem alteração na arquitetura do projeto. Salvo raras exceções, os desenvolvedores e pesquisadores acreditam que a mineração de repositórios é um caminho para se obter informações sobre mudanças na arquitetura ao longo de sua evolução. Eles também reconheceram a familiaridade dos impactos causados por esse tipo de mudança no projeto sobre o código-fonte dos mesmos. Os desenvolvedores e pesquisadores que recusaram parcialmente os resultados obtidos nos estudos anteriores, ressaltaram o potencial dessa nova fonte de informações arquiteturais e reconheceram-na como uma nova fonte de informação, desde que combinada com outros artefatos produzidos pelos projetos.

Ao final desta pesquisa é importante destacar os pontos relacionados que ainda podem ser explorados, e que pretende-se continuar pesquisando:

- Novas fontes de dados não-estruturadas: Ao longo desta tese, outras fontes de dados não-estruturadas foram apontadas como fonte de informação para a mineração de outros dados do projeto. Dessa forma, acredita-se que locais onde o projeto é discutido, como em canais de IRC utilizados pela comunidade de Software Livre, atas de reunião onde empresas de software não-livre discutem o andamento dos projetos, e informações oriundas de *issue trackers* e *bug trackers* podem conter valiosos detalhes sobre problemas e mudanças na arquitetura de projetos;

- Novas visões arquiteturais: Esta pesquisa apontou tópicos que estão relacionados à arquitetura de software. Considerando um item tão abstrato como a arquitetura de um projeto de software, é possível que, na visão de outros pesquisadores e desenvolvedores, novos tópicos arquiteturais sejam acrescentados à lista apresentada

por esta tese. Isso iniciará um novo ciclo de avaliações de informações, que podem explorar, inclusive, a base de informações disponibilizada por esse trabalho em torno do KDELibs;

- Caracterização de novos projetos de Software Livre: Este trabalho buscou um projeto com características ímpares, e que podem, ou não, se reproduzirem em outros projetos dessa natureza. Um dos intentos do autor desta tese e de seus orientadores é dar continuidade à esse estudo tendo como fonte projetos com características parecidas e projetos com outros tipos de características. É preciso estudar se essa fonte de informação está disponível em projetos menores, ou com menos interfaces com outros projetos, por exemplo;

- Automatização do processo de identificação de informações arquiteturais através de traços arquiteturais: A exploração de mensagens de *commit* apresentada neste trabalho ainda é semiautomatizada. Um dos desdobramentos previstos nesse trabalho é a construção de uma ferramenta que utilize técnicas de aprendizado de máquina, visando diminuir a interação dos usuários, minimizando, assim, a relação entre o grau de experiência e conhecimento dos projetos explorados por parte dos usuários e o alcance das informações mineradas, e;

- Novas palavras-chaves arquiteturais: Este projeto teve como base um conjunto de palavras-chaves definidas por especialistas em arquitetura de software, e portanto trata-se de uma fonte confiável. No entanto, dado o nível de abstração, já citado ao longo deste texto, que acompanha as referências relacionadas à arquitetura de software, é possível que outros termos sejam utilizados por desenvolvedores de projetos de Software Livre que não tenham sido alcançados por esses especialistas. A utilização de técnicas de identificação de palavras-chaves pode representar um novo horizonte para a mineração de informações arquiteturais.

BIBLIOGRAPHY

ABREU, F. B.; GOULÃO, M.; ESTEVES, R. Toward the design quality evaluation of object-oriented software systems. In: *Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA*. [S.l.: s.n.], 1995. p. 44–57.

AGGARWAL, K. et al. Empirical study of object-oriented metrics. *Journal of Object Technology*, v. 5, n. 8, p. 149–173, 2006.

ALKADHI, R. et al. Rationale in development chat messages: an exploratory study. In: IEEE PRESS. *Proceedings of the 14th International Conference on Mining Software Repositories*. [S.l.], 2017. p. 436–446.

ALKADHI, R. et al. How do developers discuss rationale? In: IEEE. *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. [S.l.], 2018. p. 357–369.

ALSHAYEB, M. Refactoring effect on cohesion metrics. In: IEEE. *2009 International Conference on Computing, Engineering and Information*. [S.l.], 2009. p. 3–7.

ANTONIOL, G. et al. Recovering traceability links between code and documentation. *IEEE transactions on software engineering*, IEEE, v. 28, n. 10, p. 970–983, 2002.

BACHMANN, F. et al. Designing software architectures to achieve quality attribute requirements. *IEE Proceedings-Software*, IET, v. 152, n. 4, p. 153–165, 2005.

BALIEIRO, M. A. et al. Ossnetwork: Um ambiente para estudo de comunidades de software livre usando redes sociais. In: *Experimental Software Engineering Latin America Workshop*. [S.l.: s.n.], 2007. p. 33–424.

BALUSHI, T. H. A. et al. Elicito: a quality ontology-guided nfr elicitation tool. In: SPRINGER. *International Working Conference on Requirements Engineering: Foundation for Software Quality*. [S.l.], 2007. p. 306–319.

BASILI, V. R.; ROMBACH, H. D. *TAME: Integrating measurement into software environments*. [S.l.], 1987.

BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. [S.l.: s.n.], 1998. 998 p.

BEHNAMGHADER, P. et al. A large-scale study of architectural evolution in open-source software systems. *Empirical Software Engineering*, Springer, v. 22, n. 3, p. 1146–1193, 2017.

BENGTSSON, P. Towards maintainability metrics on software architecture: An adaptation of object-oriented metrics. In: *First nordic workshop on software architecture, ronneby*. [S.l.: s.n.], 1998.

BHAT, M. et al. Automatic extraction of design decisions from issue management systems: A machine learning based approach. In: SPRINGER. *European Conference on Software Architecture*. [S.l.], 2017. p. 138–154.

BIEGEL, B. et al. Comparison of similarity metrics for refactoring detection. In: *Proceedings of the 8th working conference on mining software repositories*. [S.l.: s.n.], 2011. p. 53–62.

BLEI, D.; NG, A. Y.; JORDAN, M. I. Latent Dirichlet Allocation. *Jmlr*, v. 3, p. 993–1022, 2003. ISSN 1532-4435.

BOEHM, B. W. Verifying and validating software requirements and design specifications. *IEEE software*, IEEE Computer Society, v. 1, n. 1, p. 75, 1984.

BOJIC, D.; VELASEVIC, D. A use-case driven method of architecture recovery for program understanding and reuse reengineering. In: IEEE. *csmr*. [S.l.], 2000. p. 23.

BRUNET, J. a. et al. Do developers discuss design? *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, p. 340–343, 2014. Disponível em: ⟨http://dl.acm.org.prox.lib.ncsu.edu/citation.cfm?id=2597073.2597115⟩.

BURGE, J. E.; BROWN, D. C. Software engineering using rationale. *Journal of Systems and Software*, Elsevier, v. 81, n. 3, p. 395–413, 2008.

BUSCHMANN, F.; HENNEY, K.; SCHIMDT, D. *Pattern-Oriented Software Architecture: On Patterns And Pattern Language, Volume 5*. [S.l.]: John wiley & sons, 2007.

BUSCHMANN, F. et al. *A system of patterns: Pattern-oriented software architecture*. [S.l.]: Wiley New York, 1996.

CAI, Y. et al. Design rule spaces: A new model for representing and analyzing software architecture. *IEEE Transactions on Software Engineering*, IEEE, v. 45, n. 7, p. 657–682, 2018.

CAPILLA, R. et al. 10 years of software architecture knowledge management: Practice and future. *Journal of Systems and Software*, Elsevier, v. 116, p. 191–205, 2016.

CAPRA, E. et al. A survey on firms' participation in open source community projects. In: SPRINGER. *IFIP International Conference on Open Source Systems*. [S.l.], 2009. p. 225–236.

CHARMAZ, K. *Constructing grounded theory: A practical guide through qualitative analysis*. [S.l.]: sage, 2006.

CHIDAMBER, S. R.; KEMERER, C. F. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, IEEE, v. 20, n. 6, p. 476–493, 1994.

CHUNG, L. et al. Non-functional requirements. *Software Engineering*, Kluwer Academic, 2000.

CINNÉIDE, M. Ó. et al. Experimental assessment of software metrics using automated refactoring. In: *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*. [S.l.: s.n.], 2012. p. 49–58.

CLEARY, B.; EXTON, C. Facilitating architectural recovery, description & reuse through cognitive mapping. *STEP 2005*, p. 122–126, 2005.

CLEMENTS, P. A survey of architecture description languages. *Proceedings of the 8th International Workshop on Software Specification and Design*, n. March, p. 16–25, 1996. Disponível em: ⟨http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=501143⟩.

CLEMENTS, P. et al. *Documenting Software Architectures*. [S.l.: s.n.], 2010. 592 p. ISBN 0201703726.

CLEMENTS, P. et al. Documenting software architectures: views and beyond. In: IEEE. *25th International Conference on Software Engineering, 2003. Proceedings.* [S.l.], 2003. p. 740–741.

CODD, E. F. *Recent Investigations in Relational Data Base Systems.* [S.l.]: IBM Thomas J. Watson Research Division, 1974.

COHEN, J. Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit. *Psychological Bulletin*, v. 70, n. 4, p. 213–220, 1968. ISSN 1939-1455.

CORAZZA, A.; MARTINO, S. D.; SCANNIELLO, G. A probabilistic based approach towards software system clustering. In: IEEE. *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*. [S.l.], 2010. p. 88–96.

CORBIN, J.; STRAUSS, A. Basics of qualitative research: Techniques and procedures for developing grounded theory. Sage Publications, Inc, 2008.

CUNNINGHAM, H. Gate, a general architecture for text engineering. *Computers and the Humanities*, Springer, v. 36, n. 2, p. 223–254, 2002.

DAGPINAR, M.; JAHNKE, J. H. Predicting maintainability with object-oriented metrics-an empirical comparison. In: IEEE. *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.* [S.l.], 2003. p. 155–164.

DARCY, D. P. et al. The structural complexity of software an experimental test. *IEEE Transactions on Software Engineering*, IEEE, v. 31, n. 11, p. 982–995, 2005.

DEURSEN, A. V. et al. Symphony: View-driven software architecture reconstruction. In: IEEE. *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on*. [S.l.], 2004. p. 122–132.

DÍAZ-PACE, J. A. et al. Producing just enough documentation: An optimization approach applied to the software architecture domain. *Journal on Data Semantics*, Springer, v. 5, n. 1, p. 37–53, 2016.

DING, L.; MEDVIDOVIC, N. Focus: A light-weight, incremental approach to software architecture recovery and evolution. In: IEEE. *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*. [S.l.], 2001. p. 191–200.

DING, W. et al. How do open source communities document software architecture: An exploratory survey. In: IEEE. *2014 19th International conference on engineering of complex computer systems*. [S.l.], 2014. p. 136–145.

DUTOIT, A. H.; PAECH, B. Rationale management in software engineering. In: *Handbook of Software Engineering and Knowledge Engineering: Volume I: Fundamentals*. [S.l.]: World Scientific, 2001. p. 787–815.

EADDY, M. et al. Do crosscutting concerns cause defects? *Software Engineering, IEEE Transactions on*, IEEE, v. 34, n. 4, p. 497–515, 2008.

EBERT, C. Dealing with nonfunctional requirements in large software systems. *Annals of Software Engineering*, Springer, v. 3, n. 1, p. 367–395, 1997.

EBERT, J. et al. Gupro-generic understanding of programs an overview. *Electronic Notes in Theoretical Computer Science*, Elsevier, v. 72, n. 2, p. 47–56, 2002.

EIXELSBERGER, W. et al. Software architecture recovery of a program family. In: IEEE. *Software Engineering, 1998. Proceedings of the 1998 International Conference on*. [S.l.], 1998. p. 508–511.

FARID, H.; AZAM, F.; IQBAL, M. A. Minimizing the risk of architectural decay by using architecture-centric evolution process. *International Journal of Computer Science, Engineering and Applications*, Academy & Industry Research Collaboration Center (AIRCC), v. 1, n. 5, p. 1, 2011.

FAVRE, J.-M. Cacophony: Metamodel-driven software architecture reconstruction. In: IEEE. *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*. [S.l.], 2004. p. 204–213.

FENTON, N.; BIEMAN, J. *Software metrics: a rigorous and practical approach*. [S.l.]: CRC press, 2014.

FIUTEM, R. et al. Art: an architectural reverse engineering environment. *Journal of Software Maintenance*, Citeseer, v. 11, n. 5, p. 339–364, 1999.

FREITAS, E. P. et al. Using aspect-oriented concepts in the requirements analysis of distributed real-time embedded systems. In: *Embedded system design: Topics, techniques and trends*. [S.l.]: Springer, 2007. p. 221–230.

GARCIA, J. et al. A framework for obtaining the ground-truth in architectural recovery. *Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012*, p. 292–296, 2012.

GARLAN, D. Software architecture: a travelogue. In: *Proceedings of the on Future of Software Engineering*. [S.l.: s.n.], 2014. p. 29–39.

GARLAN, D.; SHAW, M. An introduction to software architecture. In: *Advances in software engineering and knowledge engineering*. [S.l.]: World Scientific, 1993. p. 1–39.

GLASER, B. G. *Theoretical sensitivity. mill valley*. [S.l.]: CA: Sociology Press, 1978.

GRAAF, K. A. de et al. An exploratory study on ontology engineering for software architecture documentation. *Computers in Industry*, Elsevier, v. 65, n. 7, p. 1053–1064, 2014.

GUAN, T.; WONG, K.-F. Kps: a web information mining algorithm. *Computer Networks*, Elsevier, v. 31, n. 11-16, p. 1495–1507, 1999.

GUO, G. Y.; ATLEE, J. M.; KAZMAN, R. *A software architecture reconstruction method*. [S.l.]: Springer, 1999.

GURP, J. V.; BOSCH, J. Design erosion: problems and causes. *Journal of systems and software*, Elsevier, v. 61, n. 2, p. 105–119, 2002.

HAITZER, T.; NAVARRO, E.; ZDUN, U. Reconciling software architecture and source code in support of software evolution. *Journal of Systems and Software*, Elsevier, v. 123, p. 119–144, 2017.

HALL, M. et al. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, ACM New York, NY, USA, v. 11, n. 1, p. 10–18, 2009.

HARRIS, D. R.; REUBENSTEIN, H. B.; YEH, A. S. Reverse engineering to the architectural level. In: ACM. *Proceedings of the 17th international conference on Software engineering*. [S.l.], 1995. p. 186–195.

HARRISON, R.; COUNSELL, S.; NITHI, R. An overview of object-oriented design metrics. In: IEEE. *Proceedings Eighth IEEE International Workshop on Software Technology and Engineering Practice incorporating Computer Aided Software Engineering*. [S.l.], 1997. p. 230–235.

HASSAN, A. E. The road ahead for mining software repositories. In: IEEE. *Frontiers of Software Maintenance, 2008. FoSM 2008.* [S.l.], 2008. p. 48–57.

HASSAN, A. E.; HOLT, R. C. Using development history sticky notes to understand software architecture. In: IEEE. *Program Comprehension, 2004. Proceedings. 12th IEEE International Workshop on*. [S.l.], 2004. p. 183–192.

HATCH, A. *Software Architecture Visualisation*. Tese (Doutorado) — Durham University, 2004.

HEESCH, U. V.; AVGERIOU, P.; HILLIARD, R. A documentation framework for architecture decisions. *Journal of Systems and Software*, Elsevier, v. 85, n. 4, p. 795–820, 2012.

HENDERSON-SELLERS, B. *Object-oriented metrics: measures of complexity*. [S.l.]: Prentice-Hall, Inc., 1995.

HESSE, T.-M. et al. Documented decision-making strategies and decision knowledge in open source projects: An empirical study on firefox issue reports. *Information and Software Technology*, Elsevier, v. 79, p. 36–51, 2016.

HINDLE, A.; GERMAN, D. M.; HOLT, R. What do large commits tell us?: a taxonomical study of large commits. In: ACM. *Proceedings of the 2008 international working conference on Mining software repositories (MSR 2008)*. [S.l.], 2008. p. 99–108.

HIRATA, Y.; MIZUNO, O. Do Comments Explain Codes Adequately? Investigation by Text Filtering. *Proceedings of the 8th Working Conference on Mining Software Repositories*, p. 242–245, 2011.

HITZ, M.; MONTAZERI, B. *Measuring coupling and cohesion in object-oriented systems*. [S.l.]: Citeseer, 1995.

HOLT, R. C. Structural manipulations of software architecture using tarski relational algebra. In: IEEE. *Reverse Engineering, 1998. Proceedings. Fifth Working Conference on*. [S.l.], 1998. p. 210–219.

HOTELLING, H. Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, Warwick & York, v. 24, n. 6, p. 417, 1933.

HUYNH, S. et al. Automatic Modularity Conformance Checking. In: . [S.l.: s.n.], 2008. v. 1, p. 411–420. ISBN 9781605580791.

INCE, D.; SHEPPARD, M. System design metrics: a review and perspective. In: IET. *Second IEE/BCS Conference: Software Engineering, 1988 Software Engineering 88*. [S.l.], 1988. p. 23–27.

IYER, S. S. *An analytical study of metrics and refactoring*. Tese (Doutorado) — Citeseer, 2009.

JIN, C.; LIU, J.-A. Applications of support vector mathine and unsupervised learning for predicting maintainability using object-oriented metrics. In: IEEE. *2010 Second International Conference on Multimedia and Information Technology*. [S.l.], 2010. v. 1, p. 24–27.

JURECZKO, M.; SPINELLIS, D. Using object-oriented design metrics to predict software defects. *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, p. 69–81, 2010.

KAGDI, H.; COLLARD, M. L.; MALETIC, J. I. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of software maintenance and evolution: Research and practice*, Wiley Online Library, v. 19, n. 2, p. 77–131, 2007.

KASUNIC, M. *The state of software measurement practice: results of 2006 survey*. [S.l.], 2006.

KAUR, H.; VERMA, G. N. A case study upon non-functional requirements of online banking system. *Int. J. Comput. Appl. Technol. Res*, v. 4, p. 220–225, 2015.

KAZMAN, R.; CARRIÈRE, S. J. Playing detective: Reconstructing software architecture from available evidence. *Automated Software Engineering*, Springer, v. 6, n. 2, p. 107–138, 1999.

KAZMAN, R.; O'BRIEN, L.; VERHOEF, C. *Architecture Reconstruction Guidelines Third Edition*. [S.l.], 2003.

KIM, S. et al. Predicting faults from cached history. In: IEEE COMPUTER SOCIETY. *Proceedings of the 29th international conference on Software Engineering*. [S.l.], 2007. p. 489–498.

KITCHENHAM, B. A.; PFLEEGER, S. L. Principles of survey research part 2: designing a survey. *ACM SIGSOFT Software Engineering Notes*, ACM New York, NY, USA, v. 27, n. 1, p. 18–20, 2002.

KITCHENHAM, B. A.; PFLEEGER, S. L. Principles of survey research: part 3: constructing a survey instrument. *ACM SIGSOFT Software Engineering Notes*, ACM New York, NY, USA, v. 27, n. 2, p. 20–24, 2002.

KLEEBAUM, A. et al. Tool support for decision and usage knowledge in continuous software engineering. CEUR-WS. org, 2018.

KNODEL, J. et al. Static evaluation of software architectures. In: IEEE. *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*. [S.l.], 2006. p. 10–pp.

KO, A. J.; DELINE, R.; VENOLIA, G. Information needs in collocated software development teams. In: IEEE. *Software Engineering, 2007. ICSE 2007. 29th International Conference on*. [S.l.], 2007. p. 344–353.

KOGUT, P.; CLEMENTS, P. Features of Architecture Description Languages. *Proceedings of the Eighth International Workshop on Software Specification and Design*, n. April, p. 1–13, 1995.

KRIKHAAR, R. L. *Software Architecture Reconstruction*. 148 p. Tese (Doutorado) — Universiteit van Amsterdam, june 1999.

KUHN, A.; DUCASSE, S.; GÍRBA, T. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, Elsevier, v. 49, n. 3, p. 230–243, 2007.

KURTANOVIĆ, Z.; MAALEJ, W. Mining user rationale from software reviews. In: IEEE. *2017 IEEE 25th International Requirements Engineering Conference (RE)*. [S.l.], 2017. p. 61–70.

LAMSWEERDE, A. V. Elaborating security requirements by construction of intentional anti-models. In: IEEE. *Proceedings. 26th International Conference on Software Engineering*. [S.l.], 2004. p. 148–157.

LANDIS, J. R.; KOCK, G. G. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, v. 33, n. 1, p. 159–174, 1977. Disponível em: ⟨http://doi.wiley.com/10.1002/9780470057339.vai016⟩.

LARMAN, C. *Applying UML and patterns: an introduction to object oriented analysis and design and interative development*. [S.l.]: Pearson Education India, 2012.

LEHMAN, M. M. et al. Metrics and laws of software evolution-the nineties view. In: IEEE. *Proceedings Fourth International Software Metrics Symposium*. [S.l.], 1997. p. 20–32.

LI, W.; HENRY, S. Object-oriented metrics that predict maintainability. *Journal of systems and software*, Elsevier, v. 23, n. 2, p. 111–122, 1993.

LINDERS, B. Cmmi for development, guidelines for process integration and product improvement, third edition. *Software Quality Professional*, v. 13, n. 3, p. 33–34, 06 2011. Copyright - Copyright American Society for Quality Jun 2011; Pessoas - Chrissis, Mary Beth; Konrad, Mike; Shrum, Sandy; Última atualização em - 2011-06-21; SubjectsTermNotLitGenreText - Chrissis, Mary Beth; Konrad, Mike; Shrum, Sandy. Disponível em: ⟨https://search.proquest.com/docview/873045040?accountid=14536⟩.

LORENZ, M.; KIDD, J. *Object-oriented software metrics: a practical guide*. [S.l.]: Prentice-Hall, Inc., 1994.

LUNGU, M.; LANZA, M.; GÎRBA, T. Package patterns for visual architecture recovery. In: IEEE. *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*. [S.l.], 2006. p. 10–pp.

MAALEJ, W.; HAPPEL, H.-J. Can Development Work Describe Itself? *7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, p. 191–200, 2010. Disponível em: ⟨http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5463344⟩.

MAFFORT, C. et al. Mining architectural violations from version history. *Empirical Software Engineering*, Springer, v. 21, n. 3, p. 854–895, 2016.

MAIER, M. W.; EMERY, D.; HILLIARD, R. Software architecture: introducing ieee standard 1471. *Computer*, IEEE, v. 34, n. 4, p. 107–109, 2001.

MALTON, A. J. Recovering general layering and subsystems in dependency graphs. *STEP 2005*, p. 136–141, 2005.

MANCORIDIS, S. et al. Bunch: A clustering tool for the recovery and maintenance of software system structures. In: IEEE. *Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on*. [S.l.], 1999. p. 50–59.

MANN, H. B.; WHITNEY, D. R. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, JSTOR, p. 50–60, 1947.

MAQBOOL, O.; BABRI, H. A. The weighted combined algorithm: A linkage algorithm for software clustering. In: IEEE. *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on*. [S.l.], 2004. p. 15–24.

MAQBOOL, O.; BABRI, H. A. Hierarchical clustering for software architecture recovery. *Software Engineering, IEEE Transactions on*, IEEE, v. 33, n. 11, p. 759–780, 2007.

MARCUS, A.; MALETIC, J. I. Recovering documentation-to-source-code traceability links using latent semantic indexing. In: IEEE COMPUTER SOCIETY. *Proceedings of the 25th international conference on software engineering*. [S.l.], 2003. p. 125–135.

MARTIN, R. Oo design quality metrics-an analysis of dependencies. In: *Proc. Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA'94*. [S.l.: s.n.], 1994.

MASKERI, G.; SARKAR, S.; HEAFIELD, K. Mining business topics in source code using latent dirichlet allocation. In: ACM. *Proceedings of the 1st India software engineering conference*. [S.l.], 2008. p. 113–120.

MCCABE, T. J. A complexity measure. *IEEE Transactions on software Engineering*, IEEE, n. 4, p. 308–320, 1976.

MEIRELLES, P. et al. A study of the relationships between source code metrics and attractiveness in free software projects. In: IEEE. *2010 Brazilian Symposium on Software Engineering*. [S.l.], 2010. p. 11–20.

MEIRELLES, P. R. M. *Monitoramento de métricas de código-fonte em projetos de software livre*. Tese (Doutorado) — Universidade de São Paulo, 2013.

MELO, I. et al. Perceptions of 395 developers on software architecture's documentation and conformance. In: IEEE. *2016 X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS)*. [S.l.], 2016. p. 81–90.

MENDONÇA, N. C.; KRAMER, J. An approach for recovering distributed system architectures. *Automated Software Engineering*, Springer, v. 8, n. 3-4, p. 311–354, 2001.

MENS, K. et al. Co-evolving code and design with intensional views: A case study. *Computer Languages, Systems & Structures*, Elsevier, v. 32, n. 2, p. 140–156, 2006.

MERTEN, T.; MAGER, B.; PAECH, B. Classifying Unstructured Data into Natural Language Text and Technical Information. *Proceedings of the 11th Working Conference on Mining Software Repositories*, v. 1, p. 300–303, 2014.

MIODONSKI, P. et al. Evaluation of software architectures with eclipse. *Institute for Empirical Software Engineering (IESE)-Report*, v. 107, 2004.

MISRA, S. C.; BHAVSAR, V. C. Relationships between selected software measures and latent bug-density: Guidelines for improving quality. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2003. p. 724–732.

MIZUNO, O. et al. Spam filter based approach for finding fault-prone software modules. *Proceedings - ICSE 2007 Workshops: Fourth International Workshop on Mining Software Repositories, MSR 2007*, p. 7–10, 2007.

MOSER, R.; PEDRYCZ, W.; SUCCI, G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: IEEE. *2008 ACM/IEEE 30th International Conference on Software Engineering*. [S.l.], 2008. p. 181–190.

MOTTA, T. O.; SOUZA, R. R. Gomes e; SANT'ANNA, C. Characterizing architectural information in commit messages: an exploratory study. In: ACM. *Proceedings of the XXXII Brazilian Symposium on Software Engineering*. [S.l.], 2018. p. 12–21.

MURPHY, G. C. *Lightweight Structural Summarization As an Aid to Software Evolution*. Tese (Doutorado) — University of Washington, 1996. AAI9704521.

MUTHANNA, S. et al. A maintainability model for industrial software systems using design level metrics. In: IEEE. *Proceedings Seventh Working Conference on Reverse Engineering*. [S.l.], 2000. p. 248–256.

NEU, S. et al. Telling stories about gnome with complicity. In: IEEE. *2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*. [S.l.], 2011. p. 1–8.

NUÑEZ-VARELA, A. S. et al. Source code metrics: A systematic mapping study. *Journal of Systems and Software*, Elsevier, v. 128, p. 164–197, 2017.

O'BRIEN, L.; SMITH, D.; LEWIS, G. Supporting migration to services using software architecture reconstruction. In: IEEE. *Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on*. [S.l.], 2005. p. 81–91.

OHBA, M.; GONDOW, K. Toward mining concept keywords from identifiers in large software projects. In: ACM. *ACM SIGSOFT Software Engineering Notes*. [S.l.], 2005. v. 30, n. 4, p. 1–5.

OLAGUE, H. M.; ETZKORN, L. H.; COX, G. W. An entropy-based approach to assessing object-oriented software maintainability and degradation-a method and case study. In: *Software Engineering Research and Practice*. [S.l.: s.n.], 2006. p. 442–452.

OREIZY, P.; MEDVIDOVIC, N.; TAYLOR, R. N. Architecture-based runtime software evolution. In: IEEE. *Proceedings of the 20th international conference on Software engineering*. [S.l.], 1998. p. 177–186.

OZKAYA, M. What is software architecture to practitioners: A survey. In: IEEE. *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*. [S.l.], 2016. p. 677–686.

PANICHELLA, S. et al. How developers' collaborations identified from different sources tell us about code changes. In: IEEE. *2014 IEEE International Conference on Software Maintenance and Evolution*. [S.l.], 2014. p. 251–260.

PATTISON, D. S.; BIRD, C. a.; DEVANBU, P. T. Talk and work: A preliminary report. *Proceedings - International Conference on Software Engineering*, p. 113–116, 2008. ISSN 02705257. Disponível em: ⟨http://www.scopus.com/inward/record.url?eid=2-s2.0-57049135108\&partnerID=tZOtx3y1⟩.

PEARSON, K. X. on the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, Taylor & Francis, v. 50, n. 302, p. 157–175, 1900.

PERRY, D. E.; WOLF, A. L. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, ACM, v. 17, n. 4, p. 40–52, 1992.

PINZGER, M. *ArchView - Analyzing Evolutionary Aspects of Complex Software Systems*. Tese (Doutorado) — Vienna Univ. of Technology, 2005.

PINZGER, M. et al. Revealer: A lexical pattern matcher for architecture recovery. In: IEEE. *Reverse Engineering, 2002. Proceedings. Ninth Working Conference on*. [S.l.], 2002. p. 170–178.

PORTER, M. F. An algorithm for suffix stripping. *Program*, MCB UP Ltd, v. 14, n. 3, p. 130–137, 1980.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016.

PUNTER, T. et al. Conducting on-line surveys in software engineering. In: IEEE. *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.* [S.l.], 2003. p. 80–88.

REAL, R.; VARGAS, J. M. The probabilistic basis of jaccard's index of similarity. *Systematic biology*, Society of Systematic Biologists, v. 45, n. 3, p. 380–385, 1996.

RIEDIGER, J. E. B. K. V.; WINTER, A. Gupro-generic understanding of programs. *Electronic Notes in Theoretical Computer Science*, v. 72, n. 2, 2002.

RIVA, C. *View-Based Software Architecture Reconstruction*. Tese (Doutorado) — Technical University of Vienna, 2004.

RIVA, C. et al. Establishing a software architecting environment. *Proceedings - Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, p. 188–197, 2004.

ROGERS, B. et al. Using text mining techniques to extract rationale from existing documentation. In: *Design Computing and Cognition'14*. [S.l.]: Springer, 2015. p. 457–474.

ROMANO, J. et al. Appropriate statistics for ordinal level data: Should we really be using t-test and cohen'sd for evaluating group differences on the nsse and other surveys. In: *annual meeting of the Florida Association of Institutional Research*. [S.l.: s.n.], 2006. p. 1–33.

ROST, D. et al. Software architecture documentation for developers: a survey. In: SPRINGER. *European Conference on Software Architecture*. [S.l.], 2013. p. 72–88.

SARTIPI, K. Software architecture recovery based on pattern matching. *Software Maintenance, 2003. ICSM 2003.*, p. 293–296, 2003. Disponível em: ⟨http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1235434⟩.

SEAMAN, C. B. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, IEEE, v. 25, n. 4, p. 557–572, 1999. ISSN 0098-5589.

SHAHBAZIAN, A. et al. Recovering architectural design decisions. In: IEEE. *2018 IEEE International Conference on Software Architecture (ICSA)*. [S.l.], 2018. p. 95–9509.

SHANNON, C. E. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review*, ACM New York, NY, USA, v. 5, n. 1, p. 3–55, 2001.

SHARMA, M. et al. A comparative study of static object oriented metrics. *International Journal of Advancements in Technology*, v. 3, n. 1, p. 25–34, 2012.

SHEPPERD, M. Design metrics: an empirical analysis. *Software Engineering Journal*, IET, v. 5, n. 1, p. 3–10, 1990.

SHIHAB, E.; ZHEN, M. J.; HASSAN, A. E. On the use of internet relay chat (IRC) meetings by developers of the GNOME GTK+ project. *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories, MSR 2009*, p. 107–110, 2009.

SIMON, F.; STEINBRUCKNER, F.; LEWERENTZ, C. Metrics based refactoring. In: IEEE. *Proceedings Fifth European Conference on Software Maintenance and Reengineering*. [S.l.], 2001. p. 30–38.

SIMONS, C.; SINGER, J.; WHITE, D. R. Search-based refactoring: Metrics are not enough. In: SPRINGER. *International Symposium on Search Based Software Engineering*. [S.l.], 2015. p. 47–61.

ŚLIWERSKI, J.; ZIMMERMANN, T.; ZELLER, A. When do changes induce fixes? In: ACM. *ACM sigsoft software engineering notes*. [S.l.], 2005. v. 30, n. 4, p. 1–5.

SMITH, E. et al. Improving developer participation rates in surveys. In: IEEE. *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*. [S.l.], 2013. p. 89–92.

SOMMERVILLE, I. *Engenharia de Software. 9o Edição*. [S.l.]: Pearson Education-BR, 2011.

STEVENS, W.; MYERS, G.; CONSTANTINE, L. Structured design. In: _____. *Classics in Software Engineering*. USA: Yourdon Press, 1979. p. 205–232. ISBN 0917072146.

STOERMER, C.; O'BRIEN, L. Map-mining architectures for product line evaluations. In: IEEE. *Software Architecture, 2001. Proceedings. Working IEEE/IFIP Conference on*. [S.l.], 2001. p. 35–44.

STOERMER, C.; O'BRIEN, L.; VERHOEF, C. Moving towards quality attribute driven software architecture reconstruction. In: IEEE. *null*. [S.l.], 2003. p. 46.

STOERMER, C. et al. Model-centric software architecture reconstruction. *Software: Practice and Experience*, Wiley Online Library, v. 36, n. 4, p. 333–363, 2006.

STOREY, M.-A. et al. The (r) evolution of social media in software engineering. In: ACM. *Proceedings of the on Future of Software Engineering*. [S.l.], 2014. p. 100–116.

SUTCLIFFE, A. G.; MINOCHA, S. Scenario-based analysis of non-functional requirements. In: *REFSQ*. [S.l.: s.n.], 1998. v. 98, p. 219–234.

TANG, A. et al. Human aspects in software architecture decision making: a literature review. In: IEEE. *2017 IEEE International Conference on Software Architecture (ICSA)*. [S.l.], 2017. p. 107–116.

TANG, M.-H.; KAO, M.-H.; CHEN, M.-H. An empirical study on object-oriented metrics. In: IEEE. *Proceedings sixth international software metrics symposium (Cat. No. PR00403)*. [S.l.], 1999. p. 242–249.

TERCEIRO, A. et al. Analizo: an extensible multi-language source code analysis and visualization toolkit. In: *Brazilian conference on software: theory and practice (Tools Session)*. [S.l.: s.n.], 2010.

THWIN, M. M. T.; QUAH, T.-S. Application of neural networks for software quality prediction using object-oriented metrics. *Journal of systems and software*, Elsevier, v. 76, n. 2, p. 147–156, 2005.

TZERPOS, V.; HOLT, R. C. Acdc: An algorithm for comprehension-driven clustering. In: IEEE. *wcre*. [S.l.], 2000. p. 258.

WEINREICH, R.; GROHER, I. Software architecture knowledge management approaches and their support for knowledge management activities: A systematic literature review. *Information and Software Technology*, Elsevier, v. 80, p. 265–286, 2016.

WILCOXON, F. Individual comparisons by ranking methods. In: *Breakthroughs in statistics*. [S.l.]: Springer, 1992. p. 196–202.

WOHLIN, C. et al. *Experimentation in software engineering*. [S.l.]: Springer Science & Business Media, 2012.

XENOS, M. et al. Object-oriented metrics-a survey. In: *Proceedings of the FESMA*. [S.l.: s.n.], 2000. p. 1–10.

YAN, H. et al. Discotect: A system for discovering architectures from running systems. In: IEEE COMPUTER SOCIETY. *Proceedings of the 26th International Conference on Software Engineering*. [S.l.], 2004. p. 470–479.

YING, A. T.; WRIGHT, J. L.; ABRAMS, S. Source code that talks: an exploration of eclipse task comments and their implication to repository mining. In: *Proceedings of 2nd International Workshop on Mining Software Repositories (MSR 2005)*. [S.l.]: ACM, 2005. p. 53–57.

Appendix

A

GUIA DE UTILIZAÇÃO DE TRAÇOS ARQUITETURAIS COMO PROVEDORES DE INFORMAÇÕES SOBRE MODIFICAÇÃO ARQUITETURAL

Este apêndice apresenta o passo a passo para a utilização da técnica de mineração de informações sobre mudanças arquiteturais a partir da identificação de Traços Arquiteturais em mensagens de *commit*.

Para proceder uma avaliação desse tipo, são necessários os seguintes artefatos:

A) **Software Analizo**. Disponível em: http://www.analizo.org/

B) **Ferramenta ATM**. Disponível em: https://motta-tiago.github.io/doctorate/atm.html

C) **Cliente Git**. Disponível em: https://git-scm.com/downloads

D) **Repositório alvo**. No caso desta pesquisa, foi utilizado o repositório do KDELibs, disponível em: https://github.com/KDE/kdelibs

A após a aquisição do repositório-alvo (no caso deste estudo o repositório do KDELibs), é necessária a extração do *log* de mensagens de commit. Para isso, pode ser utilizado tanto o terminal de comandos, com o comando `git log` dentro da pasta principal do repositório, quanto utilizar as funcionalidades do cliente do Git disponível, através da função exportar *log*. É importante que o formato de exportação seja em arquivo de texto plano (.txt), desprovido, portanto, de qualquer tipo de formatação. Esse arquivo será a fonte de entrada de dados para a ferramenta ATM. A ferramenta ATM já conta com o conjunto de palavras-chave produzida neste estudo, porém ela aceita tanto a inclusão, quanto a exclusão das palavras-chave de acordo com a preferência do usuário. A Figura A.1 apresenta a tela de configuração das palavras-chave.

Após a mineração dos dados, a ferramenta ATM apresentará uma lista de mensagens de *commit* candidatas a conterem traços arquiteturais, o usuário poderá confirmá-las

ou rejeitá-las. Ao final desse processo, é possível extrair as mensagens confirmadas para um arquivo em formato de texto separado por vírgulas (.csv). Nesse arquivo constam as seguintes informações referentes à mineração: chave de identificação do *commit*, mensagem de *commit*, as palavras-chave encontradas no corpo da mensagem e a avaliação realizada pelo usuário. A Figura A.2 apresenta a tela de resultados da mineração realizada pela ferramenta ATM com o *log* do projeto Finagle[1].

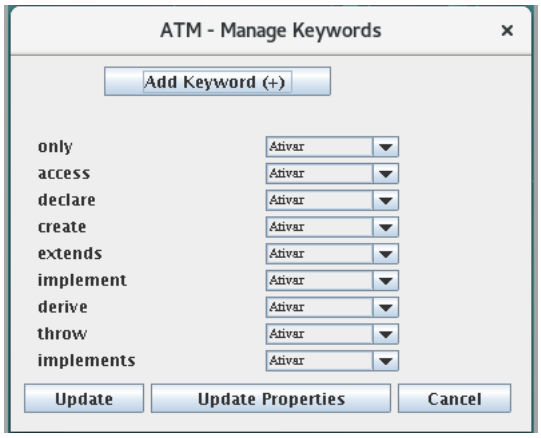


Figure A.1: Tela de configuração de palavras-chave da ferramenta ATM.

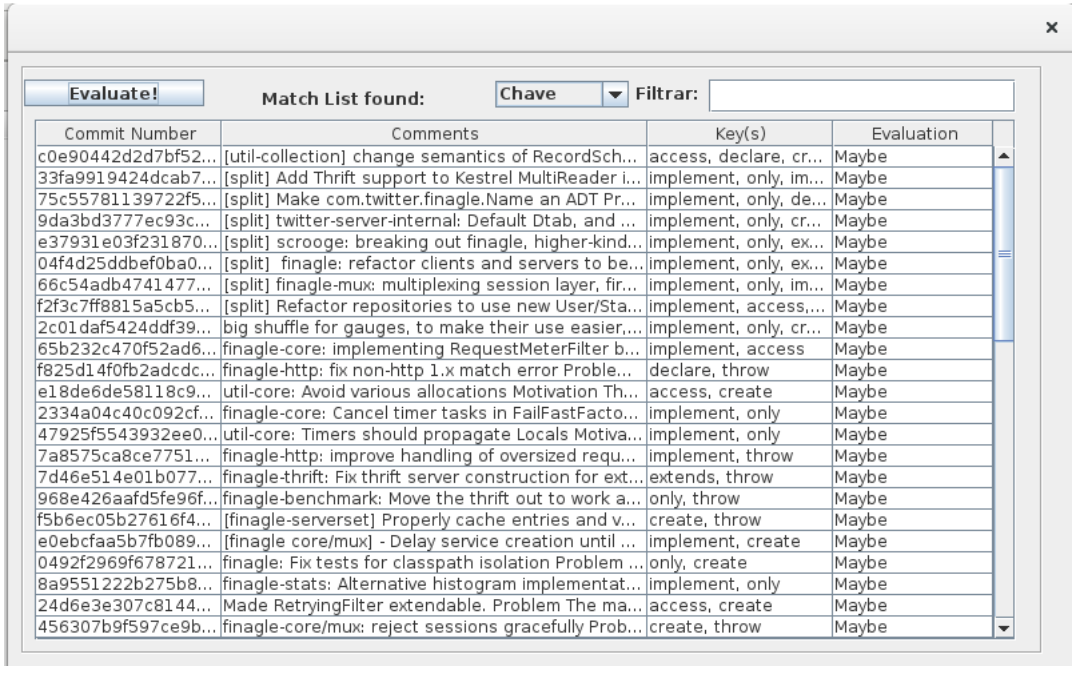


Figure A.2: Tela de resultados da mineração da ferramenta ATM a partir do *log* do projeto Finagle.

[1]Disponível em: https://github.com/twitter/finagle

Appendix

B

MÉTODOS DE RECUPERAÇÃO ARQUITETURAL

A Tabela B.1 apresenta um conjunto de técnicas de recuperação arquitetural estática disponíveis. Além do nome, ela relaciona os autores e o artigo onde a mesma foi apresentada. Trata-se de uma adaptação do levantamento apresentado por Clements (1996). Nessa ocasião, foi apresentada uma taxonomia das linguagens de descrição arquitetural disponíveis e em desenvolvimento à época.

Table B.1: Conjunto de técnicas de recuperação arquitetural estática.

Técnica	**Autores**	**Trabalho de Referência**
Alborz	Sartipi (2003)	Software Architecture Recovery Based on Pattern Matching
Algorithm for Comprehension-Driven Clustering (ACDC)	Tzerpos e Holt (2000)	ACDC: an algorithm for comprehension-driven clustering
Architecture Recovery using Concerns (ARC)	Garcia et al. (2012)	A framework for obtaining the ground-truth in architectural recovery
Archview	Pinzger (2005)	ArchView—Analyzing Evolutionary Aspects of Complex Software Systems
Archvis	Hatch (2004)	Software Architecture Visualisation
ARES	Eixelsberger et al. (1998)	Software architecture recovery of a program family
ARM	Guo, Atlee e Kazman (1999)	A software architecture reconstruction method
ARMIN	Kazman, O'Brien e Verhoef (2003)	Architecture Reconstruction Guidelines
ART	Fiutem et al. (1999)	Art: an architectural reverse engineering environment
Bauhaus	O'Brien, Smith e Lewis (2005)	Supporting migration to services using software architecture reconstruction
Bunch	Mancoridis et al. (1999)	Bunch: A clustering tool for the recovery and maintenance of software system structures
Cacophony	Favre (2004)	Cacophony: Metamodel-driven software architecture reconstruction
Dali	Kazman e Carrière (1999)	Playing detective: Reconstructing software architecture from available evidence
Disco Tect	Yan et al. (2004)	Discotect: A system for discovering architectures from running systems
Focus	Ding e Medvidovic (2001)	Focus: A light-weight, incremental approach to software architecture recovery and evolution
Gupro	Riediger e Winter (2002)	GUPRO - Generic Understanding of Programs
Intensive	Mens et al. (2006)	Co-evolving code and design with intensional views: A case study
ManSART	Harris, Reubenstein e Yeh (1995)	Reverse engineering to the architectural level

Tabela B.1 – parte 2

Técnica	**Autores**	**Trabalho de Referência**
MAP	Stoermer e O'Brien (2001)	MAP-mining architectures for product line evaluations
PBS/SBS	Holt (1998)	Structural manipulations of software architecture using Tarski relational algebra
PuLSE/SAVE	Knodel et al. (2006)	Static evaluation of software architectures
QADSAR	Stoermer, O'Brien e Verhoef (2003); Stoermer et al. (2006)	Moving towards quality attribute driven software architecture reconstruction; Model-centric software architecture reconstruction
Revealer	Pinzger et al. (2002)	Revealer: A lexical pattern matcher for architecture recovery
RMTool	Murphy (1996)	Lightweight structural summarization as an aid to software evolution
SARTool	Krikhaar (1999)	Software Architecture Reconstruction
SAVE	Miodonski et al. (2004)	Evaluation of Software Architectures with Eclipse
scaLable InforMation BOttleneck (LIMBO)	Maqbool e Babri (2007)	Hierarchical clustering for software architecture recovery
Softwarenaut	Lungu, Lanza e Gîrba (2006)	Package patterns for visual architecture recovery
Symphony, Nimeta	Riva (2004); Deursen et al. (2004)	View-Based Software Architecture Reconstruction; Symphony: View-driven software architecture reconstruction
URCA	Bojic e Velasevic (2000)	A use-case driven method of architecture recovery for program understanding and reuse reengineering
W4	Hassan e Holt (2004)	Using Development History Sticky Notes to Understand Software Architecture
Weighted Combined Algorithm (WCA)	Maqbool e Babri (2004)	The weighted combined algorithm: A linkage algorithm for software clustering
X-Ray	Mendonça e Kramer (2001)	An approach for recovering distributed system architectures
Zone-Based Recovery (ZBR)	Corazza, Martino e Scanniello (2010)	A probabilistic based approach towards software system clustering

Appendix C

LISTA DE PALAVRAS-CHAVES DISPONÍVEIS NO SURVEY REALIZADO PARA A DEFINIÇÃO DO CONJUNTO DE PALAVRAS-CHAVES ARQUITETURAIS

A sequência de tabelas abaixo, apresenta o conjunto de palavras-chaves apresentadas aos especialistas, organizadas considerando a origem dos termos. A Tabela C.1 mostra às palavra-chaves relacionadas aos Estilos Arquiteturais. A Tabela C.2 relaciona às palavra-chaves relacionadas às restrições arquiteturais apontadas na literatura. A Tabela C.3 apresenta as palavra-chaves relacionadas às Linguagens de Descrição Arquitetural escolhidas para esse estudo. Já a Tabela C.4 traz à tona as palavra-chaves relacionadas aos requisitos não-funcionais. Por fim, a Tabela C.5 apresenta as palavra-chaves não-relacionadas aos tópicos anteriores.

Table C.1: Conjunto de termos relacionados à Estilos Arquitetuarais.

Referência	**Termos**
Clements et al. (2003)	client-server, scalability, layering, portability, information_hiding_based, decomposition, services, modifiability, interoperability, deployment view, pipe-and-filter, shared-data, component, connector, module, allocation, runtime behavior, invocation, peer, event-driven, responsability, is part of, depends on, decomposition, visibility, allowed-to-use, relationship, dependability, security, channel, subscriptions, publish-subscribe, allocated-to, migrates-to, copy-migrates-to, execution-migrates-to, resource_consumption, containment, performance
Buschmann, Henney e Schimdt (2007)	exchangeability, unnecessary work, structural decomposition, whole-part, composite, completeness, container-contents, Facade, Iterator, interdependency, microkernel, coupling and cohesion, module, component, connector, relationship, view, framework, abstraction, encapsulation, information hiding, modularization, separation of concerns, sufficency, primitiveness, conceptual, collection-members

Table C.2: Conjunto de termos relacionados à Restrições Arquiteturais.

Termos
rationale, obsolesced, can access, can not access, declare, derive, only access

Table C.3: Conjunto de termos oriundos da Linguagens de Descrição Arquitetural adotadas nesse estudo.

Linguagem	**Taxonomia**	**Termos**
AADL	*Components*	thread, thread group, process, data, subprogram, processor, memory, device, bus, system
	Implementation	extends, refines type, subcomponent, call, connection, flow, mode, propertie, feature
	packages, property sets, annexes	package, category, event
	Others topics	control_laws, static_data, control_group, control_data, error_data, error_detection, provides data access, flow path, requires data access, requires bus access, provide bus access, flow source, flow sink
	Reserved words	aadlboolean, aadlinteger, aadlreal, aadlstring, access, all, and, annex, applies, binding, bus, calls, classifier, connections, constant, data, delta, device, end, enumeration, event, extends, false, features, flow, flows, group, implementation, In, inherit, initial, inverse, Is, list, memory, mode, modes, none, not, of, or, out, package, parameter, path, port, private, process, processor, properties, property, provides, public, range, reference, refined, refines, requires, server, set, sink, source, subcomponents, subprogram, system, thread, to, true, type, units, value
ACME	*Structural*	components, ports, conectors, roles, systems, representations, properties, types, requirements, constraints, attachments
	Types	type, maintain, derive, enforce, property, extends, subtype, set
	Styles	style, with, define
	Predicate	component, connector, port, role, system, property, representation, type, union, intersection, member of, size

Tabela C.3 – parte 2

Linguagem	Taxonomia	Termos
Darwin	*Components*	interface, types, semantics, constraints, evolution
	Connectors	interface, types, semantics, constraints, evolution
	Architectural Configurations	Understandability, Compositionality, Heterogeneity, Constraints, Refinement and traceability, Scalability, Evolution, Dynamism
	Tool support	active specification, multiple views, analysis, refinement, code generation, dynamism
	Keywords	assert, bind, component, dyn, export, forall, import, inst, interface, portal, provide, require, spec, to, when, int, double, string, boolean, true, false
	Portal	portal, require, provide, import, export
	Binding	peer, outward, import, inward, export, switch, dyn component, dyn instance
Rapid	*Events*	pattern, binary_pattern, performer_part, basic_pattern, guard, action_part
	Interfaces	declarations, interface, provides, requires, type, in action, out action, service, trigger, behavior, constraint, action_declaration, trasition_body service_declaration, state_trasition_rule
	Architecture	architecture, connection, constraints, declarations, basic_function_pattern, basic_pattern_list, component_generation
	Sintax	type_expression, interface_type_expression, function_type_expression, type_constructor_expression, type_declaration, interface_type_expression, interface_derivation_declaration, interface_declarative_item, interface_declarative_region, interface_provides_part, interface_requires_part, interface_action_part, provides_part, requires_part, action_part, action_declaration, in, out, name_declaration, type_name_declaration, type_constructor_name_declaration, object_name_declaration, identifier_list

Tabela C.3 – parte 3		
Linguagem	**Taxonomia**	**Termos**
Rapid	*Sintax*	interface_derivation_declaration, include_modifier, replacement_list, function_type_expression, formal_parameter_declaration_list, formal_parameter_declaration, formal_type_parameter_declaration, formal_object_parameter_declaration, type_constructor_declaration, type_constructor_expression, type_constructor_application, actual_parameter_list, actual_parameter

Table C.4: Conjunto de termos oriundos de Requisitos Não-Funcionais.

Referência	**Taxonomia**	**Termos**
Standardization and Comission (2001)	*Functionality*	Adequacy, Accuracy, Interoperability, Security, Functionality Compliance
	Reliability	Maturity, Fault Tolerance, Recoverability, Reliability Compliance
	Usability	Understandability, Learnability, Operationality, Attractiveness, Usability compliance
	Efficiency	Time behavior, Resource utilization, Efficiency compliance
	Maintainability	Analyzability, Changeability, Stability, Testability, Maintenanability compliance
	Portability	Adaptability, Installability, Co-existence, Replaceability, Portability compliance
Gazi, Umar e Sadiq (20015)	*Functionality*	feature set, capabilities, generality, Access Control, Confidentiality, Availability, Authentication, Integrity
	Usability	human factors, aesthetics, consistency, documentation, Learnability, Operability, Ease of Use, Usefulness, Productivity
	Reliability	frequency/severity of failure, recoverability, predictability, accuracy, means time to failure, Completeness, Accuracy, Maturity, Compliance
	Performance	speed efficiency, resource consumption, Space, throughput, response time, capacity, Latency
	Supportability	testability, extensibility, adaptability, maintainability, compatibility, configurability, installability, localizability, portability

Tabela C.4 – parte 2	
Referência	**Termos**
Balushi et al. (2007)	unambiguous, complete, verifiable, consistent, modifiable, traceable, and usable during operations and maintenance; Efficiency, resource_utilization, time_behavior, functionality, maintainability, portability, reliability, usability
Lamsweerde (2004)	Early deployment, incrementality, reasoning about alternatives, high assurance, security-by-construction, separation of concerns
Chung et al. (2000)	understandability, usability, modifiability, interoperability, reliability, portability, maintainability, scalability, ubiquity, configurability, customizability, adaptability, variability, volatility, traceability, security, simplicity, clarity, accuracy, integrity, modularity, nomadicity, user-friendliness, cost, robustness, timeliness, responsiveness, correctness, completeness, conciseness, cohesiveness, performance, efficiency, precision, development time, low coupling
Freitas et al. (2007)	deadline, period, cost, release time, activation latency, start and end, jitter, tolerant delay, laxity, freshness, resolution, resolution, drift, response time, throughtput, task allocation, hosts, communication, synchronization, area, power consumption, total energy, memory
Sutcliffe e Minocha (1998)	understandability, usability, modifiability, interoperability, reliability, portability, maintainability, scalability, cost, configurability, customizability, adaptability, variability, volatility, traceability, security, simplicity, nomadicity, clarity, ubiquity, integrity, modularity, development time, user-friendliness, robustness, timeliness, responsiveness, correctness, completeness, conciseness, cohesiveness, performance, efficiency, accuracy, precision, low coupling
Ebert (1997)	performance, reliability, availability, failure tolerance, usability, correctness, extendibility, maintainability, readability, reusability, fault tolerance, completely fulfilled customers, no module split
Kaur e Verma (2015)	Security, performance, availability, visibility, recoverability, confidentiality, reliability, operability, traceability, usability

Table C.5: Conjunto de termos arquiteturais não-classificados nos tópicos anteriores.

Termos
reengineering, architectural style, complexity, constraint, Application Protocol Interface

Appendix

D

SCRIPTS UTILIZADOS NO ESTUDO DE COLETA E AVALIAÇÃO DE MÉTRICAS DE CÓDIGO DO PROJETO KDELIBS

Este apêndice traz os scripts construídos visando a automatização da execução do Analizo em diferentes versões do Projeto KDELibs, assim como o script de geração do banco de dados utilizado para armazenar os dados obtidos e os scripts utilizados para processar os dados no pacote estatístico R.

D.1 SCRIPT DE AUTOMATIZAÇÃO DO ANALIZO

ExecutaAnalizo.sh: Script de automatização da execuçã o do Analizo para a extração de métricas do projeto KDELibs (parte 1).

```
1  #!/bin/bash
2
3  DEZENA=$(echo ``10")
4  CENTENA=$(echo ``100")
5  RESULT=$(echo ``1")
6
7  while read linha
8  do
9
10 tar -xzvf kdelibsOriginals.tar.gz;
11 cd kdelibs;
12 git clean -fd;
13 git checkout $linha > saida.out;
14 sleep 30s;
15 cd ..;
16 pwd;
17 if [ $RESULT -lt $DEZENA ];
18 then
```

ExecutaAnalizo.sh: Script de automatização da execução do Analizo para a extração de métricas do projeto KDELibs (parte 2).

```
19 cp -R ``kdelibs" ``/home/motta/thesis/Git/kdeLibsVersions/00$RESULT"
20 echo ``$RESULT"
21 elif [ $RESULT -lt $CENTENA ];
22 then
23 cp -R ``kdelibs" ``/home/motta/thesis/Git/kdeLibsVersions/0$RESULT"
24 echo "$RESULT"
25 else
26 cp -R ``kdelibs" ``/home/motta/thesis/Git/kdeLibsVersions/$RESULT"
27 echo ``$RESULT"
28 fi
29
30 RESULT=`expr $RESULT + 1`
31
32 done < keysGitAfterKdelibs.txt
33
34 RESULT=$(echo ``1")
35 while read linha
36 do
37 tar -xzvf kdelibsOriginals.tar.gz;
38 cd kdelibs;
39 git clean -fd;
40 git checkout $linha > saida.out;
41 sleep 30s;
42 cd ..;
43
44 if [ $RESULT -lt $DEZENA ];
45 then
46 cp -R ``kdelibs" ``/home/motta/thesis/Git/kdeLibsVersions/00$RESULT" ``_"
47 echo ``$RESULT"
48 elif [ $RESULT -lt $CENTENA ];
49 then
50 cp -R ``kdelibs" ``/home/motta/thesis/Git/kdeLibsVersions/0$RESULT" ``_"
51 echo ``$RESULT"
52 else
53 cp -R ``kdelibs" ``/home/motta/thesis/Git/kdeLibsVersions/$RESULT" ``_"
54 echo ``$RESULT"
55 fi
56
57 RESULT=`expr $RESULT + 1`
58
59 done < keysGitBeforeKdelibs.txt
```

D.2 SCRIPT DE GERAÇÃO DO BANCO DE DADOS METRICS

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 1).

```
1  SET statement_timeout = 0;
2  SET lock_timeout = 0;
3  SET idle_in_transaction_session_timeout = 0;
4  SET client_encoding = 'UTF8';
5  SET standard_conforming_strings = on;
6  SELECT pg_catalog.set_config('search_path', '', false);
7  SET check_function_bodies = false;
8  SET xmloption = content;
9  SET client_min_messages = warning;
10 SET row_security = off;
11
12 CREATE EXTENSION IF NOT EXISTS plpgsql WITH SCHEMA pg_catalog;
13
14 COMMENT ON EXTENSION plpgsql IS 'PL/pgSQL procedural language';
15
16 CREATE FUNCTION public.update_length_trigger() RETURNS trigger
17     LANGUAGE plpgsql
18     AS $$
19 BEGIN
20 -- Verificar se foi fornecido o tamanho da mensagem de commit
21 IF NEW.message IS NULL THEN
22 RAISE EXCEPTION 'The commit message can not be NULL';
23 END IF;
24 -- Registrar o tamanho da mensagem de commit
25 NEW.message_length := length(NEW.message);
26 RETURN NEW;
27 END;
28 $$;
29 ALTER FUNCTION public.update_length_trigger() OWNER TO motta;
30
31 SET default_tablespace = '';
32 SET default_with_oids = false;
33
34 CREATE TABLE public.architectural_topic (
35     id integer NOT NULL,
36     name character varying(80),
37     description character varying(300)
38 );
39
40 ALTER TABLE public.architectural_topic OWNER TO motta;
41 CREATE SEQUENCE public.architectural_topic_id_seq
42     AS integer
43     START WITH 1
44     INCREMENT BY 1
45     NO MINVALUE
46     NO MAXVALUE
47     CACHE 1;
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 2).

```
48 ALTER TABLE public.architectural_topic_id_seq OWNER TO motta;
49 ALTER SEQUENCE public.architectural_topic_id_seq
50         OWNED BY public.architectural_topic.id;
51
52 CREATE TABLE public.commit (
53     id_commit integer NOT NULL,
54     key character varying(42),
55     author character varying(200),
56     message character varying(65540),
57     is_architectural boolean,
58     components_changed_amount smallint,
59     message_length integer,
60     date_register timestamp without time zone,
61     id_analizo character varying(4),
62     commit_position character varying(6),
63     commit_type character varying(17)
64 );
65
66 ALTER TABLE public.commit OWNER TO motta;
67 CREATE TABLE public.commit_architectural_topic (
68     id_commit_commit integer NOT NULL,
69     id_architectural_topic integer NOT NULL
70 );
71
72 ALTER TABLE public.commit_architectural_topic OWNER TO motta;
73 CREATE TABLE public.commit_file_metric (
74     id_commit_commit integer NOT NULL,
75     id_metric integer NOT NULL,
76     id_file integer NOT NULL,
77     value double precision,
78     module character varying(600) NOT NULL
79 );
80
81 ALTER TABLE public.commit_file_metric OWNER TO motta;
82 CREATE TABLE public.commit_file_metric_cmt (
83     id_commit integer NOT NULL,
84     id_metric integer NOT NULL,
85     id_file integer NOT NULL,
86     value double precision,
87     object_ character varying(6000) NOT NULL
88 );
89
90 ALTER TABLE public.commit_file_metric_cmt OWNER TO motta;
91 CREATE TABLE public.commit_file_metric_originals (
92     id_commit_commit integer NOT NULL,
93     id_metric integer NOT NULL,
94     id_file integer NOT NULL,
95     value double precision,
96     module character varying(600) NOT NULL
97 );
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 3).

```
98  ALTER TABLE public.commit_file_metric_originals OWNER TO motta;
99  CREATE TABLE public.commit_general_topic (
100     id_commit_commit integer NOT NULL,
101     id_general_topic integer NOT NULL
102 );
103
104 ALTER TABLE public.commit_general_topic OWNER TO motta;
105 CREATE SEQUENCE public.commit_id_commit_seq
106     AS integer
107     START WITH 1
108     INCREMENT BY 1
109     NO MINVALUE
110     NO MAXVALUE
111     CACHE 1;
112
113 ALTER TABLE public.commit_id_commit_seq OWNER TO motta;
114 ALTER SEQUENCE public.commit_id_commit_seq OWNED BY public.commit.id_commit;
115 CREATE TABLE public.commit_keyword (
116     id_commit integer NOT NULL,
117     id_keyword integer NOT NULL
118 );
119
120 ALTER TABLE public.commit_keyword OWNER TO motta;
121 CREATE TABLE public.commit_metric_summary (
122     id_commit integer NOT NULL,
123     id_metric integer NOT NULL,
124     id_summary integer NOT NULL,
125     value double precision
126 );
127
128 ALTER TABLE public.commit_metric_summary OWNER TO motta;
129 CREATE TABLE public.commit_totals (
130     id_commit integer NOT NULL,
131     id_totals integer NOT NULL,
132     value double precision
133 );
134
135 ALTER TABLE public.commit_totals OWNER TO motta;
136 CREATE TABLE public.file (
137     id integer NOT NULL,
138     name character varying(300)
139 );
140
141 ALTER TABLE public.file OWNER TO motta;
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 4).

```
142 CREATE SEQUENCE public.file_id_seq
143     AS integer
144     START WITH 1
145     INCREMENT BY 1
146     NO MINVALUE
147     NO MAXVALUE
148     CACHE 1;
149
150 ALTER TABLE public.file_id_seq OWNER TO motta;
151 ALTER SEQUENCE public.file_id_seq OWNED BY public.file.id;
152 CREATE TABLE public.file_modified (
153     id_commit_commit integer NOT NULL,
154     id_operation integer NOT NULL,
155     id_file integer NOT NULL
156 );
157
158 ALTER TABLE public.file_modified OWNER TO motta;
159 CREATE VIEW public.files_deleted_by_commits AS
160  SELECT file_modified.id_commit_commit,
161     file_modified.id_file
162    FROM public.file_modified
163   WHERE ((file_modified.id_commit_commit < 465) AND
164          (file_modified.id_operation = 5))
165   ORDER BY file_modified.id_commit_commit, file_modified.id_file;
166 ALTER TABLE public.files_deleted_by_commits OWNER TO motta;
167
168 CREATE VIEW public.files_renamed_by_commits AS
169  SELECT file_modified.id_commit_commit,
170     file_modified.id_file
171    FROM public.file_modified
172   WHERE ((file_modified.id_commit_commit < 465) AND
173          (file_modified.id_operation = 3))
174   ORDER BY file_modified.id_commit_commit, file_modified.id_file;
175
176 ALTER TABLE public.files_renamed_by_commits OWNER TO motta;
177 CREATE TABLE public.general_topic (
178     id integer NOT NULL,
179     name character(80),
180     description character varying(300)
181 );
182
183 ALTER TABLE public.general_topic OWNER TO motta;
184 CREATE SEQUENCE public.general_topic_id_seq
185     AS integer
186     START WITH 1
187     INCREMENT BY 1
188     NO MINVALUE
189     NO MAXVALUE
190     CACHE 1;
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 5).

```
191 ALTER TABLE public.general_topic_id_seq OWNER TO motta;
192 ALTER SEQUENCE public.general_topic_id_seq OWNED BY public.general_topic.id;
193 CREATE TABLE public.keyword (
194     id integer NOT NULL,
195     name character varying(60)
196 );
197
198 ALTER TABLE public.keyword OWNER TO motta;
199 CREATE SEQUENCE public.keyword_id_seq
200     AS integer
201     START WITH 1
202     INCREMENT BY 1
203     NO MINVALUE
204     NO MAXVALUE
205     CACHE 1;
206
207 ALTER TABLE public.keyword_id_seq OWNER TO motta;
208 ALTER SEQUENCE public.keyword_id_seq OWNED BY public.keyword.id;
209 CREATE TABLE public.list_of_module_empty (
210     id_commit integer
211 );
212
213 ALTER TABLE public.list_of_module_empty OWNER TO motta;
214 CREATE TABLE public.metric (
215     id integer NOT NULL,
216     name character varying(80),
217     abbreviation character varying(7)
218 );
219
220 ALTER TABLE public.metric OWNER TO motta;
221 CREATE SEQUENCE public.metric_id_seq
222     AS integer
223     START WITH 1
224     INCREMENT BY 1
225     NO MINVALUE
226     NO MAXVALUE
227     CACHE 1;
228
229 ALTER TABLE public.metric_id_seq OWNER TO motta;
230 ALTER SEQUENCE public.metric_id_seq OWNED BY public.metric.id;
231 CREATE TABLE public.operation (
232     id integer NOT NULL,
233     abbreviation character(1),
234     description character varying(20)
235 );
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 6).

```
236
237 ALTER TABLE public.operation OWNER TO motta;
238 CREATE SEQUENCE public.operation_id_seq
239     AS integer
240     START WITH 1
241     INCREMENT BY 1
242     NO MINVALUE
243     NO MAXVALUE
244     CACHE 1;
245
246 ALTER TABLE public.operation_id_seq OWNER TO motta;
247 ALTER SEQUENCE public.operation_id_seq OWNED BY public.operation.id;
248 CREATE TABLE public.releases (
249     id integer NOT NULL,
250     name character varying(80),
251     data date
252 );
253
254 ALTER TABLE public.releases OWNER TO motta;
255 CREATE SEQUENCE public.releases_id_seq
256     AS integer
257     START WITH 1
258     INCREMENT BY 1
259     NO MINVALUE
260     NO MAXVALUE
261     CACHE 1;
262
263 ALTER TABLE public.releases_id_seq OWNER TO motta;
264 ALTER SEQUENCE public.releases_id_seq OWNED BY public.releases.id;
265 CREATE TABLE public.report_participation (
266     id integer NOT NULL,
267     time_involved character(13),
268     amount integer,
269     name character varying(150)
270 );
271
272 ALTER TABLE public.report_participation OWNER TO motta;
273 CREATE SEQUENCE public.report_participation_id_seq
274     AS integer
275     START WITH 1
276     INCREMENT BY 1
277     NO MINVALUE
278     NO MAXVALUE
279     CACHE 1;
280
281 ALTER TABLE public.report_participation_id_seq OWNER TO motta;
282 ALTER SEQUENCE public.report_participation_id_seq
283         OWNED BY public.report_participation.id;
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 7).

```
284 CREATE TABLE public.summary_metric (
285     id_summary integer NOT NULL,
286     name character varying(20)
287 );
288
289 ALTER TABLE public.summary_metric OWNER TO motta;
290 CREATE SEQUENCE public.summary_metric_id_summary_seq
291     AS integer
292     START WITH 1
293     INCREMENT BY 1
294     NO MINVALUE
295     NO MAXVALUE
296     CACHE 1;
297
298 ALTER TABLE public.summary_metric_id_summary_seq OWNER TO motta;
299 ALTER SEQUENCE public.summary_metric_id_summary_seq
300         OWNED BY public.summary_metric.id_summary;
301 CREATE TABLE public.totals (
302     id integer NOT NULL,
303     name character varying(50)
304 );
305
306 ALTER TABLE public.totals OWNER TO motta;
307 CREATE SEQUENCE public.totals_id_seq
308     AS integer
309     START WITH 1
310     INCREMENT BY 1
311     NO MINVALUE
312     NO MAXVALUE
313     CACHE 1;
314
315 ALTER TABLE public.totals_id_seq OWNER TO motta;
316
317 ALTER SEQUENCE public.totals_id_seq OWNED BY public.totals.id;
318
319 ALTER TABLE ONLY public.architectural_topic ALTER COLUMN id SET DEFAULT
320  nextval('public.architectural_topic_id_seq'::regclass);
321
322 ALTER TABLE ONLY public.commit ALTER COLUMN id_commit SET DEFAULT
323         nextval('public.commit_id_commit_seq'::regclass);
324
325 ALTER TABLE ONLY public.file ALTER COLUMN id SET DEFAULT
326         nextval('public.file_id_seq'::regclass);
327
328 ALTER TABLE ONLY public.general_topic ALTER COLUMN id SET DEFAULT
329         nextval('public.general_topic_id_seq'::regclass);
330
331 ALTER TABLE ONLY public.keyword ALTER COLUMN id SET DEFAULT
332          nextval('public.keyword_id_seq'::regclass);
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 8).

```
333 ALTER TABLE ONLY public.metric ALTER COLUMN id SET DEFAULT
334           nextval('public.metric_id_seq'::regclass);
335
336 ALTER TABLE ONLY public.operation ALTER COLUMN id SET DEFAULT
337          nextval('public.operation_id_seq'::regclass);
338
339 ALTER TABLE ONLY public.releases ALTER COLUMN id SET DEFAULT
340           nextval('public.releases_id_seq'::regclass);
341
342 ALTER TABLE ONLY public.report_participation ALTER COLUMN id SET DEFAULT
343          nextval('public.report_participation_id_seq'::regclass);
344
345 ALTER TABLE ONLY public.summary_metric ALTER COLUMN id_summary SET DEFAULT
346           nextval('public.summary_metric_id_summary_seq'::regclass);
347
348 ALTER TABLE ONLY public.totals ALTER COLUMN id SET DEFAULT
349           nextval('public.totals_id_seq'::regclass);
350
351 ALTER TABLE ONLY public.architectural_topic
352     ADD CONSTRAINT architectural_topic_pk PRIMARY KEY (id);
353
354 ALTER TABLE ONLY public.commit
355     ADD CONSTRAINT commit_pk PRIMARY KEY (id_commit);
356
357 ALTER TABLE ONLY public.file
358     ADD CONSTRAINT file_pk PRIMARY KEY (id);
359
360 ALTER TABLE ONLY public.general_topic
361     ADD CONSTRAINT general_topic_pk PRIMARY KEY (id);
362
363 ALTER TABLE ONLY public.commit_metric_summary
364     ADD CONSTRAINT id_commit_summary PRIMARY KEY
365     (id_commit, id_metric, id_summary);
366
367 ALTER TABLE ONLY public.commit_totals
368     ADD CONSTRAINT id_commit_totals PRIMARY KEY (id_commit, id_totals);
369
370 ALTER TABLE ONLY public.summary_metric
371     ADD CONSTRAINT id_summary PRIMARY KEY (id_summary);
372
373 ALTER TABLE ONLY public.totals
374     ADD CONSTRAINT id_totals PRIMARY KEY (id);
375
376 ALTER TABLE ONLY public.keyword
377     ADD CONSTRAINT key_pk PRIMARY KEY (id);
378
379 ALTER TABLE ONLY public.metric
380     ADD CONSTRAINT metric_pk PRIMARY KEY (id);
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 9).

```
381 ALTER TABLE ONLY public.operation
382     ADD CONSTRAINT operation_pk PRIMARY KEY (id);
383
384 ALTER TABLE ONLY public.commit_architectural_topic
385     ADD CONSTRAINT pk_commit_at PRIMARY KEY
386     (id_commit_commit, id_architectural_topic);
387
388 ALTER TABLE ONLY public.commit_general_topic
389     ADD CONSTRAINT pk_commit_general_topic PRIMARY KEY
390       (id_commit_commit, id_general_topic);
391
392 ALTER TABLE ONLY public.commit_keyword
393     ADD CONSTRAINT pk_commit_k PRIMARY KEY (id_commit, id_keyword);
394
395 ALTER TABLE ONLY public.file_modified
396     ADD CONSTRAINT pk_file_modified PRIMARY KEY
397       (id_commit_commit, id_operation, id_file);
398
399 ALTER TABLE ONLY public.commit_file_metric
400     ADD CONSTRAINT pk_fm_commit PRIMARY KEY
401       (id_commit_commit, id_metric, id_file, module);
402
403 ALTER TABLE ONLY public.commit_file_metric_cmt
404     ADD CONSTRAINT pk_fm_commit_cmt PRIMARY KEY
405     (id_commit, id_metric, id_file, object_);
406
407 ALTER TABLE ONLY public.commit_file_metric_originals
408     ADD CONSTRAINT pk_fm_commit_originals PRIMARY KEY
409       (id_commit_commit, id_metric, id_file, module);
410
411 ALTER TABLE ONLY public.releases
412     ADD CONSTRAINT releases_pk PRIMARY KEY (id);
413
414 ALTER TABLE ONLY public.report_participation
415     ADD CONSTRAINT report_participation_pk PRIMARY KEY (id);
416
417 ALTER TABLE ONLY public.keyword
418     ADD CONSTRAINT un_key UNIQUE (name);
419
420 ALTER TABLE ONLY public.file
421     ADD CONSTRAINT un_name UNIQUE (name);
422
423 CREATE INDEX idx_tbl_cfm ON public.commit_file_metric
424         USING btree (id_commit_commit, id_metric, id_file);
425
426 CREATE INDEX idx_tbl_commit ON public.commit_file_metric
427         USING btree (id_commit_commit);
428
429 CREATE INDEX idx_tbl_commit_metric ON public.commit_file_metric USING
430         btree (id_commit_commit, id_metric);
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 10).

```
431 CREATE INDEX idx_tbl_file ON public.commit_file_metric USING btree
432 (id_file);
433
434 CREATE INDEX idx_tbl_metric ON public.commit_file_metric USING btree
435 (id_metric);
436
437 CREATE TRIGGER update_length_trigger BEFORE INSERT OR UPDATE ON
438 public.commit FOR EACH ROW EXECUTE PROCEDURE
439  public.update_length_trigger();
440
441 ALTER TABLE ONLY public.commit_architectural_topic
442     ADD CONSTRAINT architectural_topic_fk FOREIGN KEY
443       (id_architectural_topic) REFERENCES public.architectural_topic(id)
444       MATCH FULL ON UPDATE CASCADE ON DELETE SET NULL;
445
446 ALTER TABLE ONLY public.commit_architectural_topic
447     ADD CONSTRAINT commit_fk FOREIGN KEY
448     (id_commit_commit) REFERENCES public.commit(id_commit)
449      MATCH FULL ON UPDATE CASCADE ON DELETE SET NULL;
450
451 ALTER TABLE ONLY public.commit_file_metric
452     ADD CONSTRAINT commit_fk FOREIGN KEY
453     (id_commit_commit) REFERENCES public.commit(id_commit)
454      MATCH FULL ON UPDATE CASCADE ON DELETE SET NULL;
455
456 ALTER TABLE ONLY public.commit_general_topic
457     ADD CONSTRAINT commit_fk FOREIGN KEY
458     (id_commit_commit) REFERENCES public.commit(id_commit)
459      MATCH FULL ON UPDATE CASCADE ON DELETE SET NULL;
460
461 ALTER TABLE ONLY public.file_modified
462     ADD CONSTRAINT commit_fk FOREIGN KEY
463     (id_commit_commit) REFERENCES public.commit(id_commit)
464      MATCH FULL ON UPDATE CASCADE ON DELETE SET NULL;
465
466 ALTER TABLE ONLY public.commit_file_metric_cmt
467     ADD CONSTRAINT commit_fk_2 FOREIGN KEY
468     (id_commit) REFERENCES public.commit(id_commit)
469     MATCH FULL ON UPDATE CASCADE ON DELETE SET NULL;
470
471 ALTER TABLE ONLY public.commit_keyword
472     ADD CONSTRAINT commit_key_fk FOREIGN KEY
473     (id_commit) REFERENCES public.commit(id_commit)
474     MATCH FULL ON UPDATE CASCADE ON DELETE SET NULL;
475
476 ALTER TABLE ONLY public.commit_file_metric_originals
477     ADD CONSTRAINT commit_originals_fk FOREIGN KEY
478       (id_commit_commit) REFERENCES public.commit(id_commit)
479        MATCH FULL ON UPDATE CASCADE ON DELETE SET NULL;
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 11).

```
480 ALTER TABLE ONLY public.commit_file_metric
481     ADD CONSTRAINT file_fk FOREIGN KEY (id_file)
482     REFERENCES public.file(id) MATCH FULL
483     ON UPDATE CASCADE ON DELETE SET NULL;
484
485 ALTER TABLE ONLY public.file_modified
486     ADD CONSTRAINT file_fk FOREIGN KEY (id_file)
487     REFERENCES public.file(id) MATCH FULL
488     ON UPDATE CASCADE ON DELETE SET NULL;
489
490 ALTER TABLE ONLY public.commit_file_metric_cmt
491     ADD CONSTRAINT file_fk_2 FOREIGN KEY
492     (id_file) REFERENCES public.file(id) MATCH FULL
493     ON UPDATE CASCADE ON DELETE SET NULL;
494
495 ALTER TABLE ONLY public.commit_file_metric_originals
496     ADD CONSTRAINT file_originals_fk FOREIGN KEY
497     (id_file) REFERENCES public.file(id) MATCH FULL
498     ON UPDATE CASCADE ON DELETE SET NULL;
499
500 ALTER TABLE ONLY public.commit_metric_summary ADD CONSTRAINT fk_commit
501 FOREIGN KEY (id_commit) REFERENCES public.commit(id_commit);
502
503 ALTER TABLE ONLY public.commit_totals ADD CONSTRAINT fk_commit
504 FOREIGN KEY (id_commit) REFERENCES public.commit(id_commit);
505
506 ALTER TABLE ONLY public.commit_metric_summary ADD CONSTRAINT fk_metric
507 FOREIGN KEY (id_metric) REFERENCES public.metric(id);
508
509 ALTER TABLE ONLY public.commit_metric_summary ADD CONSTRAINT fk_summary
510 FOREIGN KEY (id_summary) REFERENCES public.summary_metric(id_summary);
511
512 ALTER TABLE ONLY public.commit_totals ADD CONSTRAINT fk_totals
513 FOREIGN KEY (id_totals) REFERENCES public.metric(id);
514
515 ALTER TABLE ONLY public.commit_general_topic ADD CONSTRAINT
516   general_topic_fk FOREIGN KEY (id_general_topic) REFERENCES
517   public.general_topic(id) MATCH FULL ON UPDATE CASCADE
518  ON DELETE SET NULL;
519
520 ALTER TABLE ONLY public.commit_keyword ADD CONSTRAINT keyword_fk
521 FOREIGN KEY (id_keyword) REFERENCES public.keyword(id) MATCH FULL
522 ON UPDATE CASCADE ON DELETE SET NULL;
523
524 ALTER TABLE ONLY public.commit_file_metric ADD CONSTRAINT metric_fk
525 FOREIGN KEY (id_metric) REFERENCES public.metric(id) MATCH FULL
526 ON UPDATE CASCADE ON DELETE SET NULL;
```

CreateDatabaseMetrics.sql: Script de criação do banco de dados Metrics no PostgreSQL (parte 12).

```
527 ALTER TABLE ONLY public.commit_file_metric_cmt ADD CONSTRAINT metric_fk_2
528 FOREIGN KEY (id_metric) REFERENCES public.metric(id) MATCH FULL
529 ON UPDATE CASCADE ON DELETE SET NULL;
530
531 ALTER TABLE ONLY public.commit_file_metric_originals ADD CONSTRAINT
532  metric_originals_fk FOREIGN KEY (id_metric) REFERENCES public.metric(id)
533 MATCH FULL ON UPDATE CASCADE ON DELETE SET NULL;
534
535 ALTER TABLE ONLY public.file_modified ADD CONSTRAINT operation_fk
536 FOREIGN KEY (id_operation) REFERENCES public.operation(id) MATCH FULL
537 ON UPDATE CASCADE ON DELETE SET NULL;
```

D.3 SCRIPTS DE PROCESSAMENTO ESTATÍSTICO DOS DADOS EM R

Para cada métrica explorada por esse estudo foi construído um script em R individual. No entanto, os scripts são similiares entre si, a diferença entre eles é o código da métrica (*id_metric*) armazenado no banco de dados Metrics. Dessa forma, segue abaixo o script produzido em R para a métrica ACC, o qual pode ser utilizado para as demais, desde que seja atualizado o código da métrica, conforme já descrito.

acc.R: Script de análise estatística da variações da métrica ACC utilizando a linguagem R (parte 1).

```
1  library(MLmetrics, warn.conflicts = FALSE)
2  library(dplyr)
3  library(RPostgreSQL)
4
5  #Estabelecendo uma conexao
6  con <- DBI::dbConnect(RPostgreSQL::PostgreSQL(), host = "localhost",
7   db = "metrics_new")
8
9  #Carregando todos os valores para a metrica ACC
10 acc_metric_all <- dbGetQuery(con, 'select A.id_commit_commit AS id_commit,
11 B.id_commit_commit AS id_commit_before, A.value AS value_commit, B.value
12 AS value_before, A.id_file AS id_file, A.module AS module_from
13 commit_file_metric AS A join commit_file_metric AS B on
14 A.id_commit_commit = B.id_commit_commit-464 AND
15 A.id_file = B.id_file AND A.id_metric = B.id_metric AND A.id_metric=1 AND
16 A.module = B.module')
17
18 #Identificando commits arquiteturais e nao arquiteturais (commits
19 # arquiteturais tem id <=232) e calculando os deltas
20 acc_metric_all <- acc_metric_all %>%
21  mutate(arch = (id_commit <= 232), delta = value_commit - value_before)
22
23 #Selecionando os commitsArquiteturais
24 acc_metric_all_arch <- acc_metric_all %>%
25  filter(id_commit <=232)
```

acc.R: Script de análise estatística da variações da métrica ACC utilizando a linguagem R (parte 2).

```
26 #Selecionando os commits nao-arquiteturais
27 acc_metric_all_narch <- acc_metric_all %>%
28  filter(id_commit > 232)
29
30 #Teste de significancia estatistica de Wilcoxon entre os deltas obtidos
31 # por commits arquiteturais e nao-arquiteturais
32 wilcox.test(acc_metric_all_arch$delta, acc_metric_all_narch$delta)
33
34 #Obtendo o valor medio da metrica ACC por commit
35 acc_metric_mean <- tbl(con, "commit_file_metric") %>%
36  filter(id_metric == 1, value>=0) %>%
37  group_by(id_commit_commit, id_metric) %>%
38  summarise(value=mean(value, na.rm = TRUE)) %>%
39  collect()
40
41 #Coletando os arquivos modificados por commit
42 files_modified_by_commit <- tbl(con, "file_modified") %>%
43  group_by(id_commit_commit) %>%
44  summarise(quant_files=count(id_file)) %>%
45  collect()
46
47 #Identificando os commits como sendo arquiteturais ou nao, e se sao os
48 # anteriores ou nao, alem do commit que faz o par.
49 acc_metric_mean_2 <- acc_metric_mean %>%
50  mutate(arch = (id_commit_commit <= 232), before =
51  (id_commit_commit >= 465), orig_commit = if_else(id_commit_commit >= 465,
52  id_commit_commit - 464, id_commit_commit + 0.0))
53
54 #retirando os campos nao utilizados
55 acc_metric_mean_2 <- acc_metric_mean_2[,-c(2,4,5,6)]
56
57 #Separando os commits-arquiteturais e seus repectivos pares (anteriores)
58 # e calculando a variacao
59 acc_metric_mean_3 <- cbind(acc_metric_mean_2[1:232,],
60  acc_metric_mean_2[465:696,]) %>%
61  mutate(delta = value...2 - value...4)
62
63 #Separando os commits-nao-arquiteturais e seus repectivos pares
64 # (anteriores) e calculando a variacao
65 acc_metric_mean_4 <- cbind(acc_metric_mean_2[233:464,],
66  acc_metric_mean_2[697:928,]) %>%
67  mutate(delta = value...2 - value...4)
68
69 #  Secao de Mosaic plots
70 #Comparando commits com os valores de metrica obtidos para ACC observando
71 # se os valores se mantem ou variam
72 #Coletando os valores da metrica ACC para commits arquiteturais
73 dataArch <- data.frame(delta = acc_metric_mean_3, arch = "Arquitetural")
```

acc.R: Script de análise estatística da variações da métrica ACC utilizando a linguagem R (parte 3).

```
74  #Coletando os valores da metrica ACC para commits nao-arquiteturais
75  dataNonArch <- data.frame(delta = acc_metric_mean_4,
76   arch = "Nao-Arquitetural")
77  #Classificando as mudancas no dataframe auxiliar X
78  x <- bind_rows(dataArch, dataNonArch) %>%
79    mutate(sinal = abs(sign(delta.delta)))
80  #Retirando as colunas nao-utilizadas nesse dataframe
81  x <- x[,-c(1,2,3,4)]
82  #Separando os padroes (Manutencao e Variacao)
83  x[3] <- lapply(x[3], gsub, pattern = "0", replacement = "Mantem",
84   fixed = TRUE)
85  x[3] <- lapply(x[3], gsub, pattern = "1", replacement = "Varia",
86   fixed = TRUE)
87  #criando a tabela de contingencia usada no mosaic plot
88  tab <- table(x$arch, x$sinal)
89  #definindo as cores utilizadas
90  color_s <- colorspace::qualitative_hcl(3)
91  paleta1 <- c(Red = "#bc1405",
92        Yellow = "#f4c430")
93  paleta2 <- c(Blue = "#4fb2ff",
94        Green = "#1a9b1e",
95        Red = "#bc1405")
96
97  #Gerando o grafico em mosaico sem titulo
98  mosaicplot(tab, main = "", cex.axis = 1, las = 1, col = paleta1)
99  #registrando os valores em cada parte do mosaico
100 text(.33, .79, tab[1][1], col = "black", cex = 1)
101 text(.77, .65, tab[2][1], col = "black", cex = 1)
102 text(.33, .28, tab[3][1], col = "black", cex = 1)
103 text(.77, .17, tab[4][1], col = "black", cex = 1)
104
105 #Estabelecendo a significancia estatistica com o metodo Qui-Quadrado
106 chisq.test(tab)
107
108 #Comparando commits com os valores de metrica obtidos para ACC observando
109 # as variacoes (se aumentam ou diminuem)
110 #Classificando as mudancas no dataframe auxiliar X para a nova comparacao
111 # (se aumentam ou diminuem)
112 x <- bind_rows(dataArch, dataNonArch) %>%
113   mutate(sinal = sign(delta.delta)) %>% filter(sinal != 0)
114 #Retirando as colunas nao-utilizadas nesse dataframe
115 x <- x[,-c(1,2,3,4)]
116 #Separando os padroes (Manutencao e Variacao)
117 x[3] <- lapply(x[3], gsub, pattern = "-1", replacement = "Diminui",
118  fixed = TRUE)
119 x[3] <- lapply(x[3], gsub, pattern = "1", replacement = "Aumenta",
120  fixed = TRUE)
121 #criando a tabela de contingencia usada no mosaic plot
122 tab <- table(x$arch, x$sinal)
```

acc.R: Script de análise estatística da variações da métrica ACC utilizando a linguagem R (parte 4).

```
123 #Gerando o grafico em mosaico sem o titulo
124 mosaicplot(tab, main = "", cex.axis = 1, las = 1, col = paleta2)
125 #registrando os valores em cada parte do mosaico
126 text(.38, .71, tab[1][1], col = "black", cex = 1)
127 text(.82, .73, tab[2][1], col = "black", cex = 1)
128 text(.38, .18, tab[3][1], col = "black", cex = 1)
129 text(.82, .22, tab[4][1], col = "black", cex = 1)
130
131 #Estabelecendo a significancia estatistica com o metodo Qui-Quadrado
132 chisq.test(tab)
133
134 #Comparando commits com os valores de metrica obtidos para ACC observando
135 # se aumentam, diminuem ou se mantem
136 #Classificando as mudancas no dataframe auxiliar X para a comparacao dos
137 # tres grupos (se aumentam, diminuem ou se mantem)
138 x <- bind_rows(dataArch, dataNonArch) %>%
139   mutate(sinal = sign(delta.delta))
140 #Retirando as colunas nao-utilizadas nesse dataframe
141 x <- x[,-c(1,2,3,4)]
142 x[3] <- lapply(x[3], gsub, pattern = "0", replacement = "Mantem",
143  fixed = TRUE)
144 x[3] <- lapply(x[3], gsub, pattern = "-1", replacement = "Diminui",
145  fixed = TRUE)
146 x[3] <- lapply(x[3], gsub, pattern = "1", replacement = "Aumenta",
147  fixed = TRUE)
148 tab <- table(x$arch, x$sinal)
149
150 #Gerando o grafico em mosaico sem o titulo
151 mosaicplot(tab, main = "", cex.axis = 1, las = 1, col = paleta2)
152 #registrando os valores em cada parte do mosaico
153 text(.35, .82, tab[1][1], col = "black", cex = 1)
154 text(.78, .88, tab[2][1], col = "black", cex = 1)
155 text(.35, .52, tab[3][1], col = "black", cex = 1)
156 text(.78, .66, tab[4][1], col = "black", cex = 1)
157 text(.35, .2, tab[5][1], col = "black", cex = 1)
158 text(.78, .3, tab[6][1], col = "black", cex = 1)
159
160 #Estabelecendo a significancia estatistica com o metodo Qui-Quadrado
161 chisq.test(tab)
162
163 #Carregando os topicos arquiteturais
164 architectural_types <- dbGetQuery(con,'select id_architectural_topic,
165 id_commit_commit from architectural_topic join commit_architectural_topic
166 on id = id_architectural_topic')
167 #Reestabelecendo o nome do campo de id
168 colnames(acc_metric_mean_3)[1] <- "id_commit_commit"
169 #Selecionando os commits que registraram aumento nos valores da metrica
```

acc.R: Script de análise estatística da variações da métrica ACC utilizando a linguagem R (parte 5).

```
# ACC e seus respectivos topicos arquiteturais
acc_metric_mean_3_type <- inner_join(acc_metric_mean_3,
 architectural_types,  by=c("id_commit_commit")) %>%
  mutate(sinal = sign(delta)) %>%
  filter(delta>0)
#Sumarizando os dados obtidos para commits com aumento no valor da
# metrica ACC
acc_metric_mean_3_type_summary <- acc_metric_mean_3_type %>%
  group_by(id_architectural_topic) %>%
  count(id_architectural_topic)
#Exibindo o sumario obtido
acc_metric_mean_3_type_summary;

#Transformando a variavel acc_metric_mean_3_type_summary em um vetor de
# quantidades
acc_metric_mean_3_type_summary <- c(acc_metric_mean_3_type_summary$n)
#Definindo a configuracao do barplot
par(mar = c(3,6,1,1))
#Construindo o barplot com os topicos arquiteturais para commits com
# aumento no valor da metrica ACC
bp_all_m1 <- barplot(acc_metric_mean_3_type_summary,
  names.arg = names(acc_metric_mean_3_type_summary),
  main = "Distribuicao de Commits com aumento no valor da metrica ACC",
  horiz = TRUE, beside = TRUE, yaxt = "n",
  space = 0.5, col = 4, xlim = c(0,30))
#Definindo o nome dos topicos arquiteturais (Eixo Y)
labels_b_all_m1 <- c("Topico 1", "Topico 2", "Topico 3", "Topico 4",
  "Topico 5", "Topico 6", "Topico 7", "Topico 8", "Topico 9",
  "Topico 10","Topico 11", "Topico 12", "Topico 13", "Topico 15")
#Inserindo os labels do eixo Y
text(0, bp_all_m1, labels = labels_b_all_m1, pos = 2, xpd=TRUE )
#Inserindo os valores em cada barra (correspondente a quantidade de cada
# topico)
text(0, bp_all_m1, acc_metric_mean_3_type_summary, pos = 4, xpd=TRUE )

#Selecionando os commits que registraram diminuicao nos valores da metrica
# ACC e seus respectivos topicos arquiteturais
acc_metric_mean_3_type <- inner_join(acc_metric_mean_3,
 architectural_types,  by=c("id_commit_commit")) %>%
 mutate(sinal = sign(delta)) %>%
 filter(delta<0)
#Sumarizando os dados obtidos para commits com diminuicao no valor da
# metrica ACC
acc_metric_mean_3_type_summary <- acc_metric_mean_3_type %>%
 group_by(id_architectural_topic) %>%
 count(id_architectural_topic)
#Exibindo o sumario obtido
acc_metric_mean_3_type_summary;
```

acc.R: Script de análise estatística da variações da métrica ACC utilizando a linguagem R (parte 6).

```
218 #Transformando a variavel acc_metric_mean_3_type_summary em um vetor de
219 # quantidades
220 acc_metric_mean_3_type_summary <- c(acc_metric_mean_3_type_summary$n)
221 #Definindo a configuracao do barplot
222 par(mar = c(3,6,1,1))
223 #Construindo o barplot com os topicos arquiteturais para commits com
224 # diminuicao nos valores da metrica ACC
225 bp_all_m1 <- barplot(acc_metric_mean_3_type_summary,
226  names.arg = names(acc_metric_mean_3_type_summary),
227  main = "Distribuicao de Commits com diminuicao no valor da metrica ACC",
228  horiz = TRUE, beside = TRUE, yaxt = "n",
229  space = 0.5, col = 3, xlim = c(0,30))
230 #Definindo o nome dos topicos arquiteturais (Eixo Y)
231 labels_b_all_m1 <- c("Topico 1", "Topico 3", "Topico 4",
232  "Topico 6", "Topico 7", "Topico 8", "Topico 9",
233  "Topico 10","Topico 11", "Topico 13", "Topico 14", "Topico 15")
234 #Inserindo os labels do eixo Y
235 text(0, bp_all_m1, labels = labels_b_all_m1, pos = 2, xpd=TRUE )
236 #Inserindo os valores em cada barra (correspondente a quantidade de cada
237 # topico)
238 text(0, bp_all_m1, acc_metric_mean_3_type_summary, pos = 4, xpd=TRUE )
239
240 #Selecionando os commits que nao registraram variacao nos valores da
241 # metrica ACC e seus respectivos topicos arquiteturais
242 acc_metric_mean_3_type <- inner_join(acc_metric_mean_3,
243  architectural_types, by=c("id_commit_commit")) %>%
244  mutate(sinal = sign(delta)) %>%
245  filter(delta==0)
246 #Sumarizando os dados obtidos para commits com diminuicao no valor da
247 # metrica ACC
248 acc_metric_mean_3_type_summary <- acc_metric_mean_3_type %>%
249  group_by(id_architectural_topic) %>%
250  count(id_architectural_topic)
251 #Exibindo o sumario obtido
252 acc_metric_mean_3_type_summary;
253
254 #Transformando a variavel acc_metric_mean_3_type_summary em um vetor de
255 # quantidades
256 acc_metric_mean_3_type_summary <- c(acc_metric_mean_3_type_summary$n)
257 #Definindo a configuracao do barplot
258 par(mar = c(3,6,1,1))
259 #Construindo o barplot com os topicos arquiteturais para commits sem
260 # variacao nos valores da metrica ACC
261 bp_all_m1 <- barplot(acc_metric_mean_3_type_summary,
262  names.arg = names(acc_metric_mean_3_type_summary),
263  main = "Distribuicao de Commits com manutencao do valor da metrica ACC",
264  horiz = TRUE, beside = TRUE, yaxt = "n",
265  col = 2, space = 0.5, xlim = c(0,35))
```

acc.R: Script de análise estatística da variações da métrica ACC utilizando a linguagem R (parte 7).

```
266 #Definindo o nome dos topicos arquiteturais (Eixo Y)
267 labels_b_all_m1 <- c("Topico 1", "Topico 3", "Topico 4", "Topico 5",
268  "Topico 6", "Topico 7", "Topico 8", "Topico 9",
269  "Topico 10","Topico 11", "Topico 12", "Topico 13", "Topico 15")
270 #Inserindo os labels do eixo Y
271 text(0, bp_all_m1, labels = labels_b_all_m1, pos = 2, xpd=TRUE )
272 #Inserindo os valores em cada barra (correspondente a quantidade de cada
273 # topico)
274 text(0, bp_all_m1, acc_metric_mean_3_type_summary, pos = 4, xpd=TRUE )
275
276 #Executando o teste de Significancia de Wilcox, comparando os valores
277 # obtidos para a media da metrica ACC em commits arquiteturais e
278 # nao-arquiteturais
279 wilcox.test(acc_metric_mean_3$delta, acc_metric_mean_4$delta)
280
281 #Exibindo a estatistica descritiva dos valores da metrica ACC para commits
282 # arquiteturais
283 summary(acc_metric_mean_3$delta)
284
285 #Exibindo a estatistica descritiva dos valores da metrica ACC para commits
286 # nao-arquiteturais
287 summary(acc_metric_mean_4$delta)
288
289 #Boxplot contendo a variacao media dos valores da metrica ACC, comparando
290 # commits arquiteturais e nao-arquiteturais
291 boxplot(acc_metric_mean_3$delta, acc_metric_mean_4$delta, names=
292  c("Com Mudancas Arquiteturais","Sem Mudancas Arquiteturais"))
```

Appendix

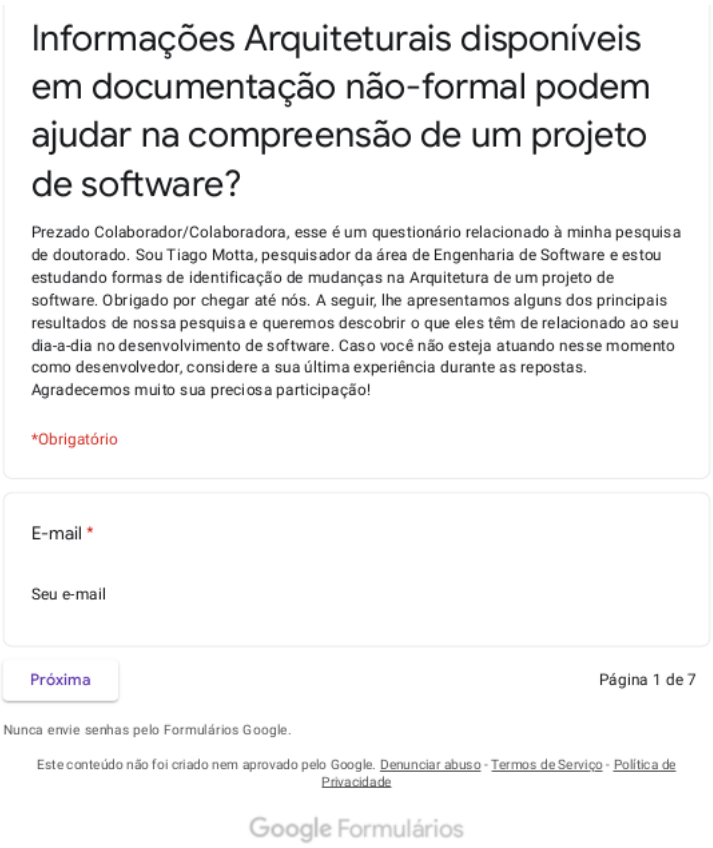
E

FORMULÁRIOS UTILIZADOS NO ESTUDO DE AVALIAÇÃO DOS RESULTADOS OBTIDOS PELA MINERAÇÃO DE TRAÇOS ARQUITETURAIS E EXTRAÇÃO DE MÉTRICAS DE CÓDIGO EM COMMITS COM ALTERAÇÃO NA ARQUITETURA

Este Apêndice apresenta o formulário utilizado no estudo de avaliação dos resultados obtidos por esta tese e apresentado no Capítulo 5 e direcionado para os Desenvolvedores que tem como linguagem nativa o Português. Os outros três formulários são derivações desse. Para Desenvolvedores foi disponibilizada uma versão em língua Inglesa desse formulário. Para os Pesquisadores, houve uma adaptação nas perguntas, conforme já descrito na Tabela 5.1. A versão deste formulário para os Pesquisadores também esteve disponível em língua Portuguesa e em língua Inglesa.

E.1 FORMULÁRIO APRESENTADO À DESENVOLVEDORES EM LÍNGUA PORTUGUESA

A seguir são apresentados um conjunto de figuras que juntas, representam as telas oferecidas aos desenvolvedores que optaram por preencher o formulário em língua Portuguesa. Para completa compreensão, considere as figuras (E.1 à E.16) na sequência apresentada.



Informações Arquiteturais disponíveis em documentação não-formal podem ajudar na compreensão de um projeto de software?

Prezado Colaborador/Colaboradora, esse é um questionário relacionado à minha pesquisa de doutorado. Sou Tiago Motta, pesquisador da área de Engenharia de Software e estou estudando formas de identificação de mudanças na Arquitetura de um projeto de software. Obrigado por chegar até nós. A seguir, lhe apresentamos alguns dos principais resultados de nossa pesquisa e queremos descobrir o que eles têm de relacionado ao seu dia-a-dia no desenvolvimento de software. Caso você não esteja atuando nesse momento como desenvolvedor, considere a sua última experiência durante as repostas. Agradecemos muito sua preciosa participação!

*Obrigatório

E-mail *

Seu e-mail

Próxima

Página 1 de 7

Nunca envie senhas pelo Formulários Google.

Este conteúdo não foi criado nem aprovado pelo Google. Denunciar abuso - Termos de Serviço - Política de Privacidade

Google Formulários

Figure E.1: Formulário para desenvolvedores em língua Portuguesa - Seção 1.

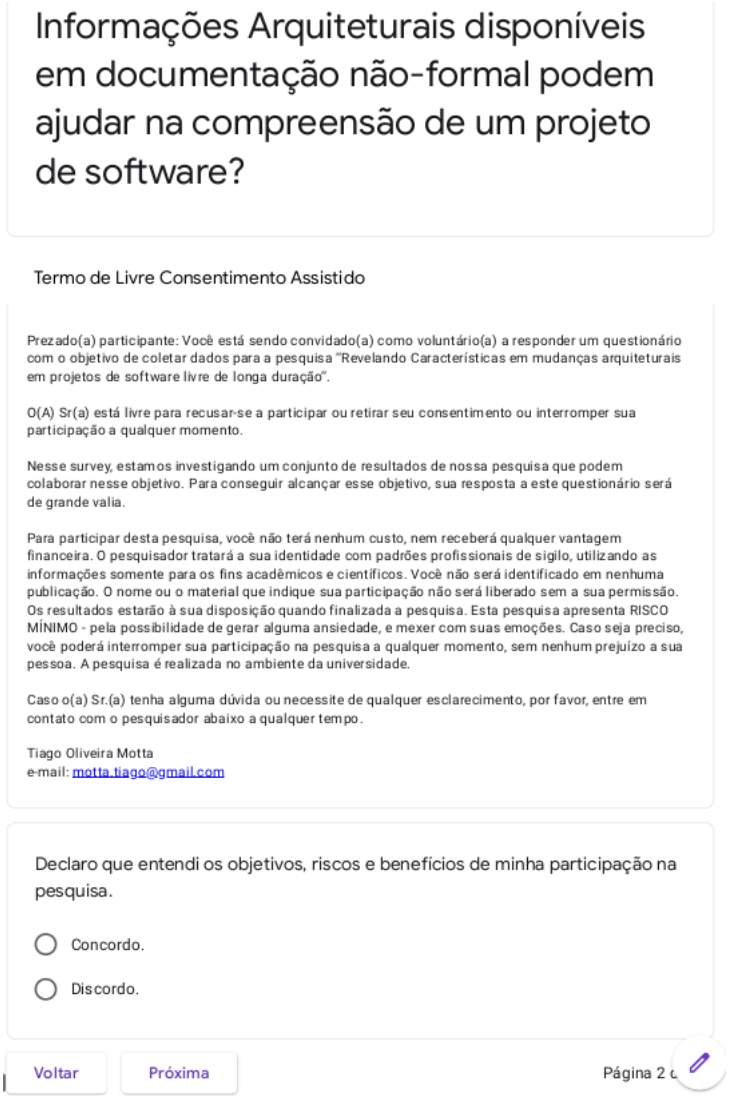


Figure E.2: Formulário para desenvolvedores em língua Portuguesa - Seção 2.

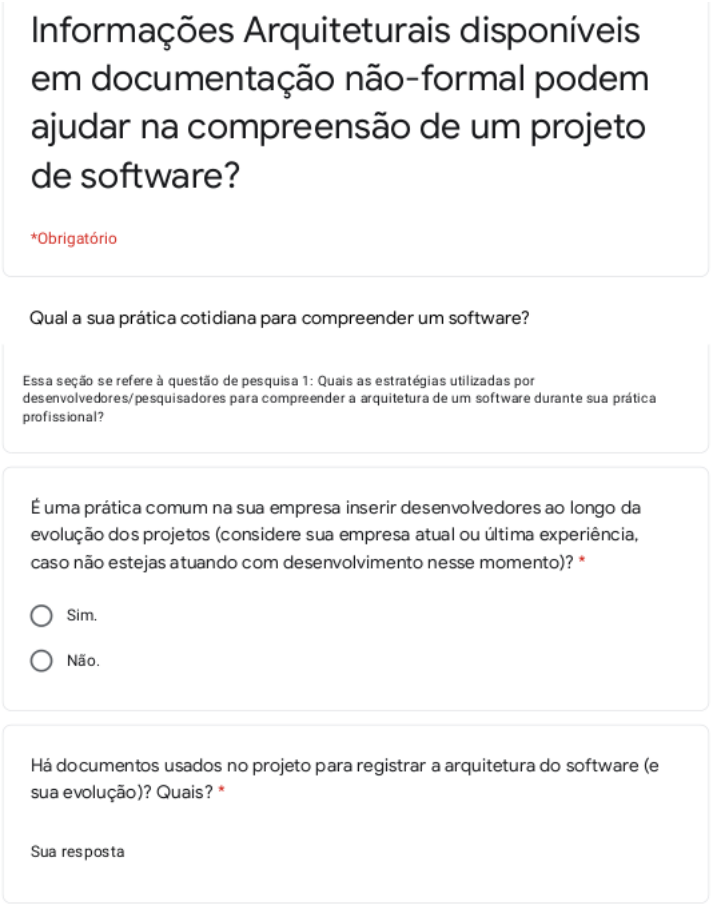


Figure E.3: Formulário para desenvolvedores em língua Portuguesa - Seção 3 (parte 1).

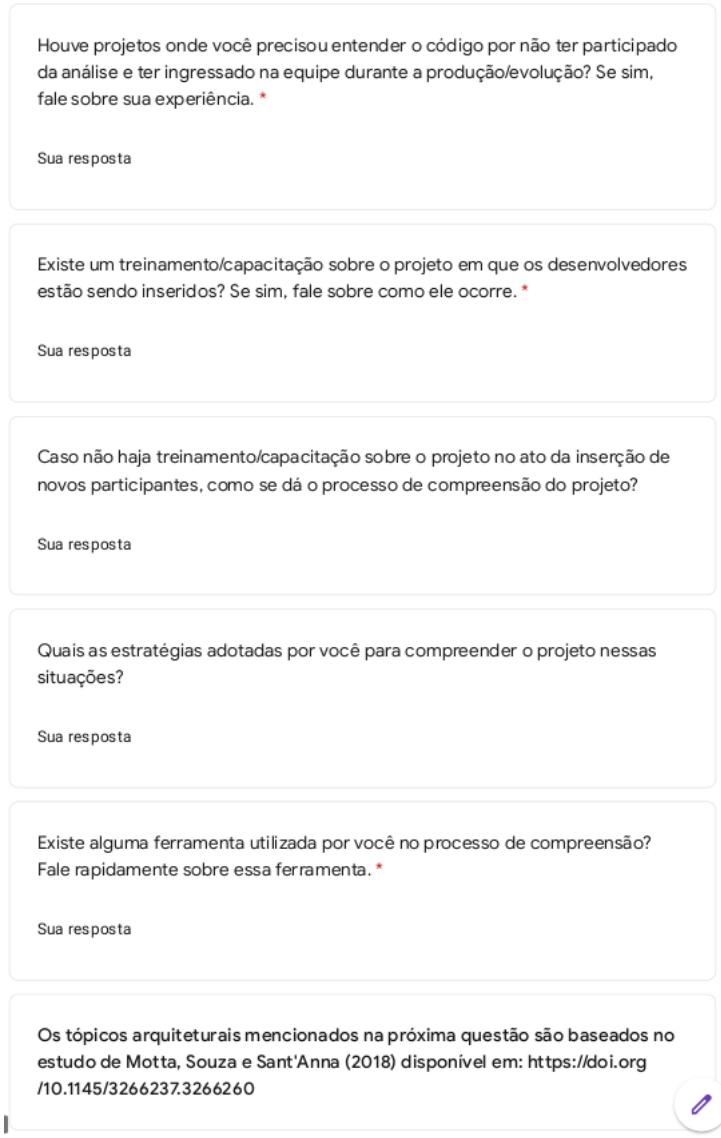


Figure E.4: Formulário para desenvolvedores em língua Portuguesa - Seção 3 (parte 2).

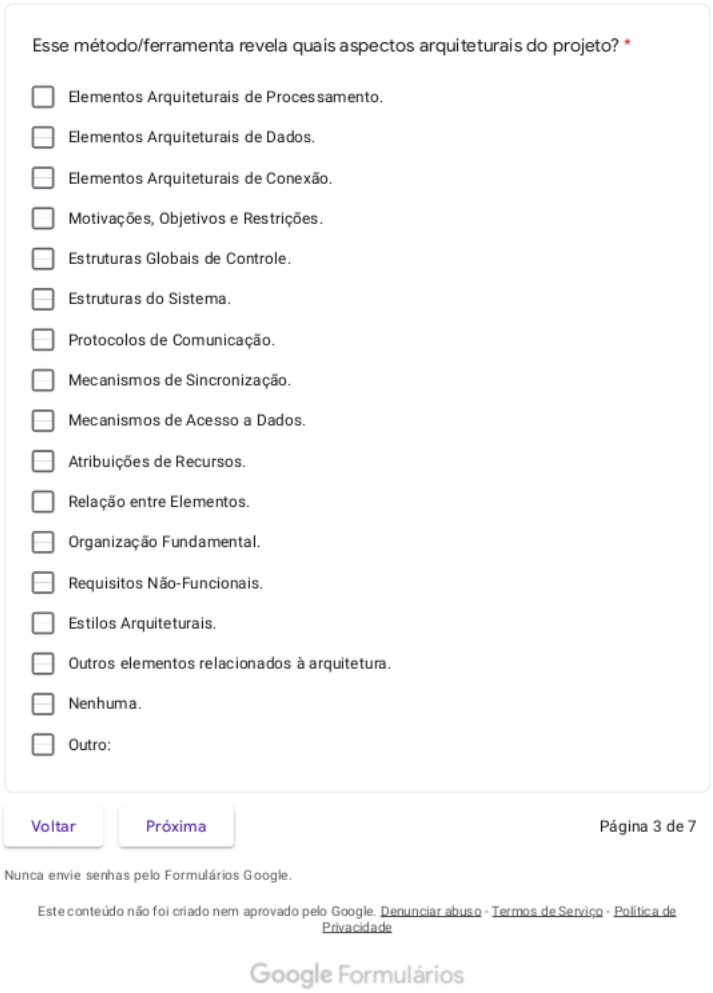


Figure E.5: Formulário para desenvolvedores em língua Portuguesa - Seção 3 (parte 3).

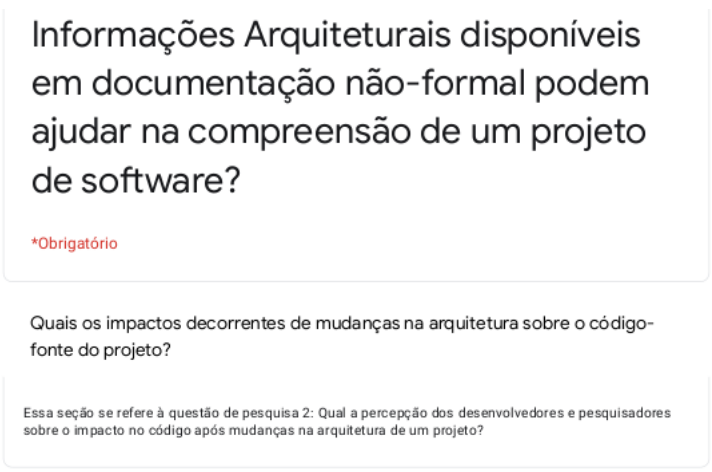


Figure E.6: Formulário para desenvolvedores em língua Portuguesa - Seção 4 (parte 1).

Métricas coletadas e variações obtidas. O trabalho de Oliveira Filho (2013) apresenta uma breve descrição dessas métricas: https://teses.usp.br/teses/disponiveis/45/45134/tde-25092013-142158/publico/dissertacao.pdf

MÉTRICAS DE TAMANHO	
Nome da Métrica	**Tendência Verificada**
Tamanho Médio dos Métodos **(AMLOC)**	**AUMENTO**
Número Médio de Parâmetros **(ANPM)**	**REDUÇÃO**
Linhas de Código **(LOC)**	**AUMENTO**
Número de Atributos **(NOA)**	**MANUTENÇÃO**
Número de Métodos **(NOM)**	**AUMENTO**
Tamanho do Maior Método **(MMLOC)**	**AUMENTO**
MÉTRICAS DE COMPLEXIDADE	
Complexidade Ciclomática Média **(ACCM)**	**REDUÇÃO**
Número de Atributos Públicos **(NPA)**	**MANUTENÇÃO**
Número de Métodos Públicos **(NPM)**	**MANUTENÇÃO**
MÉTRICAS DE ACOPLAMENTO	
Conexões Aferentes por Classe **(ACC)**	**AUMENTO**
Acoplamento entre Objetos **(CBO)**	**MANUTENÇÃO**
Profundidade da Árvore de Herança **(DiT)**	**MANUTENÇÃO**
Número de Classes Filhas **(NOC)**	**MANUTENÇÃO**
Respostas por Classe **(RFC)**	**AUMENTO**
MÉTRICAS DE COESÃO	
Perda de Coesão por Método **(LCOM4)**	**MANUTENÇÃO**
Complexidade Estrutural **(SC)**	**AUMENTO**

Figure E.7: Formulário para desenvolvedores em língua Portuguesa - Seção 4 (parte 2).

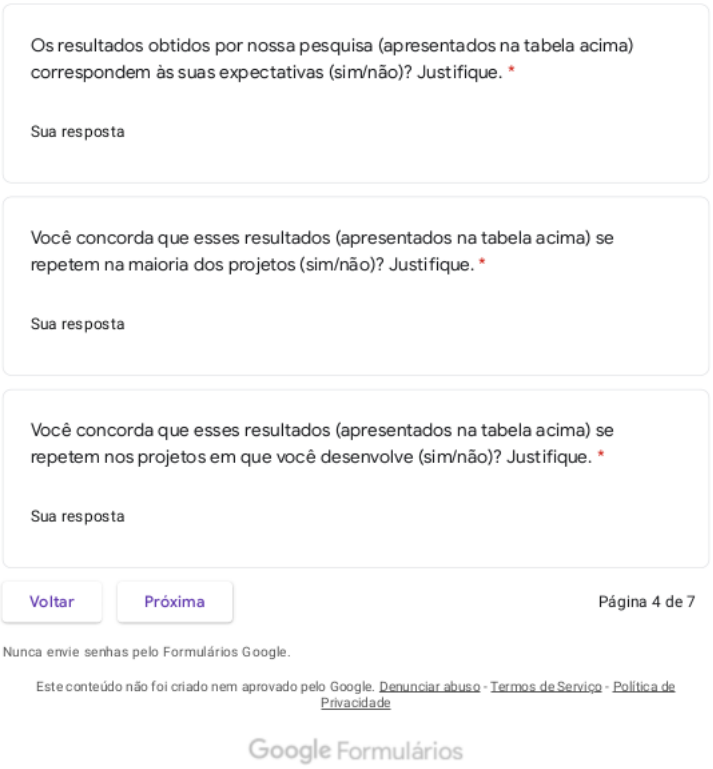


Figure E.8: Formulário para desenvolvedores em língua Portuguesa - Seção 4 (parte 3).

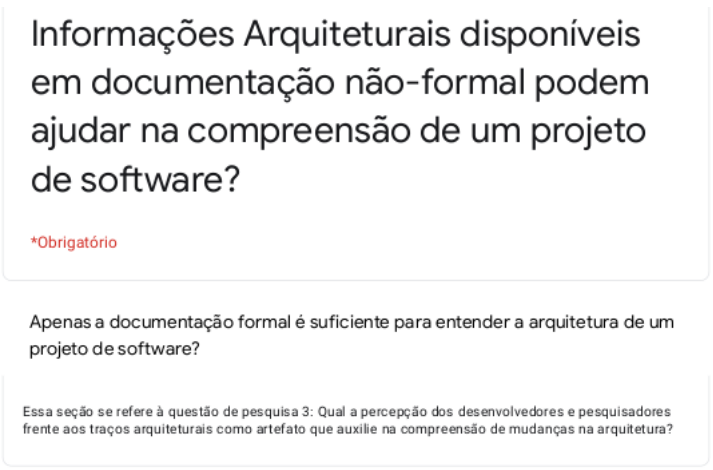


Figure E.9: Formulário para desenvolvedores em língua Portuguesa - Seção 5 (parte 1).

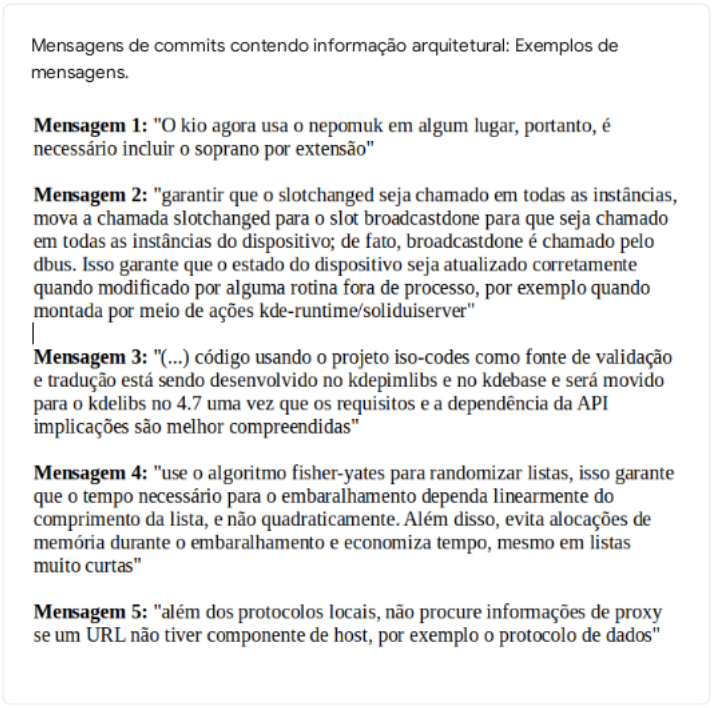


Figure E.10: Formulário para desenvolvedores em língua Portuguesa - Seção 5 (parte 2).

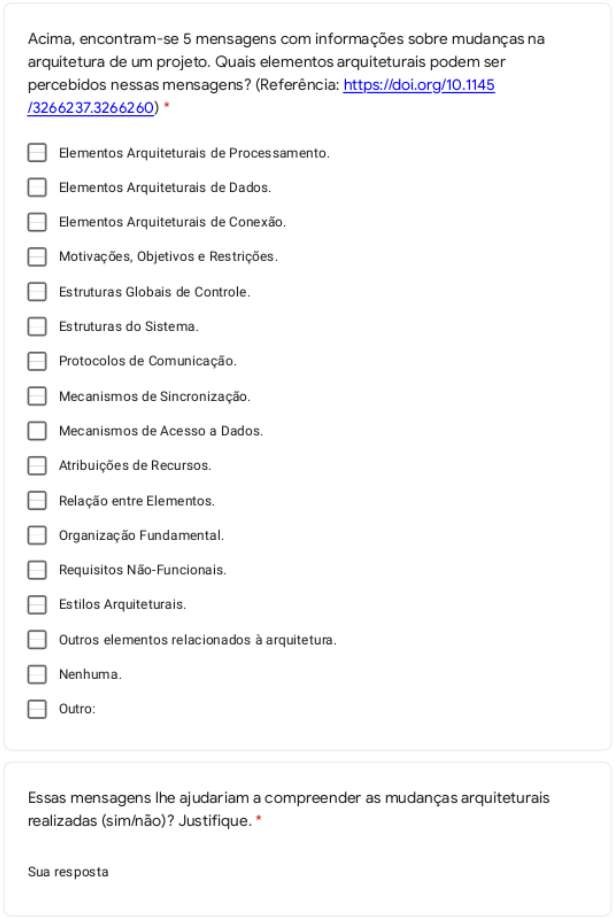


Figure E.11: Formulário para desenvolvedores em língua Portuguesa - Seção 5 (parte 3).

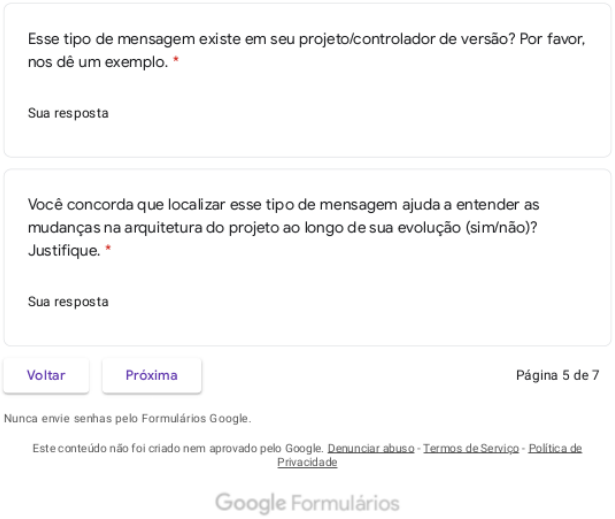


Figure E.12: Formulário para desenvolvedores em língua Portuguesa - Seção 5 (parte 4).

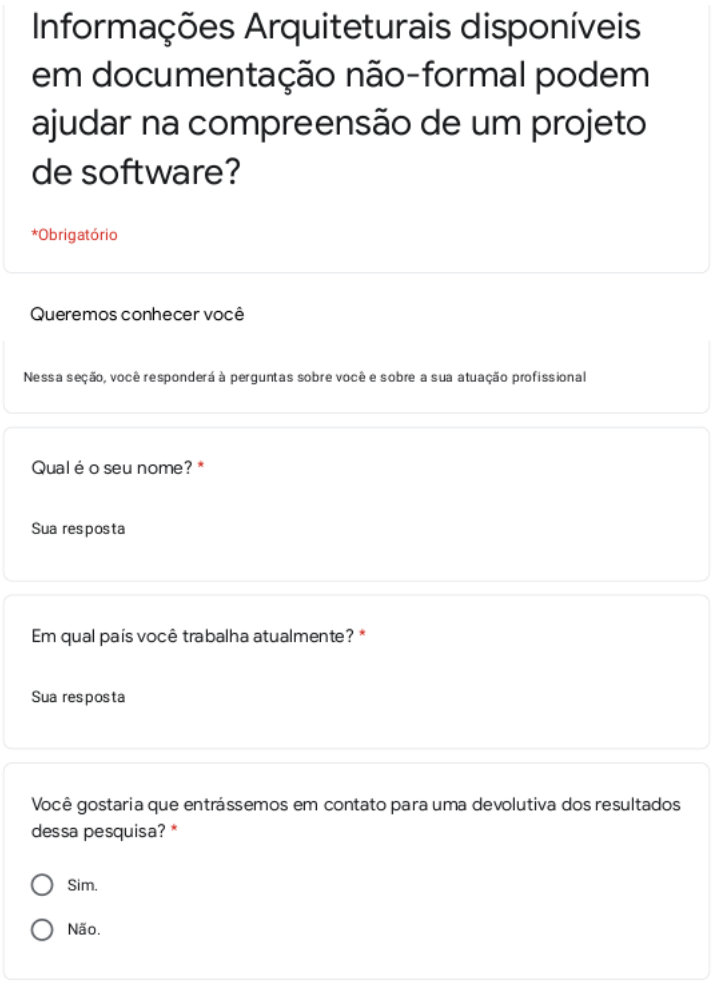


Figure E.13: Formulário para desenvolvedores em língua Portuguesa - Seção 6 (parte 1).

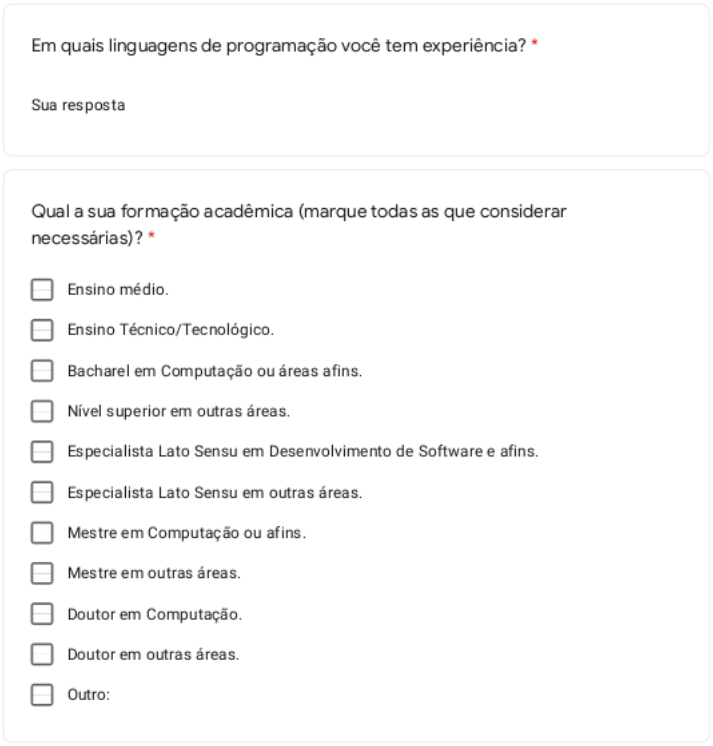


Figure E.14: Formulário para desenvolvedores em língua Portuguesa - Seção 6 (parte 2).

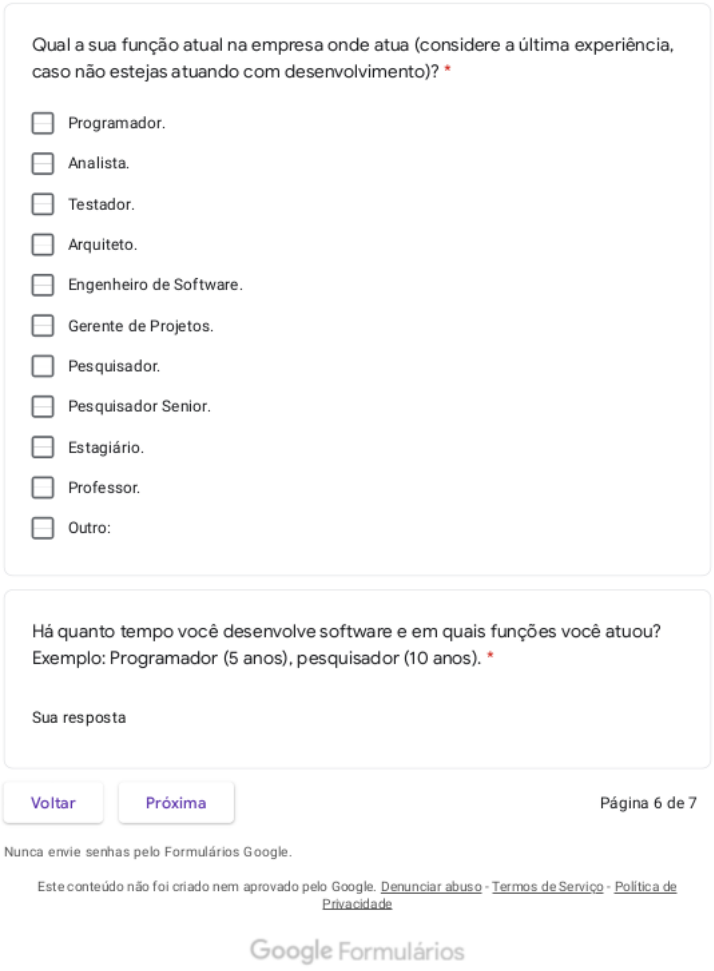


Figure E.15: Formulário para desenvolvedores em língua Portuguesa - Seção 6 (parte 3).

Figure E.16: Formulário para desenvolvedores em língua Portuguesa - Parte 7.