UNIVERSIDADE FEDERAL FLUMINENSE

CRISTIANE DA SILVA RODRIGUES PEREIRA

Um estudo sobre a correlação entre acoplamentos de software entre ramos e esforços de merge

NITERÓI

2020

UNIVERSIDADE FEDERAL FLUMINENSE

CRISTIANE DA SILVA RODRIGUES PEREIRA

Um estudo sobre a correlação entre acoplamentos de software entre ramos e esforços de merge

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Engenharia de Sistemas e Informação.

Orientador:
Leonardo Gresta Paulino Murta

Co-orientador:
Gleiph Ghiotto Lima de Menezes

NITERÓI

2020

Ficha catalográfica automática - SDC/BEE
Gerada com informações fornecidas pelo autor

P436e Pereira, Cristiane da Silva Rodrigues
Um estudo sobre a correlação entre acoplamentos de software entre ramos e esforços de merge / Cristiane da Silva Rodrigues Pereira ; Leonardo Gresta Paulino Murta, orientador ; Gleiph Ghiotto Lima de Menezes, coorientador. Niterói, 2020.
78 p.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2020.

DOI: http://dx.doi.org/10.22409/PGC.2020.m.11065108796

1. Engenharia de Software. 2. Desenvolvimento de Software. 3. Linguagem de programação. 4. Produção intelectual. I. Murta, Leonardo Gresta Paulino, orientador. II. Menezes, Gleiph Ghiotto Lima de, coorientador. III. Universidade Federal Fluminense. Instituto de Computação. IV. Título.

CDD -

Bibliotecário responsável: Sandra Lopes Coelho - CRB7/3389

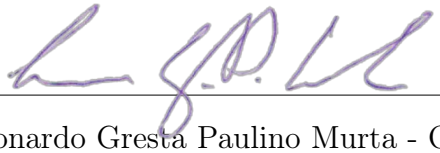
CRISTIANE DA SILVA RODRIGUES PEREIRA

Um estudo sobre a correlação entre acoplamentos de *software* entre ramos e esforços de *merge*

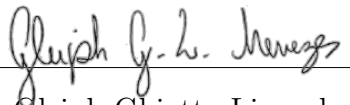
Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Mestre em Computação. Área de concentração: Engenharia de Sistemas e Informação.

Aprovada em novembro de 2020.

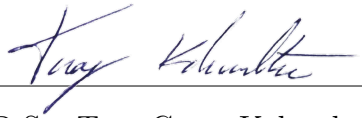
BANCA EXAMINADORA



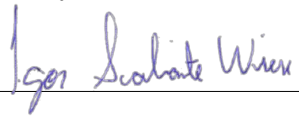
Prof. D.Sc. Leonardo Gresta Paulino Murta - Orientador, UFF



Prof. D.Sc. Gleiph Ghiotto Lima de Menezes - Co-Orientador, UFJF



Prof. D.Sc. Troy Costa Kohwalter, UFF



Prof. D.Sc. Igor Scaliante Wiese, UTFPR

Niterói

2020

"O conformismo é o carcereiro da liberdade e o inimigo do crescimento."

(John Fitzgerald Kennedy)

"O período de maior ganho em conhecimento e experiência é o período mais difícil da vida de alguém."

(Dalai Lama)

Agradecimentos

Primeiramente, gostaria de agradecer a Deus por ter me permitido ingressar em um curso de mestrado e por ter me sustentado até aqui.

Gostaria de agradecer ao meu marido e ao meu filho pela compreensão da minha ausência devido à dedicação ao mestrado, e por me darem força e coragem para continuar nesta jornada.

Gostaria de agradecer a minha mãe por sempre me incentivar e por ter cuidado do meu filho nos momentos em que mais precisei.

Gostaria de agradecer ao meu orientador Leonardo Murta pela paciência, pelos ensinamentos, pela dedicação e por sempre me incentivar a seguir em frente. O Prof. Leonardo é um exemplo de professor e de ser humano.

Gostaria de agradecer ao meu co-orientador Gleiph Ghiotto pela paciência, por me auxiliar na adaptação da ferramenta *merge-nature* e pela dedicação na correção do artigo e dos capítulos da dissertação.

Gostaria de agradecer ao Prof. José Ricardo pelo fundamental apoio para a execução em GPU, dos experimentos para a extração do acoplamento lógico.

Gostaria de agradecer à Tayane por ter sido uma grande amiga durante o curso e pela parceria em trabalhos que desenvolvemos juntas.

Finalmente, gostaria de agradecer à Marinha do Brasil pela oportunidade da concessão da licença para cursar o mestrado.

Resumo

O desenvolvimento de software colaborativo exige trabalho paralelo, e as alterações simultâneas precisam ser integradas posteriormente. Quando um *merge* falha, devido a conflitos diretos ou indiretos, o desenvolvedor deve intervir manualmente. Até o momento, os trabalhos existentes na literatura fornecem algumas evidências iniciais de que o acoplamento estrutural é uma das razões para conflitos indiretos. No entanto, os trabalhos existentes não avaliam formalmente até que ponto o acoplamento estrutural se correlaciona com o esforço de *merge* e não consideram outros tipos de acoplamentos, como lógico e conceitual. Neste trabalho, foram propostas métricas para quantificar os acoplamentos estrutural, lógico e conceitual entre as mudanças que ocorrem nos ramos. Em seguida, foi investigada a correlação entre as métricas de acoplamento de software e as métricas de esforço de *merge* (ou seja, retrabalho, trabalho desperdiçado e trabalho extra) em 12 projetos de código aberto, totalizando 6.376 *merges*. Observou-se uma fraca correlação entre todas as métricas de acoplamento e o esforço de trabalho extra. No entanto, alinhado à literatura, notou-se que a correlação entre o acoplamento estrutural e o esforço de trabalho extra é maior que os outros dois. Além disso, observou-se uma correlação moderada entre os acoplamentos estrutural e conceitual e os esforços de retrabalho e trabalho desperdiçado, mas foi identificada uma correlação baixa entre o acoplamento lógico e os esforços de retrabalho e trabalho desperdiçado. Finalmente, pôde-se observar uma correlação moderada entre os acoplamentos lógico e conceitual e uma fraca correlação entre os acoplamentos estrutural e lógico, bem como entre os acoplamentos estrutural e conceitual.

Palavras-chave: Acoplamento estrutural; acoplamento lógico; acoplamento conceitual; esforço de *merge*; correlação.

Abstract

Collaborative software development demands parallel work, and the concurrent changes need to be merged afterwards. When a merge fails, either due to direct or indirect conflicts, the developer must intervene manually. Thus far, existing work in the literature provides some initial evidence that structural coupling is one of the reasons for indirect conflicts. However, the existing work does not formally assess the extent in which structural coupling correlates with merge effort and does not consider other types of couplings, such as logical and conceptual. In this work, we propose metrics to quantify the structural, logical, and conceptual couplings among changes that occur across branches. Then, we investigate the correlation between the software coupling metrics and the merge effort metrics (i.e., rework, wasted work, and extra work) over 12 open-source projects, summing up 6,376 merges. We could observe a weak correlation between all coupling metrics and the extra work effort. However, aligned to the literature, we note that the correlation between the structural coupling and the extra work effort is greater than the other two. In addition, we observed a moderate correlation between structural and conceptual couplings, and rework and wasted work efforts, but a low correlation between logical coupling and rework and wasted work efforts. Finally, we could observe a moderate correlation was observed between logical and conceptual couplings and a weak correlation between structural and logical couplings, as well as between structural and conceptual couplings.

Keywords: Structural coupling; logical coupling; conceptual coupling; merge effort; correlation.

Lista de Figuras

Lista de Tabelas

Conteúdo

Capítulo 1

Introdução

1.1 Motivação

O desenvolvimento de *software* em larga escala geralmente requer vários desenvolvedores trabalhando em paralelo em ramos implícitos ou explícitos [9]. O trabalho simultâneo em ramos implícitos ocorre quando os desenvolvedores clonam um repositório e realizam *commits* locais em paralelo com outros desenvolvedores. Por outro lado, o trabalho simultâneo em ramos explícitos ocorre quando os desenvolvedores criam ramos nomeados para implementar novos recursos, separar a manutenção corretiva da manutenção perfectiva, customizar o sistema, etc. [8]. Nesse caso, vários desenvolvedores podem trabalhar no mesmo ramo, em paralelo com outros desenvolvedores que trabalham em ramos diferentes. Em ambos os casos, as alterações paralelas são usualmente reintegradas (ou seja, sofrem *merge*) em algum momento [35].

Infelizmente, os *merges* falham em 10 a 20% das vezes [13, 27] devido aos conflitos diretos ou indiretos entre as alterações realizadas em paralelo [34, 27]. Os conflitos diretos de *merge* ocorrem quando alterações paralelas afetam a mesma região do código-fonte e são detectados pelos sistemas de controle de versão durante o *merge*. Por outro lado, conflitos indiretos de *merge* ocorrem quando alterações paralelas, em diferentes regiões do código-fonte, são acopladas entre si [51]. Por exemplo, quando uma declaração de método é alterada em paralelo à sua invocação. Os sistemas de controle de versão tradicionais não são capazes de detectar conflitos indiretos que podem impedir que o sistema compile ou se comporte conforme o esperado durante a execução.

Quando os conflitos de *merge* ocorrem são necessárias intervenções do desenvolvedor, eventualmente exigindo esforço para resolvê-los. Esse esforço de *merge* pode ser dividido

em três tipos [37, 46]: retrabalho, trabalho desperdiçado e trabalho extra. O retrabalho consiste em alterações duplicadas nos ramos. O trabalho desperdiçado consiste em alterações realizadas nos ramos que não são incorporadas ao *merge*. Finalmente, o trabalho extra consiste em alterações realizadas durante o *merge* que não estavam disponíveis nos ramos. Como os conflitos de *merge* são considerados difíceis de resolver [52, 30, 35] e os conflitos indiretos não são detectados automaticamente pelos sistemas de controle de versão, é fundamental investigar se os acoplamentos entre ramos se correlacionam com os esforços de *merge*. Essas informações podem ajudar os construtores de ferramentas a conceber melhores ferramentas de percepção para predizer a dificuldade dos *merges*, os gerentes a alocar melhor os desenvolvedores para as tarefas ou os desenvolvedores para sincronizar as atividades entre si, visando reduzir o esforço de *merge*.

A literatura [52, 17] fornece algumas evidências iniciais de que o acoplamento estrutural [16] é uma das razões para conflitos indiretos. No entanto, eles não quantificam o grau em que o acoplamento estrutural se correlaciona com o esforço de *merge*. Além disso, outros tipos de acoplamento, como lógico [18, 47] e conceitual [42, 56] ainda não foram considerados na literatura e podem se correlacionar com o esforço de *merge*.

1.2 Objetivos

Diante da motivação apresentada, o objetivo principal desta dissertação é compreender, por meio da análise de repositórios de software, qual acoplamento entre ramos se correlaciona com os esforços de *merge*. Sendo assim, o objetivo deste estudo está subdividido em dois objetivos específicos:

- avaliar a correlação entre os acoplamentos e os esforços de *merge*; e
- analisar a influencia das características dos ramos na correlação entre os acoplamentos e os esforços de *merge*.

Para atingir esses objetivos específicos, foram analisados doze projetos de código aberto, na linguagem Java, obtidos no GitHub[1]. Com o intuito de automatizar a extração dos acoplamentos estrutural, lógico e conceitual, foram adaptadas três ferramentas, *merge-nature* [34], *TIPMerge* [15] e *Semantic Similarity Java* [2], que já coletavam os acoplamentos estrutural, lógico e conceitual, respectivamente, para que coletassem os

[1]https://github.com/

acoplamentos entre os métodos modificados entre os ramos. Para tal, foram criadas duas métricas de acoplamento entre ramos, uma absoluta e outra normalizada, as quais estão descritas no Capítulo 3. Finalmente, para extrair os esforços de retrabalho, trabalho desperdiçado e trabalho extra foi utilizada a ferramenta *merge-effort* [37]. Os objetivos foram convertidos em quatro questões de pesquisa:

- **QP1: Quais acoplamentos entre ramos se correlacionam com o outro?**
- **QP2: Quais esforços de *merge* se correlacionam com o outro?**
- **QP3: Quais acoplamentos entre ramos se correlacionam com os esforços de *merge*?**
- **QP4: Quais características (número de arquivos modificados, número de linhas modificadas, número de *commits* e número de autores) influenciam a correlação entre tipos de acoplamento e esforços de *merge*?**

Em relação à **QP1**, observou-se que a correlação entre os acoplamentos lógico e conceitual é moderada ($\rho = 0,61$), enquanto que a correlação entre os acoplamentos estrutural e lógico ($\rho = 0,40$), bem como entre os acoplamentos estrutural e conceitual é fraca ($\rho = 0,44$), mostrando que os acoplamentos podem se complementar, pois há uma sobreposição direcional entre eles. Em relação à **QP2**, observou-se, como esperado, uma forte correlação ($\rho = 0,87$) entre os esforços de retrabalho e trabalho desperdiçado. No entanto, a correlação é moderada entre o esforço de trabalho extra e os esforços de retrabalho ($\rho = 0,50$) e trabalho desperdiçado ($\rho = 0,63$). Considerando a **QP3**, observou-se uma correlação moderada entre o acoplamento estrutural e os esforços de retrabalho ($\rho = 0,51$) e trabalho desperdiçado ($\rho = 0,52$), como também, entre o acoplamento conceitual e os esforços de retrabalho ($\rho = 0,50$) e trabalho desperdiçado ($\rho = 0,50$) . No entanto, foi encontrada uma fraca correlação entre o acoplamento lógico e os esforços de retrabalho e trabalho desperdiçado. Além disso, foi identificada uma fraca correlação entre todas as métricas de acoplamento e o esforço de trabalho extra. Mesmo assim, alinhada à literatura [52, 17], a correlação entre o acoplamento estrutural e o esforço de trabalho extra foi maior dentre os três acoplamentos. Finalmente, em relação à **QP4**, observou-se que o número de arquivos modificados e o número de linhas modificadas influenciam a correlação entre acoplamentos e esforços de *merge*, com um grande tamanho de efeito. Além disso, o número de *commits* influencia a correlação entre o acoplamento estrutural e o esforço de trabalho desperdiçado. No entanto, o número de autores não foi estatisticamente significativo nesta análise.

1.3 Organização

Esta dissertação está organizada em cinco capítulos, incluindo a Introdução.

O Capítulo 2 apresenta os conceitos gerais sobre conflitos de *merge*, esforços de *merge* e acoplamentos de *software*, juntamente com os trabalhos relacionados a este estudo.

O Capítulo 3 apresenta a formalização do problema, um exemplo utilizado durante o documento para explicar os conceitos e novas métricas propostas para quantificar o acoplamento das alterações entre ramos.

O Capítulo 4 descreve as questões de pesquisa, os projetos utilizados nas análises e as adaptações realizadas nas ferramentas que detectam os acoplamentos estrutural, lógico e conceitual. Além disso, apresenta os resultados das análises com as respectivas discussões.

Por último, o Capítulo 5 apresenta as contribuições deste estudo, as limitações e os possíveis trabalhos futuros relacionados ao assunto.

Capítulo 2

Revisão da Literatura

2.1 Introdução

O desenvolvimento de *software* é comumente realizado em paralelo e o controle das alterações de cada desenvolvedor, de forma que se mantenha a integridade do código-fonte do *software*, é uma necessidade recorrente. Por isso, o uso de uma ferramenta que permita o controle de versão de forma que diversos desenvolvedores possam realizar alterações de forma concorrente, pode trazer melhorias para o processo de desenvolvimento.

O sistema de controle de versões (do inglês, *Version Control Systems* (VCS)) é uma ferramenta utilizada para gerenciar o desenvolvimento de *software* em paralelo. Ele permite a um grupo de desenvolvedores, trabalhar no mesmo conjunto de artefatos de *software* (i.e., código-fonte, modelos e outros) de forma que as alterações sejam armazenadas em um repositório. Para viabilizar o desenvolvimento em paralelo, os VCS empregam o uso de ramos para que os desenvolvedores trabalhem de forma isolada. Porém, as versões dos artefatos de *software*, modificadas nos ramos, usualmente precisam ser combinadas e este processo é conhecido como *merge*. Os VCS realizam *merge*, mas quando estes falham é necessário que o desenvolvedor intervenha manualmente para resolver os conflitos, que podem ser diretos ou indiretos.

Neste capítulo são descritos alguns conceitos básicos para compreender o restante desta dissertação, tais como, *merge*, conflitos de *merge* (direto e indireto), esforço de *merge* e acoplamentos de *software*. Para tanto, este capítulo está estruturado da seguinte forma: na Seção 2.2 são descritos os conflitos diretos e indiretos de *merge*, algumas causas dos conflitos indiretos e os tipos de esforços de *merge* (retrabalho, trabalho desperdiçado e trabalho extra) que são demandados durante a solução dos conflitos de *merge*; na Seção 2.3 é apresentado o estado da arte dos acoplamentos de *software*; e na Seção 2.4 são

descritos os trabalhos relacionados à pesquisa realizada nesta dissertação.

2.2 *Merge* de *software*

O desenvolvimento paralelo é muito comum em sistemas de *software* que possuem grandes equipes de desenvolvedores, onde as mudanças dos artefatos de *software* são realizadas em ramos diferentes. Os ramos permitem que as equipes trabalhem de forma isolada e não interfiram nas mudanças realizadas por uma outra equipe até que essas modificações sejam combinadas.

O uso de ramos em VCS permite que as equipes criem seu próprio ramo de desenvolvimento referente a uma determinada modificação do projeto, porém, em algum momento as mudanças realizadas nos ramos precisam ser integradas em um ramo principal ou nos ramos com as versões de entrega do *software* [10]. Portanto, *merge* é a combinação de duas ou mais versões de um artefato que foram criadas em paralelo [35]. Essas versões podem ser formadas por uma sequência de *commits* que foram realizados por diversos desenvolvedores e portanto, o *merge* dessas versões pode envolver conflitos de *merge*. O desenvolvimento paralelo pode frequentemente levar a conflitos que podem ter uma solução trivial ou uma solução mais demorada e difícil [41].

A Figura 2.1 ilustra o uso de ramos e o processo de *merge* de *software*. Primeiramente, o desenvolvedor cria o ramo *bugfix* e a partir do *commit* c_{10}, realiza algumas modificações no código e gera os *commits* c_{11} e c_{14}. Já no ramo *master*, um outro desenvolvedor trabalha em paralelo e gera os *commits* c_{12}, c_{13} e c_{15}. Ao final dessas modificações, é realizado o *merge* entre os ramos, e o *commit* de *merge* c_{16} é gerado a partir dos *commits* pais c_{14} e c_{15}.

Os VCSs tradicionais utilizam a técnica de *merge* textual que consistem em considerar os artefatos de *software* como arquivos de texto. Consequentemente, não consideram a sintaxe ou a semântica específica de cada artefato de *software* [35]. Com isso, as diferenças são baseadas apenas em função das linhas, palavras ou outras características de arquivos texto.

Duas técnicas de *merge* são geralmente implementadas em VCS: *two-way* e *three-way*. Durante a combinação das duas versões de *software*, a técnica de *merge two-way* leva em consideração apenas essas duas versões. Já a técnica de *merge three-way*, leva em conta as duas versões a serem combinadas, bem como o ancestral comum, ou seja, o ponto de onde começou o ramo. No entanto, a técnica *three-way* é mais eficaz do que a técnica *two-way*,

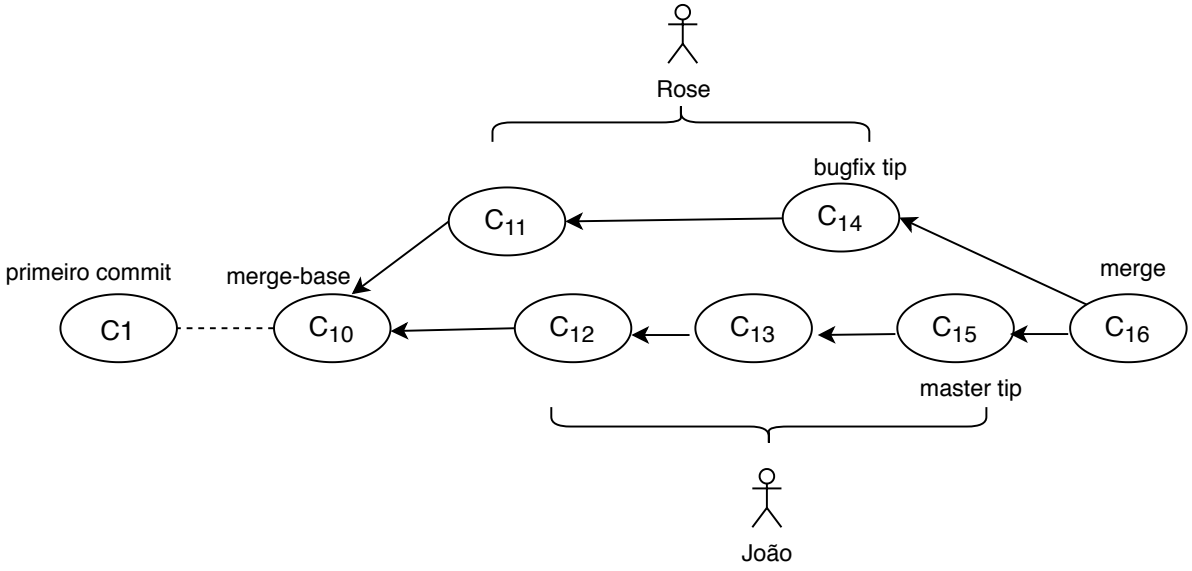
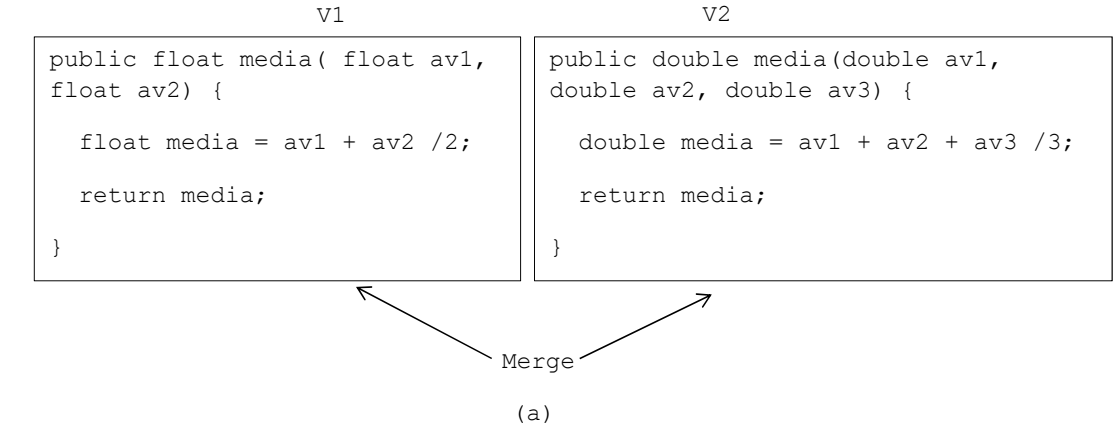


Figura 2.1: Exemplo de *merge* entre ramos.

porque as informações contidas no ancestral comum também são utilizadas durante o processo de *merge*. Com isso, mais conflitos podem ser detectados e portanto, é a técnica mais utilizada pelos VCS.

A Figura 2.2 (a) ilustra o funcionamento das técnicas de *merge two-way*, onde somente as versões V1 e V2 são utilizadas para realizar o *merge*. A Figura 2.2 (b) ilustra a técnica de *merge three-way*, onde a base (ancestral comum) e as versões V1 e V2 são consideradas para o processo de *merge*.



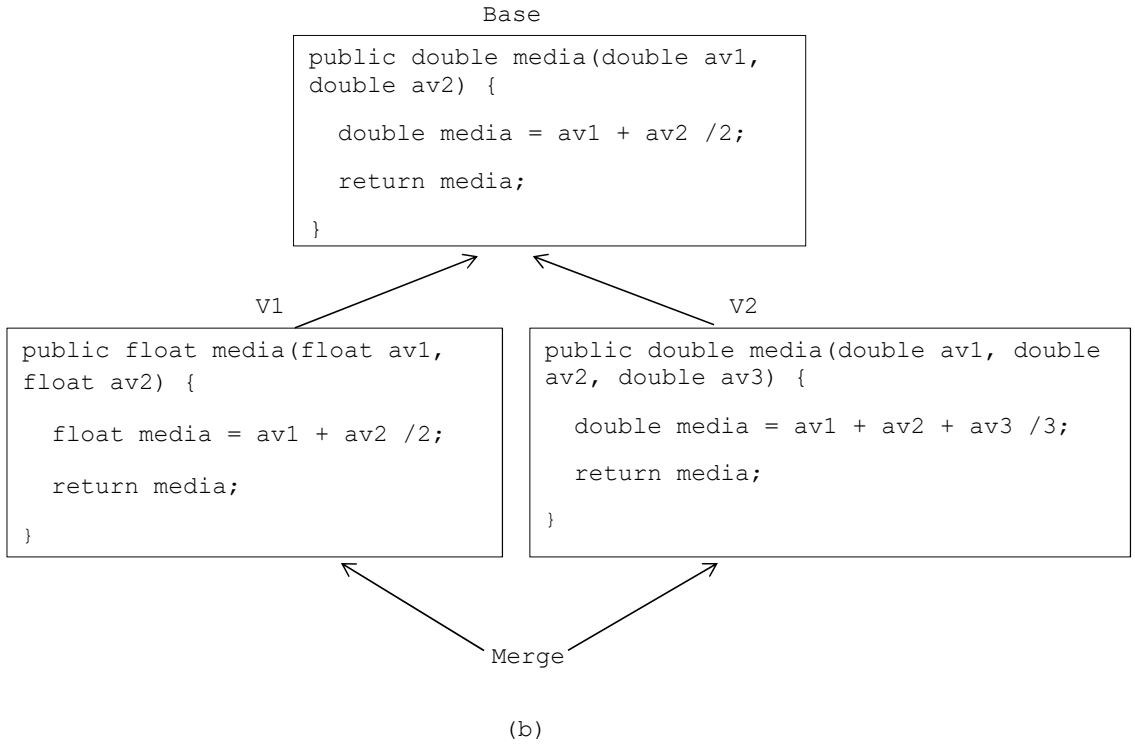


Figura 2.2: Técnicas de *merge*: *two-way* e *three-way*.

Conflitos de *merge* podem ser classificados como diretos ou indiretos. O conflito direto consiste na mudança em paralelo sobre a mesma região de um artefato. Já os conflitos indiretos ocorrem quando diferentes artefatos são modificados por diferentes desenvolvedores e o sistema deixa de se comportar como esperado [51].

Enquanto os conflitos diretos são detectados automaticamente pelos VCS no momento em que o *merge* é realizado, os conflitos indiretos são detectados apenas em tempo de compilação ou de execução. Ou seja, quando o código falha ao compilar ou executar devido, respectivamente, às quebras sintáticas ou semânticas geradas pela alteração dos artefatos. Vale a pena notar que os conflitos indiretos são mais difíceis de serem identificados e possivelmente ocorrem devido à existência de um acoplamento entre os artefatos que foram modificados [15, 32]. Por exemplo, quando dois desenvolvedores estão trabalhando em paralelo e um desenvolvedor na sua área de trabalho exclui um método de uma classe que um outro desenvolvedor acabou de incluir a chamada a este método em outra classe; no momento do *merge* não ocorrerá o conflito, mas em tempo de compilação será detectado o conflito, visto que há um acoplamento estrutural entre os métodos modificados.

Quando a operação de *merge* falha devido a conflitos diretos (ou seja, textuais) ou indiretos (ou seja, sintáticos ou semânticos), é necessária uma intervenção do desenvolvedor, eventualmente, exigindo algum esforço para resolvê-los. Portanto, o desenvolvedor precisa realizar o *merge* de forma manual e, para isso, deve optar por umas das alternativas: concatenar as duas versões em alguma ordem, escolher entre uma das duas versões, combinar as duas versões ou escrever um novo código [34]. Alguns trabalhos anteriores desenvolveram métricas para quantificar três tipos de esforços relacionados ao *merge* [45, 46, 49, 37]: retrabalho, trabalho desperdiçado e trabalho extra. O retrabalho ocorre quando os desenvolvedores executam ações idênticas em paralelo nos ramos que estão sendo integrados. O trabalho desperdiçado ocorre devido a ações que foram executadas durante o desenvolvimento paralelo, mas foram descartadas durante o *merge*. Finalmente, o trabalho extra é uma consequência das ações adicionais que o desenvolvedor precisa executar para resolver um conflito de *merge*.

Com base na formalização definida por Moura e Murta [37], para obter as referidas ações dos esforços de *merge*, primeiramente são realizadas três operações de *diff*. O primeiro *diff* é realizado entre o *commit* do *merge-base* ($commit_{base}$) e o *commit* de *merge* ($commit_{merge}$), para obter as ações que foram incorporadas ao *merge*. De acordo com a Figura 2.1, os $commit_{base}$ e $commit_{merge}$ correspondem aos *commits* C_{10} e C_{16}, respectivamente. Portanto, as ações de *merge* são definidas como:

$$acoes_{merge} = diff(commit_{base}, commit_{merge}) \tag{2.1}$$

Em seguida, é realizado o *diff* entre $commit_{base}$ e $commit_{pai1}$, com o objetivo de obter as ações que foram realizadas no ramo 1. De acordo com a Figura 2.1, o $commit_{pai}$, corresponde aos *commits* C_{14} e C_{15}, que são os últimos *commits* anteriores ao *commit* de *merge*. Logo, as ações do ramo 1 são definidas como:

$$acoes_{ramo1} = diff(commit_{base}, commit_{pai1}) \tag{2.2}$$

Por fim, é realizado o *diff* entre $commit_{base}$ e $commit_{pai2}$ para obter as ações que foram realizadas no ramo 2. As ações do ramo 2 são definidas como

$$acoes_{ramo2} = diff(commit_{base}, commit_{pai2}) \tag{2.3}$$

A partir dessas ações, são identificadas as ações que representam os esforços de retrabalho, trabalho desperdiçado e trabalho extra. Para o cálculo do retrabalho é realizada a interseção entre as ações realizadas nos ramos. O retrabalho é definido formalmente como:

$$acoes_{retrabalho} = acoes_{ramo1} \cap acoes_{ramo2}) \tag{2.4}$$

Para determinar o trabalho desperdiçado e o trabalho extra, primeiramente, é necessário identificar as ações que foram realizadas em algum dos ramos. Por isso, as ações dos ramos são definidas como:

$$acoes_{ramos} = acoes_{ramo1} + acoes_{ramo2} \tag{2.5}$$

Para calcular o esforço de trabalho desperdiçado, é utilizado o complemento relativo das ações de *merge* nas ações dos ramos, definido como:

$$acoes_{retrabalho} = acoes_{ramos} \backslash acoes_{merge} \tag{2.6}$$

Por fim, para calcular o esforço do trabalho extra é utilizado o complemento relativo das ações dos ramos nas ações de *merge*, o trabalho extra é definido como:

$$acoes_{extra} = acoes_{merge} \backslash acoes_{ramos} \tag{2.7}$$

A Figura 2.3 ilustra a resolução de um conflito de *merge*. As linhas destacadas em verde claro sinalizam as alterações realizadas nos ramos e a linha destacada em verde escuro, sinaliza a adição da linha para resolver o conflito. Neste caso, o retrabalho ocorre na sétima linha dos *commits* c_{14} e c_{15}, onde os desenvolvedores inseriram a linha *"}"* em duplicidade. O trabalho desperdiçado é representado pelas linhas *"if (name.length () == 0) {"*, *"System.out.println ("invalid subject name!");"*, e *"}"*. Essas três linhas foram inseridas durante o trabalho paralelo, mas durante o *merge*, o desenvolvedor não as utilizou. Finalmente, o trabalho extra é representado pela linha *"System.out.println (LOG: invalid subject name!)"*, porque a linha foi inserida durante o *merge*. Então, para este exemplo específico, existe uma ação de retrabalho, três ações de trabalho desperdiçado e uma ação de trabalho extra.

Figura 2.3: Exemplo de *merge* do método *getName()* da classe *Course*.

2.3 Acoplamento de *software*

Acoplamento de *software* é uma medida que visa identificar o grau de interdependência entre as entidades (por exemplo, pacotes, classes ou métodos) de um sistema de *software* [12]. A análise do acoplamento é útil para diversas atividades de desenvolvimento e gerenciamento, como prever o esforço de manutenção [11], entender o programa [19], prever a suscetibilidade à falha [56] e prever a propagação de erros [23]. Assim, o alto acoplamento não é desejável, pois pode aumentar os custos de manutenção e comprometer a reutilização [44].

A literatura descreve diferentes técnicas de acoplamento de *software*: a dinâmica e a estática. A análise estática envolve a coleta dos acoplamentos estrutural [16], lógico [18, 47] e conceitual [42, 56]. A análise dinâmica do código-fonte consiste na execução do código para detectar as dependências [21]. Esta técnica é mais utilizada para detectar as dependências de código de linguagens dinamicamente tipadas como *Python* [40], onde a tipagem é verificada durante o tempo de execução do *script*. Nesta dissertação, somente o acoplamento estático é analisado e são detalhados os três tipos de acoplamento: estrutural, lógico e conceitual.

2.3.1 Acoplamento Estrutural

O acoplamento estrutural, também conhecido como acoplamento sintático [38, 14], captura as dependências sintáticas entre entidades de *software* e é medido através da análise estática do código-fonte. Por isso, depende da linguagem em que o código foi escrito [38].

No acoplamento estrutural, a dependência ocorre quando, por exemplo, um método de uma classe invoca um método de outra classe [20]. O domínio da métrica de acoplamento estrutural é binário (zero ou um), onde um significa que há dependência e zero significa que não há dependência. Este tipo de acoplamento é direcional.

A Figura 2.4 apresenta como a identificação da dependência é realizada no acoplamento estrutural. Por exemplo, o método *listEnrollments()* depende do método *getGrade()*, porque *listEnrollments()* invoca *getGrade()*. No entanto, o método *getGrade()* não depende do método *listEnrollments()*, porque *getGrade()* não invoca *listEnrollments()*. Então, o valor do acoplamento estrutural para *listEnrollments()* $\rightarrow$*getGrade()* é um e para *getGrade()* $\rightarrow$ *listEnrollments()* é zero.

```java
public class Course {
  ...
  public void listEnrollments() {
    PrintStream out = System.out;
    for (Enrollment enrollment : enrollments) {
      out.println(enrollment.getStudent());
      out.println(enrollment.getGrade());
    }
  }
  ...
}
```

```java
public class Enrollment {
  ...
  public double getGrade() {
    return grade;
  }
  ...
}
```

Figura 2.4: Dependências estruturais entre os métodos *listEnrollments()* e *getGrade()*.

2.3.2 Acoplamento Lógico

Por outro lado, o acoplamento lógico, também conhecido como *co-change* [4, 36] e como acoplamento evolutivo [26], pode ser obtido por meio da análise do histórico de *commits* em um projeto [59]. Assim, as entidades que são frequentemente alteradas juntas indicam um possível acoplamento lógico entre elas.

O acoplamento lógico também é direcional e pode ser identificado através da mineração de regras de associação, por meio das métricas de suporte e confiança. Uma regra de associação é um par *(X, Y)* de dois conjuntos disjuntos. Na notação $X \rightarrow Y$, X é chamado de antecedente e Y é chamado de consequente [1]. Isso significa que, quando X ocorre, Y também ocorre, mesmo que não sejam estruturalmente relacionados [38].

O valor de suporte de um acoplamento refere-se à quantidade de vezes que duas entidades acopladas foram alteradas juntas. Por outro lado, o valor de confiança de um acoplamento normaliza o suporte pelo número total de mudanças de uma das entidades e é uma medida da intensidade de sua co-evolução. Além disso, a métrica de confiança consiste na probabilidade condicional de ter o consequente da regra, considerando que o antecedente está presente. Enquanto o suporte é uma métrica simétrica, a confiança é assimétrica devido à sua normalização. Assim, para obter a direção da dependência lógica, Zimmermann et al. [59] sugere o uso da métrica de confiança. O domínio do acoplamento lógico possui valores entre zero e um, representando a força do acoplamento,

em que um significa que sempre que uma entidade é modificada, a outra entidade também é modificada. Além disso, o acoplamento lógico refere-se às dependências evolutivas ou de mudança entre as entidades de *software* que são frequentemente alteradas juntas [25, 38].

Por exemplo, a Figura 2.5 mostra a frequência em que os métodos foram alterados juntos ao longo do histórico do sistema. Com isso, entre as nove modificações que afetaram o método *listEnrollments()*, quatro também afetaram o método *getGrade()*. Portanto, a confiança da regra *listEnrollments ()* $\rightarrow$ *getGrade ()* é de 0,44 ($4 \div 9$), o que significa que 44 % dos *commits* que mudaram *listEnrollments()* também mudaram *getGrade()*. Por isso, *getGrade()* é 44 % acoplado a *listEnrollments()* [1]. Por outro lado, entre as quatro modificações que afetaram o método *getGrade()*, todas elas também afetaram o método *listEnrollments()*. Portanto, *listEnrollments()* é 100 % ($4 \div 4$) acoplado a *getGrade()*.

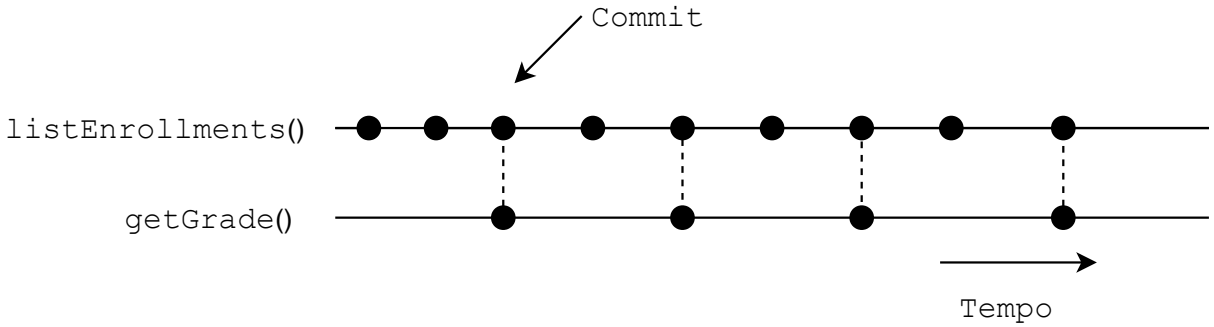


Figura 2.5: Dependências lógicas entre os métodos *listEnrollments()* e *getGrade()*.

2.3.3 Acoplamento Conceitual

Alguns autores [58, 5] definem o acoplamento conceitual [25, 56, 19] como dependência escondida (do inglês, *hidden dependency*) ou implícita no código-fonte. No acoplamento conceitual, também conhecido como semântico [4, 6], a dependência entre os métodos ocorre quando um método possui termos semelhantes aos termos de outro método. Esse tipo de acoplamento utiliza técnicas de recuperação de informações para extrair as informações semânticas presentes nos comentários e nos identificadores do código-fonte [42]. Além disso, identificadores utilizados pelos desenvolvedores no código-fonte para nomes de classes, métodos ou atributos, contêm informações importantes e representam aproximadamente metade do código-fonte no *software* [25]. Assim, quanto maior a similaridade

[1] Observe que a confiança de $X \rightarrow Y$ deve ser interpretada como o quão fortemente Y é logicamente acoplado a X, pois indica a porcentagem de alterações em X que também exigiram alterações em Y

de informações entre duas entidades (métodos), maior é o acoplamento conceitual entre elas.

Diferentemente do acoplamento estrutural e lógico, o acoplamento conceitual é simétrico e sua métrica possui valores entre zero e um [25]. Segundo Poshyvanyk et al. [43], o acoplamento conceitual é definido como alto quando o valor está próximo de um e baixo quando está próximo de zero. A identificação do acoplamento conceitual consiste em calcular a frequência dos termos em cada método e, em seguida, comparar as frequências entre esses métodos.

Na Figura, 2.4 os métodos *getGrade()* e *listEnrollments()* possuem dois termos semelhantes: *enrollment* e *grade*. Portanto, o valor conceitual do acoplamento do par de métodos *listEnrollments()* e *getGrade()* é 0,5 e esse cálculo é descrito em mais detalhes no Capítulo 3.

2.4 Trabalhos Relacionados

As métricas de acoplamento de *software* têm sido estudadas há muito tempo. Na literatura, alguns estudos abordam a interação entre dois tipos de métricas de acoplamento, como estrutural e lógico [3, 38], e conceitual e lógico [4], onde identificam se um tipo de acoplamento leva a outro tipo de acoplamento.

Oliva e Gerosa [38] analisaram 150 mil *commits* do repositório da *Apache Software Foundation* para investigar a proporção de acoplamentos lógicos que envolvem elementos não estruturalmente relacionados e a proporção de elementos com acoplamentos estruturais que envolvem elementos não relacionados logicamente. Eles concluíram que 91% de todos os acoplamentos lógicos estabelecidos envolvem artefatos não estruturalmente relacionados e os acoplamentos estruturais não levam a acoplamentos lógicos, mas há evidências de que pares de classes estruturalmente acopladas, geralmente incluem acoplamentos lógicos.

Da mesma forma, Ajienka e Capiluppi [3] analisaram 79 projetos de *software* de código aberto de diferentes tamanhos para investigar a sobreposição e interseção entre o acoplamento lógico e estrutural entre as classes. Os autores observaram uma pequena sobreposição entre os acoplamentos estrutural e lógico na maioria dos projetos de *software*. Além disso, os autores realizaram a correlação entre dois vetores, um com valores de acoplamento estrutural entre as classes e outro, com valores de acoplamento lógico entre as classes, e então, identificaram que não há correlação significativa entre os acoplamentos

estrutural e lógico entre as classes.

Em um trabalho mais recente, Ajienka, Capiluppi e Counsell [4] investigaram a interação entre os acoplamentos conceitual e lógico em 79 projetos de *software* de código aberto. Eles observaram uma relação bidirecional entre os acoplamentos conceitual e lógico entre as classes de *software*. Os autores concluíram que 70% das classes que são conceitualmente acopladas também são logicamente acopladas e as classes que são logicamente acopladas geralmente compartilham algum grau de acoplamento conceitual.

Costa et al. [15] propôs um estudo para recomendar desenvolvedores que são mais adequados para realizar *merges*, considerando as mudanças dos desenvolvedores nos ramos e o acoplamento lógico entre os arquivos modificados entre os ramos. Eles desenvolveram uma ferramenta chamada *TIPMerge* que identifica os arquivos-chave, ou seja, arquivos que são alterados em paralelo nos ramos ou arquivos que foram alterados em um ramo e estão associados a arquivos alterados em outro ramo. Além disso, ele identifica os desenvolvedores que fizeram alterações nos arquivos-chave em cada ramo para calcular a experiência geral dos desenvolvedores com os arquivos-chave, com base no histórico do projeto. Depois de analisar essas informações, *TIPMerge* recomenda uma lista classificada de desenvolvedores que são mais adequados para integrar um par de ramos. Embora este estudo considere o acoplamento lógico entre ramos, ele não considera os acoplamentos estrutural e conceitual, nem analisa a correlação entre acoplamentos de *software* entre ramos e esforços de *merge*.

Alguns estudos recentes na literatura analisaram diferentes perspectivas que podem influenciar os conflitos de *merge*. Menezes et al. [34] examinaram manualmente o histórico de cinco projetos Java de código aberto e analisaram automaticamente 2.731 projetos Java de código aberto. Para cada *merge* com falha, eles coletaram o número e o tamanho das regiões de conflito, as construções de linguagem contidas nestas regiões e a maneira como os desenvolvedores as resolveram. Os autores procuraram por padrões entre as características das regiões de conflito e as decisões tomadas pelos desenvolvedores, para resolvê-las. Com base nas decisões dos desenvolvedores para resolver conflitos, os autores propuseram uma classificação de seis maneiras para resolver uma parte conflitante. Eles concluíram que os conflitos de *merge* variam muito em complexidade e resolução; e é provável que não seja possível ter uma ferramenta automatizada que possa resolver todos os tipos de conflitos. Os autores realizaram uma ampla avaliação no *merge*, no entanto, eles não compararam se algum acoplamento de *software* está correlacionado com qualquer esforço de *merge*.

Leßenich et al. [31] propuseram sete indicadores para identificar conflitos de *merge*. Por exemplo, com base no número de *commits* em um ramo, o número de nós AST modificados, o número de mudanças realizadas dentro das declarações da classe ou acima das declarações da classe, e o número de arquivos alterados por ambos os ramos. No entanto, com base nas suas conclusões, nenhum dos sete indicadores tem uma correlação forte com o número de conflitos de *merge*.

Kerzazi e Khomh [28] analisaram os fatores que têm um impacto significativo no tempo de espera e nos resultados dos lançamentos de *software*. Eles identificaram que o *merge* pode ser um fator significativo, e a complexidade das mudanças em termos do número de acoplamentos pode ajudar na estimativa do esforço para realizar o *merge*. No entanto, eles não analisaram se há correlação entre acoplamentos e esforços de *merge*.

Baseado na percepção dos desenvolvedores, McKee et al. [33] mostram que os desenvolvedores consideram o número de arquivos conflitantes, linhas conflitantes e o tamanho das alterações como fatores importantes para a dificuldade de resolução. Este trabalho usa essas métricas para quantificar a dificuldade de resolver um conflito de *merge*. Eles descobriram que a complexidade das linhas conflitantes e do arquivo como um todo, o número de linhas de código envolvidas no conflito e a falta de conhecimento dos desenvolvedores nas linhas de código em conflito, impactam na dificuldade dos desenvolvedores para resolver um conflito. Santos e Murta [50] abordam a análise da correlação entre algumas métricas de *merge* de ramos e o esforço de *merge* de trabalho extra. Foram analisados oito ramos de quatro projetos. As métricas de *merge* incluem: quantidade de artefatos modificados nos ramos, quantidade de linhas diferentes nos ramos, *precision* e *recall* entre artefatos alterados nos ramos, quantidade de artefatos modificados em comum e quantidade de conflitos físicos. Com base em seus resultados, a quantidade de artefatos modificados em comum e a quantidade de conflitos físicos são correlacionados com o esforço de *merge* de trabalho extra, com $\rho = 0,62$ e $\rho = 0,99$, respectivamente. Embora esses resultados pareçam ser interessantes, eles não consideram os conflitos indiretos. Além disso, seu *corpus* sofre uma ameaça considerável à validade externa devido ao seu tamanho.

Portanto, nenhum trabalho aborda a análise de acoplamento entre ramos para identificar a correlação entre acoplamentos de *software* e esforços de *merge*.

2.5 Considerações Finais

Neste capítulo, são apresentados conceitos fundamentais para a compreensão desta pesquisa, incluindo as principais características dos conflitos de *merge*, dos esforços de *merge* e dos acoplamentos de *software*. Descreve-se também que o desenvolvimento paralelo é uma prática comum em grandes equipes de desenvolvimento de *software* e o uso de ramos é interessante para isolar o trabalho dessas equipes. Foram discutidos os conceitos sobre o processo de *merge* e os conflitos que podem surgir durante o *merge*: conflitos diretos e indiretos. Além disso, são descritas as técnicas de *merge*: *two-way* e *three-way*, sendo que a última é a mais adequada, pois leva em consideração o ancestral comum de duas versões para decidir o que deve ser mantido na versão resultante do *merge*.

Ressalta-se que os conflitos indiretos podem ser causados pela existência de acoplamento de *software* entre os artefatos envolvidos no *merge*. Como também, são discutidos os conceitos sobre os acoplamentos de *software*: estrutural, lógico e conceitual e que alguns autores os conhecem como sintático, evolutivo e semântico, respectivamente.

Por fim, os trabalhos relacionados mostram que existe uma pequena sobreposição entre os acoplamentos estrutural e lógico e nenhuma correlação significativa entre esses dois tipos de acoplamentos entre classes [3]. Como também, os acoplamentos estruturais não levam a acoplamentos lógicos [38]. Cabe ressaltar, que existe uma relação bidirecional entre os acoplamentos conceitual e lógico entre as classes [4].

Além disso, os trabalhos relacionados mostram que o acoplamento lógico entre ramos já foi utilizado, sendo que para recomendar desenvolvedores para realizar o *merge* [15]. Outro estudo [34] mostra que os conflitos de *merge* variam muito em complexidade e resolução, enquanto um outro trabalho [28] mostra que a complexidade das mudanças referente ao número de acoplamentos, pode ajudar na estimativa do esforço para realizar o *merge*. Considerando indicadores para conflitos de *merge*, constatou-se que nenhum deles possui uma correlação forte com o número de conflitos de *merge*[31]. Além do mais, a quantidade de artefatos modificados em comum e a quantidade de conflitos físicos são correlacionados com o esforço de *merge* de trabalho extra [50]. Por fim, um estudo [33] descobriu que a complexidade das linhas conflitantes, o número de linhas de código envolvidas no conflito e a falta de conhecimento dos desenvolvedores nas linhas de código em conflito impactam na dificuldade de resolver um conflito de *merge*.

Capítulo 3

Acoplamento entre Ramos

3.1 Introdução

Este capítulo apresenta um cenário de exemplo e aborda conceitos relevantes sobre acoplamento de *software* e esforço de *merge*, sempre revisitando o cenário de exemplo para contextualizar e explicar os conceitos.

3.2 Cenário de Exemplo

Este cenário de exemplo é baseado em um sistema de gerenciamento acadêmico que possui um histórico inicial de dez *commits*, nos quais as classes *Student*, *Enrollment*, *Course* e *Subject* foram codificadas. A Figura 3.1 apresenta as classes e seus métodos mais importantes implementados até o *commit* c_{10}, e a Tabela 3.1 mostra o histórico de *commits* de c_1 a c_{10}, com os métodos alterados por cada *commit*. Por exemplo, pode-se notar que os *commits* c_3 e c_5 alteraram todos os métodos listados, enquanto o *commit* c_1 mudou apenas o método *getStudentId()*. Como os nomes dos métodos são únicos, os nomes das classes foram omitidos na Tabela 3.1.

Tabela 3.1: Histórico de *commits* detalhado do c_1 ao c_{10}, mostrando os métodos modificados por cada *commit*.

Métodos	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}
getStudentId()	x		x	x	x	x	x			
getName()			x		x	x	x		x	x
listEnrollments()		x	x	x	x	x	x	x	x	x
Enrollment()			x	x	x	x				
getGrade()			x		x		x		x	
Course()		x	x	x	x	x	x	x		

```java
public class Student {
  private long studentId;
  private String name;
  ...
  public long getStudentId() {
    ...
  }
  ...
}
```

```java
public class Enrollment {
  private Student student;
  private Course course;
  private int grade;
  public Enrollment(Student s, Course c) {
    student = s;
    course = c;
    ...
  }
  public int getGrade() {
    return grade;
  }
  ...
}
```

```java
public class Course {
  private Subject subject;
  private List<Enrollment> enrollments;
  ...
  public Course(Subject s) {
    ...
  }
  public void listEnrollments() {
    ...
  }
  ...
}
```

```java
public class Subject {
  private String name;
  ...
  public String getName() {
    ...
  }
  ...
}
```

Figura 3.1: Código-fonte das classes *Student*, *Enrollment*, *Course* e *Subject* no *commit* c_{10}.

Suponha que o *commit* c_{10} tenha sido entregue e implantado e, após algumas semanas, um usuário relatou um *bug*. Como estratégia para corrigir o *bug*, João e Rose decidiram criar um ramo *bugfix* que evoluiu paralelamente ao ramo *master*, como mostra a Figura 2.1.

Rose realizou os *commits* c_{11} e c_{14} no ramo *bugfix*. No *commit* c_{11}, ela editou o método construtor *Enrollment()* para invocar o método *getStudentId()*, da classe *Student*. Então, no *commit* c_{14}, ela modificou o tipo do atributo *grade* e o tipo de retorno do método *getGrade()*, ambos na classe *Enrollment*, de *int* para *double*. Ela modificou também, o método construtor *Course()*, para invocar o método *getName()*, da classe *Subject*, e o método *getName()*, na classe *Subject*, conforme descrito na Figura 3.2.

Concorrentemente, no ramo *master*, João realizou os *commits* c_{12}, c_{13}, e c_{15} para adicionar novas funcionalidades ao sistema. Primeiro, no *commit* c_{12}, João modificou os métodos *getStudentId()*, na classe *Student*, e *listEnrollments()*, na classe *Course*. Então, no *commit* c_{13}, ele modificou o método *getName()*, na classe *Subject*. Finalmente, no *commit* c_{15}, ele editou o método *listEnrolments()*, na classe *Course*, para invocar o método *getGrade()*, da classe *Enrollment*, conforme mostrado na Figura 3.3.

Depois que Rose corrigiu o erro, João integrou os ramos através do *commit* de *merge* c_{16}. No entanto, um conflito surgiu durante o processo de *merge* e João interveio manualmente para resolvê-lo. A Figura 2.3 demonstra as alterações realizadas no método *getName()*, durante o trabalho nos ramos *bugfix* e *master* (linhas destacadas em verde claro), na classe *Subject* e na adição de João para resolver o conflito (linha destacada em verde escuro). Nesta dissertação, os ramos *bugfix* e *master* são chamados de ramo *esquerdo* e ramo *direito*, respectivamente.

3.3 Visão Geral da Abordagem

A detecção de acoplamento geralmente é realizada na versão de um sistema, onde todas as classes de um projeto são analisadas para identificar as dependências entre elas. No entanto, quando os desenvolvedores estão trabalhando em paralelo, o acoplamento entre as alterações realizadas entre os ramos pode ser uma fonte potencial de conflitos indiretos que podem levar a falhas de criação e de teste. Para identificar o acoplamento entre as mudanças entre ramos, optou-se por trabalhar no grão do método, pois é o nível de abstração mais baixo para programas orientados a objetos e, consequentemente, oferece uma maior precisão para as análises. Para tanto, são identificados os arquivos que foram

```java
public class Enrollment {
  private Student student;
  private Course course;
  private double grade;
  public Enrollment(Student s, Course c) {
    student = s;
    course = c;
    long studentId = student.getStudentId();
    ...
  }
  public double getGrade() {
    return grade;
  }
  ...
}
```

```java
public class Course{
  private Subject subject;
  private List<Enrollment> enrollments;
  ...
  public Course(Subject s) {
    subject = s;
    String subjectName = subject.getName();
    ...
  }
  ...
}
```

```java
public class Subject {
  private String name;
  ...
  public String getName() {
    if (name.length() == 0) {
      System.out.println("invalid subject name!");
    }
    return name;
  }
  ...
}
```

Figura 3.2: Modificações realizadas no ramo *bugfix* – as linhas destacadas em verde foram alteradas.

```java
public class Student {
  private long studentId;
  ...
  public long getStudentId() {
    if (studentId < 1) {
      return "invalid student id!";
    }
    return studentId;
  }
  ...
}
```

```java
public class Course {
  private Subject subject;
  private List<Enrollment> enrollments;
  ...
  public void listEnrollments() {
    PrintStream out = System.out;
    for (Enrollment enrollment : enrollments) {
      out.println(enrollment.getStudent());
      out.println(enrollment.getGrade());
    }
  }
  ...
}
```

```java
public class Subject {
  private String name;
  ...
  public String getName() {
    if (name.isEmpty()) {
      return "invalid subject name!";
    }
    return name;
  }
  ...
}
```

Figura 3.3: Mudanças realizadas no ramo *master* – as linhas destacadas em verde foram alteradas.

modificados em cada ramo e, em seguida, os métodos específicos que foram alterados. Finalmente, é analisado se as alterações nos métodos de um ramo têm dependências com as alterações nos métodos do outro ramo.

Nesta seção, primeiramente, o problema é formalizado utilizando-se a teoria dos conjuntos (Seção 3.3.1). Em seguida, é explicado o processo de extração dos dados do histórico das versões dos repositórios (Seção 3.3.2) e como é realizada a medição dos acoplamentos estrutural, lógico e conceitual entre os métodos alterados entre os ramos (Seção 3.3.3). Por fim, apresenta-se as duas métricas propostas que resumem os acoplamentos entre ramos (Seções 3.3.4 e 3.3.5).

3.3.1 Formalização do Problema

Com base na formalização introduzida por Costa et al. [15] um projeto p é definido como uma tupla (M, C), em que M é um conjunto de métodos e C é um conjunto de *commits*. Cada *commit* $c_i \in C$ é uma tupla (M_i, P_i), onde $M_i \subseteq M$ é o conjunto de métodos alterados (adicionados, removidos ou editado) por c_i e $P_i \subset C$ é o conjunto de *commits* pais de c_i.

Os *commits* são organizados em um gráfico acíclico direcionado, como no exemplo mostrado na Figura 2.1, em que c_1 é o primeiro *commit* do projeto e não tem pai. Os outros *commits* têm pelo menos um pai (por exemplo, c_{12} é o pai de c_{13}) e o *commit* de *merge* c_{16} tem dois pais: c_{14} e c_{15}. Além disso, c_{10} é o *merge-base* que corresponde ao ancestral comum entre os *commits* c_{14} e c_{15}. Todos os *commits* acessíveis a partir de c_i formam seu histórico, incluindo o próprio c_i e o fecho transitivo sobre seus pais. Por exemplo, $\{c_1, ..., c_{10}, c_{11}, c_{14}\}$ é o histórico de *commits* c_{14}. Deste modo, o histórico de um *commit* $c_i \in C$ é definido como:

$$H_i = \{c_i\} \cup \bigcup_{p_j \in P_i} H_j \tag{3.1}$$

Dois *commits* $c_i, c_j \in C$ que não se alcançam (por exemplo, $c_i \notin H_j \wedge c_j \notin H_i$) são chamados de variantes (por exemplo, *commit* c_{14} e c_{15} na Figura 2.1). As variantes podem ter uma história comum, que compreende todos os *commits* existentes nas duas histórias. Por exemplo, $\{c_1, ..., c_{10}\}$ é o histórico comum de *commits* c_{14} e c_{15}. O histórico comum de *commits* $c_i, c_j \in C$ é definido como:

$$CH_{i,j} = H_i \cap H_j \tag{3.2}$$

O histórico de cada variante também inclui *commits* que não pertencem ao histórico comum, formando uma linha de desenvolvimento independente chamada histórico do ramo. Por exemplo, na Figura 2.1, $\{c_{11}, c_{14}\}$ é o histórico do ramo de c_{14} ao realizar o *merge* com c_{15} e $\{c_{12}, c_{13}, c_{15}\}$ é o histórico do ramo de c_{15} ao realizar o *merge* com c_{14}. Como os ramos podem ser criados a partir de outros ramos, o histórico do ramo pode variar dependendo do ramo oposto, como consequência de diferentes históricos comuns. O histórico do ramo de $c_i \in C$ ao realizar o *merge* com $c_j \in C$ é definido como:

$$HR_{i,j} = H_i \setminus H_j \tag{3.3}$$

Cada histórico do ramo compreende a um conjunto de métodos alterados por seus *commits*. Os métodos alterados no histórico do ramo de $c_i \in C$ ao realizar o *merge* com $c_j \in C$ são definidos como:

$$M_{i,j} = \bigcup_{c_k \in HR_{i,j}} M_k \tag{3.4}$$

De acordo com o exemplo, os métodos alterados no *ramo esquerdo* correspondem à união dos métodos modificados nos *commits* c_{11} (*Enrollment()*) e c_{14} (*Course()*, *getGrade()* e *getName()*). Portanto, os métodos alterados no histórico do *ramo esquerdo* ($M_{14,15}$) são: *Enrollment()*, *Course()*, *getGrade()*, e *getName()*. Além disso, os métodos alterados no *ramo direito* correspondem à união dos métodos modificados nos *commits* c_{12} (*getStudentId()*), c_{13} (*getName()*) e c_{15} (*listEnrollments()*). Portanto, os métodos alterados no histórico do *ramo direito* ($M_{15,14}$) são: *getStudentId()*, *getName()* e *listEnrollments()*.

O acoplamento é extraído da dependência entre os métodos modificados nos ramos integrados. Durante a extração do acoplamento, alguns *commits* são importantes e recebem nomes especiais: os *tips* e o *merge-base*. O **tip** é o último *commit* de um ramo e o **merge-base** é o *commit* mais recente no histórico comum de dois *tips*. Por exemplo, c_{14} é o *tip* do *ramo esquerdo*, c_{15} é o *tip* do *ramo direito* e c_{10} é *merge-base*. Por uma questão de generalidade, eles são chamados respectivamente de c_e, c_d e c_b. Como resultado, $M_{e,d}$ é o conjunto de métodos alterados no ramo esquerdo (ou seja, métodos alterados entre c_b e c_e) e $M_{d,e}$ é o conjunto de métodos alterados no ramo direito (ou seja, métodos alterados

entre c_b e c_d).

Dados os *ramos esquerdo* e *direito*, os possíveis acoplamentos entre os ramos são calculados considerando o produto cartesiano entre os métodos em $M_{e,d}$ e os métodos em $M_{d,e}$. Considerando o cenário de exemplo, $M_{e,d}$ possui quatro métodos modificados e $M_{d,e}$ possui três métodos modificados, resultando em 12 (4×3) pares no produto cartesiano. Assim, considerando um acoplamento simétrico (por exemplo, acoplamento conceitual), dois métodos têm o mesmo valor de acoplamento em ambas as direções, levando a 12 valores possíveis. Por outro lado, considerando um acoplamento assimétrico (por exemplo, acoplamento estrutural e lógico), cada par pode ter valores diferentes em cada direção (por exemplo, *getGrade()* → *listEnrollments()* e *listEnrollments()* → *getGrade()*), levando a 24 possíveis valores.

3.3.2 Extração dos Dados

Nesta seção, é explicado como são processados os dados dos VCS para identificar os métodos alterados (ou seja, $M_{e,d}$ e $M_{d,e}$) que são utilizados para calcular o acoplamento entre ramos. A Figura 3.4 mostra as etapas comuns que são realizadas: (1) identificação das alterações nos artefatos de *software*, (2) extração da árvore de sintaxe abstrata (AST) e (3) identificação das declarações e das invocações de método alteradas nos ramos *esquerdo* e *direito*.

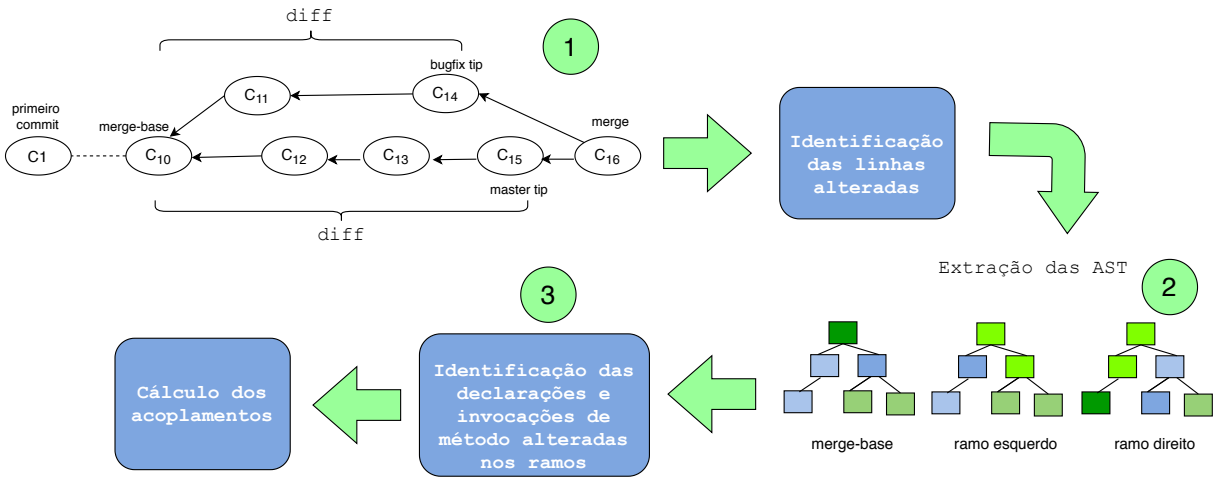


Figura 3.4: Etapas da extração dos dados para o cálculo dos acoplamentos entre ramos.

O primeiro passo identifica as linhas alteradas nos ramos *esquerdo* e *direito*. Esta identificação é possível utilizando ferramentas *diff*. Por exemplo, considerando a classe *Course* nos *commits* c_b e c_d, uma ferramenta *diff* indicaria que a linha nove foi adicionada. Vale a pena mencionar que as ferramentas *diff* utilizadas em VCS, como o Git,

reconhecem mudanças utilizando granularidade de linha. Assim, eles não sabem que uma linha pertence a uma declaração de método.

A segunda etapa consiste em extrair a AST de c_b, c_e e c_d para identificar os métodos modificados. De agora em diante, chama-se AST_i de AST extraída de um determinado *commit* c_i. Portanto, AST_b corresponde à AST extraída do *merge-base*, e AST_e e AST_d correspondem às ASTs extraídas dos ramos *esquerdo* e *direito*, respectivamente. As AST_e e AST_d são capazes de fornecer informações sobre métodos que ainda estão no código após as modificações. Para identificar informações sobre métodos removidos, é verificado se um determinado método pertence à AST_b, mas não pertence mais à AST_e ou AST_d. Desta forma, foi identificado um possível acoplamento entre uma invocação de método adicionada no *ramo esquerdo* (AST_e) e uma declaração de método removida do *ramo direito* (AST_d) ou vice-versa.

A terceira etapa consiste em identificar os métodos alterados nas classes. Esta etapa utiliza os resultados da primeira etapa, que obtém as linhas alteradas, e da segunda etapa, que extrai as ASTs, para identificar as declarações e invocações de método que foram afetadas pelas linhas alteradas. Por exemplo, no *ramo esquerdo*, a ferramenta *diff* identificou que Rose alterou a décima primeira linha da classe *Enrollment* (ver Figura 3.2). Então, consultando a AST_e, é possível identificar que a décima primeira linha representa o início da declaração do método *getGrade()*. Assim, a terceira etapa identifica que *getGrade()* foi alterado. Este processo é realizado para todas as alterações nos ramos *esquerdo* e *direito* para identificar mudanças nas declarações de método, conforme descrito, e invocações de método. Como consequência, no cenário de exemplo, as alterações no *ramo esquerdo* (veja a Figura 3.2) afetou as declarações dos métodos *Enrollment()*, *getGrade()*, *Course()* e *getName()*, e as invocações dos métodos *getStudentId()*, *getName()*, *length()*, e *println()*. Por outro lado, as mudanças no *ramo direito* (ver Figura 3.3) afetou as declarações dos métodos *getStudentId()*, *listEnrollments()* e *getName()*, e as invocações dos métodos *getGrade()*, *println()* e *isEmpty()*. Finalmente, as invocações relacionadas aos métodos declarados em bibliotecas externas são excluídas de nossa análise, porque sua declaração não estará sujeita a alterações. Como consequência, as invocações dos métodos *println()*, *length()* e *isEmpty()* são excluídas do conjunto final de métodos alterados.

Durante a evolução do software, o número de linhas onde uma declaração ou invocação de método está localizada pode mudar devido às linhas adicionadas ou removidas. Assim, para comparar o conteúdo em diferentes versões (por exemplo, c_b, c_e e c_d), aplica-se uma etapa de pré-processamento para ajustar os números das linhas. Como exemplo, o bloco

de código do método *getGrade()* em c_b corresponde ao intervalo da linha 10 a 12. Ao inserir as oitava e nona linhas no método *Enrollment* (veja a Figura 3.2), o bloco de código do método *getGrade()* mudou para o intervalo da linha 11 para 13. Por causa disso, para relacionar as linhas modificadas em c_l com as declarações do método em c_b, mantém-se um mapa das linhas modificadas em cada versão. Assim, pode-se identificar que a décima primeira linha modificada em c_e corresponde à oitava linha do método *getGrade()* em c_b.

3.3.3 Medição de acoplamento entre ramos

Após identificar as linhas e os métodos modificados em ambos os ramos, são executadas diferentes etapas para calcular cada métrica de acoplamento entre ramos, que são descritas nesta seção.

Para extrair o **acoplamento estrutural**, é analisado se uma declaração de método modificado no *ramo esquerdo* tem uma invocação adicionada ou alterada no *ramo direito* ou vice versa. Utiliza-se o mecanismo de *binding* disponível nas ASTs, o qual cria identificadores únicos para declaração e invocação de métodos, possibilitando a comparação mesmo quando estão em ASTs diferentes. Conforme discutido na Seção 3.3.2, a AST_b possui os métodos que são comuns a ambos os ramos e as informações sobre os métodos que foram removidos dos ramos. Assim, para calcular o acoplamento, é verificado se os *bindings* da declaração do método do *ramo esquerdo* (em AST_b) e da invocação do método do *ramo direito* (em AST_d) são iguais. Se os *bindings* corresponderem, é indicado um acoplamento estrutural entre os métodos dos ramos. Da mesma forma, é verificado se os *bindings* da declaração do método do *ramo direito* (em AST_b) e da invocação do método do *ramo esquerdo* (em AST_e) são iguais. A Figura 3.5 ilustra as comparações que são realizadas entre os *bindings* das ASTs. Como o acoplamento estrutural é assimétrico, o acoplamento é medido nas duas direções. Uma medida indica a intensidade do acoplamento estrutural do *ramo esquerdo* ao *ramo direito*. A outra medida indica a intensidade do acoplamento estrutural do *ramo direito* ao *ramo esquerdo*. Semelhante à medição tradicional do acoplamento estrutural, 1 (um) significa a existência de dependência e 0 (zero) significa a ausência de dependência. Observe que a identificação das invocações dos métodos são necessárias apenas para o cálculo do acoplamento estrutural. As outras métricas de acoplamento não utilizam essas informações.

De acordo com o cenário de exemplo, a Tabela 3.2 mostra a intensidade do acoplamento estrutural entre os métodos modificados nos ramos. A coluna $\mathbf{E} \rightarrow \mathbf{D}$ mostra os métodos da coluna da esquerda que estão acoplados aos métodos da coluna da direita. A

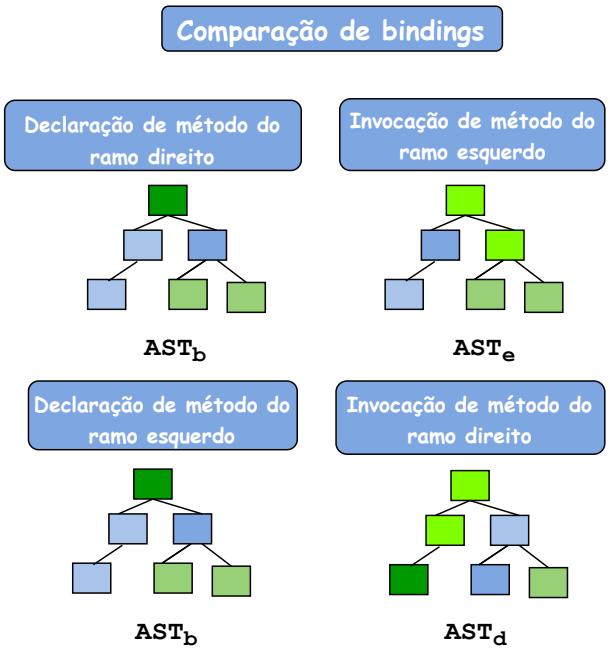


Figura 3.5: Comparação dos *bindings* para identificação do acoplamento estrutural.

coluna **D** $\rightarrow$ **E** mostra os métodos da coluna da direita que estão acoplados com o método da coluna da esquerda. A coluna **Média** representa o acoplamento médio considerando ambas as direções. Observando a Tabela 3.2, é possível ver que *Enrollment()* depende de *getStudentId()*, pois o método *Enrollment()*, modificado no *ramo esquerdo*, invoca o método *getStudentId()*, modificado no *ramo direito*. Contudo, o método *getStudentId()* não depende do método *Enrollment()*, pois o método *getStudentId()* não invoca o método *Enrollment()*.

Para extrair o **acoplamento lógico**, é verificado se os pares de métodos modificados nos ramos foram frequentemente co-alterados na história comum. Assim, a confiança é calculada entre os pares de métodos desde o primeiro *commit* do projeto até o *merge-base*.

De acordo com o cenário de exemplo, a Tabela 3.3 mostra os valores de acoplamento entre os métodos modificados nos ramos *esquerdo* e *direito*, considerando a frequência de mudanças nos métodos apresentados na Tabela 3.1. A coluna **E** $\rightarrow$ **D** mostra o acoplamento entre os métodos da coluna da esquerda e os métodos da coluna da direita. A coluna **D** $\rightarrow$ **E** mostra o acoplamento entre os métodos da coluna da direita e os métodos da coluna da esquerda. A coluna **Média** representa o valor médio de **E** $\rightarrow$ **D** e **D** $\rightarrow$ **E**. Como discutido anteriormente, a confiança da regra **X** $\rightarrow$ **Y** indica quão fortemente Y está acoplado a X. Por exemplo, o acoplamento lógico para *getStudentId()* $\rightarrow$ *getGrade()* é 0,75 porque de quatro modificações em *getGrade()*, três também afetaram *getStudentId()* $(3 \div 4 = 0,75)$. Por outro lado, o acoplamento lógico para *getGrade()* $\rightarrow$

Tabela 3.2: Extração do acoplamento estrutural entre os métodos modificados nos ramos esquerdo e direito.

Métodos		**Acoplamento**		**Média**
Esquerda	**Direita**	**E→D**	**D→E**	
Enrollment()	*getStudentId*()	1	0	0.50
Enrollment()	*getName*()	0	0	0.00
Enrollment()	*listEnrollments*()	0	0	0.00
getGrade()	*getStudentId*()	0	0	0.00
getGrade()	*getName*()	0	0	0.00
getGrade()	*listEnrollments*()	0	1	0.50
Course()	*getStudentId*()	0	0	0.00
Course()	*getName*()	1	0	0.50
Course()	*listEnrollments*()	0	0	0.00
getName()	*getStudentId*()	0	0	0.00
getName()	*getName*()	0	0	0.00
getName()	*listEnrollments*()	0	0	0.00

getStudentId() é 0,5 porque de seis modificações em *getStudentId()* no histórico comum, apenas três também afetaram *getGrade()* ($3 \div 6 = 0,5$).

Para os acoplamentos estrutural e lógico, o valor médio é calculado para permitir a comparação entre o acoplamento conceitual, que é simétrico, e os acoplamentos estrutural e lógico, que são assimétricos. Essa métrica fornece uma percepção geral do acoplamento entre dois métodos, independentemente da direção do acoplamento.

Para extrair o **acoplamento conceitual**, são identificadas as linhas alteradas nos métodos modificados nos ramos *esquerdo* e *direito*. Em seguida, os termos das linhas alteradas são extraídos utilizando a técnica descrita por Ajienka e Capiluppi [2], discutida no Capítulo 2.3. Finalmente, é utilizada a similaridade de cosseno entre os métodos para quantificar o acoplamento conceitual de mudanças entre ramos.

Por exemplo, com base nas mudanças nos ramos *esquerdo* e *direito* (ver Figuras 3.2 e 3.3), o pré-processamento das linhas modificadas nos métodos *listEnrollments()* e *getGrade()* gera os seguintes termos, respectivamente: [*"list"*, *"enrol"*, *"enrol"*, *"grade"*] e [*"grade"*, *"grade"*, *"grade"*]. O pré-processamento consiste na eliminação de caracteres não alfanuméricos e palavras irrelevantes, convertendo todas as palavras em minúsculas e transformando algumas palavras em sua forma raiz por meio da técnica de *stemming* [2]. Neste caso, os termos *"enrol"* e *"grade"* pertencem a ambos os métodos, e os termos restantes são exclusivos de cada método. Isso leva a uma similaridade de cosseno de 37% entre esses dois métodos. A tabela 3.4 mostra a intensidade do acoplamento conceitual entre todos os métodos modificados no *ramo esquerdo* e no *ramo direito*. Como essa mé-

Tabela 3.3: Extração do acoplamento lógico entre os métodos modificados nos ramos esquerdo e direito.

Métodos		**Acoplamento**		**Média**
Esquerda	**Direita**	**L->R**	**R->L**	
Enrollment()	*getStudentId*()	1.00	0.66	0.83
Enrollment()	*getName*()	0.75	0.50	0.62
Enrollment()	*listEnrollments*()	1.00	0.44	0.72
getGrade()	*getStudentId*()	0.75	0.50	0.62
getGrade()	*getName*()	1.00	0.66	0.83
getGrade()	*listEnrollments*()	1.00	0.44	0.72
Course()	*getStudentId*()	0.71	0.83	0.77
Course()	*getName*()	0.57	0.66	0.61
Course()	*listEnrollments*()	1.00	0.77	0.88
getName()	*getStudentId*()	0.66	0.66	0.66
getName()	*getName*()	1.00	1.00	1.00
getName()	*listEnrollments*()	1.00	0.66	0.83

trica de acoplamento é simétrica, ela tem o mesmo valor nas duas direções. Por exemplo, a Tabela 3.4 mostra que *Enrollment()* e *getStudentId()* são 94% conceitualmente acoplados, mas *getGrade()* e *getStudentId()* não são conceitualmente acoplados.

Tabela 3.4: Extração do acoplamento conceitual entre os métodos modificados nos ramos esquerdo e direito.

Métodos		**Acoplamento**
Esquerda	**Direita**	
Enrollment()	*getStudentId*()	0.94
Enrollment()	*getName*()	0.00
Enrollment()	*listEnrollments*()	0.20
getGrade()	*getStudentId*()	0.00
getGrade()	*getName*()	0.00
getGrade()	*listEnrollments*()	0.37
Course()	*getStudentId*()	0.00
Course()	*getName*()	0.58
Course()	*listEnrollments*()	0.00
getName()	*getStudentId*()	0.02
getName()	*getName*()	0.91
getName()	*listEnrollments*()	0.00

3.3.4 Acoplamento Total

O Acoplamento Total é uma métrica proposta para resumir os acoplamentos estrutural, lógico e conceitual das mudanças entre ramos, sendo utilizada para calcular a correlação com os esforços de *merge*.

A métrica calcula a intensidade do acoplamento geral de dois ramos através da soma dos valores médios de cada par de métodos modificados entre ramos, em ambas as direções (**E** $\rightarrow$ **D** e **D** $\rightarrow$ **E**). A métrica $Total_{tipo}$ é definida na Equação 3.5, onde $tipo \in \{estrutural, logico, conceitual\}$.

$$Total_{tipo} = \sum_{m_e \in M_{e,d}} \sum_{m_d \in M_{d,e}} \frac{Acoplamento^{tipo}_{m_e,m_d} + Acoplamento^{tipo}_{m_d,m_e}}{2} \tag{3.5}$$

De acordo com o cenário de exemplo, $Total_{estrutural}$ é a soma do acoplamento médio dos doze pares de métodos. De acordo com a Tabela 3.2, este valor é $1,5$. Da mesma forma, $Total_{logico}$ é 9.09 e $Total_{conceitual}$ é $3,02$ [1].

3.3.5 Acoplamento Normalizado

A interpretação da métrica anterior pode ser difícil, pois os valores dependem do número de métodos modificados. Assim, foi definida uma segunda métrica para normalizar os acoplamentos no espaço contínuo, entre zero e um. Essa nova métrica, chamada $Normalizado_{tipo}$, consiste em dividir a métrica $Total_{tipo}$ pelo número de pares no produto cartesiano entre $M_{e,d}$ e $M_{d,e}$, onde $tipo \in \{estrutural, logico, conceitual\}$. Esta métrica é definida na Equação 3.6.

$$Normalizado_{tipo} = \frac{Total_{tipo}}{|M_{e,d}| \times |M_{d,e}|} \tag{3.6}$$

De acordo com o exemplo, foram identificados doze pares no produto cartesiano entre os métodos alterados no *ramo esquerdo* e no *ramo direito* (ver Tabela 3.4). Assim, a métrica $Normalizado_{estrutural}$ é $1,5 \div 12 = 0,12$, a métrica $Normalizado_{logico}$ é $9,09 \div 12 = 0,76$ e a métrica $Normalizado_{conceitual}$ é $3,47 \div 12 = 0,29$.

3.4 Considerações Finais

Neste capítulo, foi apresentado um cenário de exemplo baseado em um sistema de gerenciamento acadêmico para exemplificar o cálculo dos acoplamentos entre ramos. Foram mostradas as etapas do processamento dos dados extraídos dos VCS. De forma geral, as etapas incluem a identificação das linhas modificadas, a extração das AST e a identificação

[1] Uma vez que o acoplamento conceitual é simétrico, $Acoplamento^{conceitual}_{m_e,m_d} = Acoplamento^{conceitual}_{m_d,m_e} = (Acoplamento^{conceitual}_{m_e,m_d} + Acoplamento^{conceitual}_{m_d,m_e})/2$.

dos métodos que foram alterados nos ramos.

Além disso, foram apresentadas as etapas necessárias para o cálculo de cada tipo de acoplamento. Para o acoplamento estrutural, é analisado se uma declaração de método modificado no *ramo esquerdo* possui uma invocação adicionada ou alterada no *ramo direito* ou vice versa. Para o acoplamento lógico, é verificado se os pares de métodos modificados nos ramos foram frequentemente co-alterados na história comum e assim, a confiança é calculada entre os pares de métodos desde o primeiro *commit* do projeto até o *merge-base*. Para o acoplamento conceitual, são extraídos os termos das linhas alteradas e é realizado o cálculo da similaridade de cosseno entre os métodos modificados.

Por fim, foram propostas duas métricas, absoluta e normalizada, para o cálculo dos acoplamentos entre ramos. Em virtude dos acoplamentos estrutural e lógico serem assimétricos e o acoplamento conceitual ser simétrico, o valor médio é calculado para os acoplamentos estrutural e lógico, com o objetivo de fornecer uma percepção geral do acoplamento entre dois métodos, independentemente da direção do acoplamento. A métrica $Total_{tipo}$ (i.e., métrica absoluta) calcula a intensidade do acoplamento geral de dois ramos através da soma dos valores médios de cada par de métodos modificados entre ramos, em ambas as direções. A métrica $Normalizado_{tipo}$ (i.e., métrica normalizada), consiste em dividir a métrica $Total_{tipo}$ pelo número de pares no produto cartesiano.

Capítulo 4

Avaliação

4.1 Introdução

Neste capítulo, são apresentados os estudos que foram conduzidos para entender se a existência de acoplamento está relacionada com o esforço de *merge*: análise da correlação e análise da influência dos atributos dos ramos. Na análise da correlação, foram estudadas as correlações entre os acoplamentos e os esforços de *merge* dos doze projetos do corpus. Na análise da influência dos atributos dos ramos, foram estudadas as influências de quatro atributos dos ramos na correlação entre os acoplamentos e os esforços de *merge*. O objetivo é observar o contexto do relacionamento entre acoplamento e esforço de *merge* a fim de identificar se a presença de acoplamento entre ramos aumenta a ocorrência de esforços de *merge*.

A análise da correlação entre as métricas de acoplamento (estrutural, lógico e conceitual) entre ramos e os esforços de *merge* (retrabalho, trabalho desperdiçado e trabalho extra) foi realizada em 6.376 *merges* de doze projetos obtidos do *GitHub*. Foi investigado se a variação do *threshold* dos acoplamentos lógico e conceitual aumenta ou diminui as correlações entre o acoplamento e o esforço de *merge*. Os resultados mostraram que à medida que o *threshold* aumenta, a correlação também aumenta.

A análise da influência dos atributos dos ramos na correlação também foi realizada considerando 6.376 *merges*, sendo que foram extraídos quatro atributos de cada ramo: número de arquivos modificados, número de linhas modificadas, número de *commits* e número de autores. Os *merges* foram segregados em dois grupos: um grupo de *merges* que possuem a média harmônica dos atributos menor que ou igual à mediana e outro grupo de *merges* que possuem a média harmônica dos atributos maior que a mediana.

Para cada grupo, a correlação foi calculada entre os acoplamentos e os esforços de *merge* e os resultados mostraram, no geral, que os grupos que possuem valores maiores para os atributos dos ramos, também possuem maior correlação.

O restante deste capítulo é organizado como segue. A Seção 4.2 apresenta os materiais e métodos da pesquisa. A Seção 4.3 mostra os resultados e as discussões da análise da correlação entre os acoplamentos e os esforços de *merge* e, da análise da influência dos atributos dos ramos na correlação entre acoplamentos e esforços de *merge*, respectivamente. A Seção 4.4 discute as ameaças à validade e a Seção 4.5 apresenta as principais conclusões.

4.2 Materiais e Métodos

Esta seção apresenta as questões de pesquisa, a seleção do *corpus*, as análises que foram realizadas e a implementação das ferramentas que extraem os acoplamentos entre ramos.

4.2.1 Questões de Pesquisa

Foram elaboradas quatro questões de pesquisa que guiarão este estudo. Cada uma delas é descrita e detalhada a seguir, e, as respostas a essas questões são apresentadas na Seção 4.3. São elas:

- QP1: Quais acoplamentos entre ramos se correlacionam com o outro? Nesta questão, foi analisado se há correlação entre os acoplamentos estrutural, lógico e conceitual entre ramos.
- QP2: Quais esforços de *merge* se correlacionam com o outro? Nesta questão, foi observado se há correlação entre os esforços de retrabalho, de trabalho desperdiçado e de trabalho extra.
- QP3: Quais acoplamentos entre ramos se correlacionam com esforço de *merge*? A existência de acoplamento entre artefatos que foram modificados entre ramos pode influenciar a ocorrência de conflitos indiretos [52, 17], e conflitos indiretos podem levar a mais esforço durante o *merge*. Então, nesta questão de pesquisa, pretende-se observar se os acoplamentos estrutural, lógico ou conceitual se correlacionam com os esforços de retrabalho, trabalho desperdiçado ou trabalho extra durante o desenvolvimento paralelo e o *merge*.

- QP4: Quais características influenciam a correlação entre os acoplamentos entre ramos e os esforços de *merge*? Após identificar a correlação entre os acoplamentos e os esforços de *merge* e com o intuito de procurar por padrões que explicam essas correlações, foram extraídas algumas características dos ramos como: o número de arquivos modificados (adicionados, removidos e editados), o número de linhas alteradas (adicionadas, removidas e editadas), o número de *commits* e o número de autores.

Assim, essa questão de pesquisa poder ser decomposta em quatro questões de pesquisa mais específicas listadas a seguir:

- QP4.1: O número de arquivos modificados entre ramos influencia a correlação entre os acoplamentos e os esforços de *merge*? Como discutido na literatura [31], o número de arquivos alterados por ambos os ramos tem uma correlação média com o número de conflitos de *merge*. Com essa questão, pretende-se analisar os efeitos do número de arquivos modificados entre ramos, na correlação entre os acoplamentos e os esforço de *merge*.

- QP4.2: O número de linhas modificadas entre ramos influencia a correlação entre os acoplamentos e os esforços de *merge*? Leßenich et al. [31] indicam que o número de linhas alteradas nos ramos mostra uma correlação média com o número de conflitos diretos de *merge*. Assim, investigamos os efeitos do número de linhas alteradas entre ramos na correlação entre os acoplamentos e os esforços de *merge*.

- QP4.3: O número de *commits* entre ramos influencia a correlação entre os acoplamentos e os esforços de *merge*? Na literatura, um estudo [31] conclui que o número de *commits* possui fraca correlação com o número de conflitos de *merge*. Portanto, com esta questão, pretende-se analisar os efeitos do número de *commits* realizados entre os ramos, na correlação entre os acoplamentos e os esforços de *merge*.

- QP4.4: O número de autores entre ramos influencia a correlação entre os acoplamentos e os esforços de *merge*? Conforme discutido na literatura [39], o número de desenvolvedores ativos no ramo mostra baixa correlação com conflitos de *merge*. Assim, foram investigados os efeitos do número de autores trabalhando nos ramos, na correlação entre acoplamentos e esforços de *merge*.

4.2.2 Seleção do *Corpus*

Primeiramente, foram definidos alguns requisitos para selecionar os projetos, como pode ser visto na Figura 4.1. Os projetos devem conter mais de 5.000 estrelas, mais de 5.000 *commits* e mais de 250 *merges*. O critério das estrelas permite buscar por projetos populares e amplamente utilizados. O critério de *commits* permite buscar por projetos com histórico ativo e duradouro. O critério de *merges* permite o descarte de projetos que possuam uma quantidade de *commits* de *merge* muito maior que a dos demais projetos selecionados, assim como o descarte de projetos que possuem poucos *merges*. Além disso, os projetos devem ter sido desenvolvidos na linguagem de programação Java, considerando que a coleta dos acoplamentos estrutural, lógico e conceitual entre métodos depende da análise sintática, impossibilitando uma implementação independente de linguagem. Dessa maneira, a linguagem Java foi escolhida porque é uma linguagem amplamente utilizada mundialmente [29]. Para obter projetos de acordo com os requisitos definidos, foi utilizado o *GitHub*, visto que ele contém um número significativo de projetos e uma API que permite a consulta.

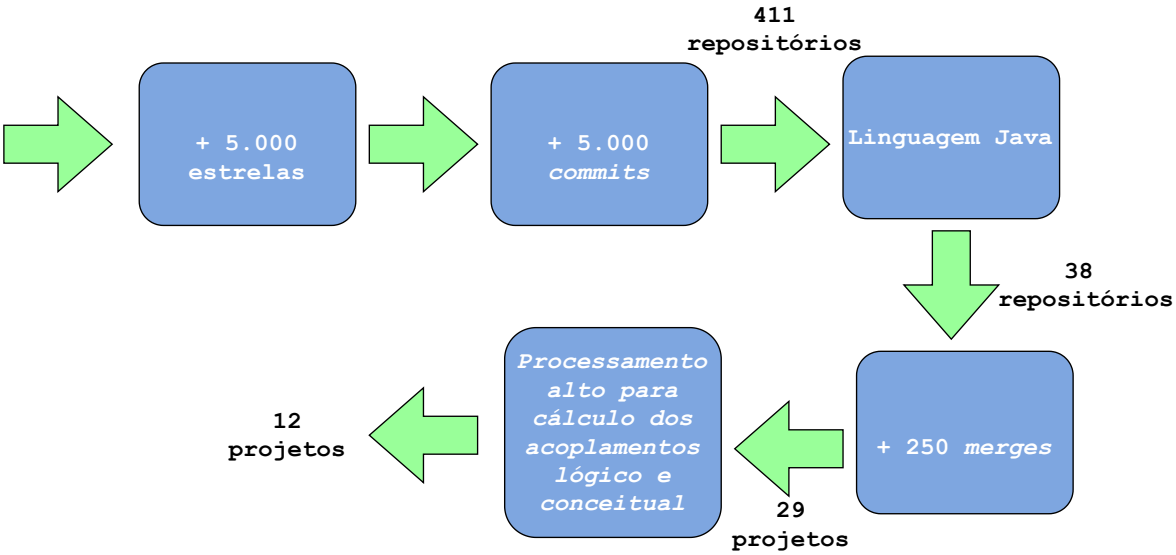


Figura 4.1: Requisitos definidos para a seleção do *corpus*.

De acordo com os critérios definidos, em 7 de abril de 2019, a pesquisa por projetos hospedados no *GitHub* retornou 411 repositórios, sendo que após o filtro da linguagem Java, restaram 38 repositórios. Foram descartados dois repositórios, um por não ser um projeto de *software* e o outro por ser escrito no idioma chinês e, portanto, tornar inviável a confirmação do seu propósito. Aplicou-se um filtro relacionado ao número de *commits* de *merge* e foram descartados os projetos com menos de 250 *merges*, restando 29 projetos. Inicialmente, foram extraídos os acoplamentos dos projetos com um número menor de

Tabela 4.1: Descrição geral dos projetos selecionados.

Projeto	**Descrição**	***Commits***	***Merges***
antlr/antlr4	Parser para texto estruturado ou arquivos binários	7.152	1.425
arduino/Arduino	Plataforma de computação física	6.968	778
dianping/cat	Plataforma de monitoramento de aplicativos	7.239	1.924
dbeaver/dbeaver	Ferramenta de banco de dados multiplataforma	12.632	913
alibaba/druid	*Pool* de conexão com banco de dados	5.947	1.390
google/ExoPlayer	*Media player* extensível para Android	5.589	442
apache/flink	*Framework* de processamento de fluxo	16.235	653
apache/incubator-shardingsphere	Soluções de middleware de banco de dados distribuído	16.022	1.388
mockito/mockito	*Framework* para testes de unidade	5.604	443
netty/netty	*Framework* para aplicação de rede assíncrona	9.244	277
ReactiveX/RxJava	Implementação Java VM de Reactive Extensions	5.529	1.566
zaproxy/zaproxy	Ferramenta de segurança para aplicações Web	6.776	1.944
Totais		104.937	13.143

merges e, devido à grande carga de processamento e tempo para extrair os acoplamentos lógico e conceitual, foram descartados os projetos maiores em termo do número de *commits*, ficando com doze projetos. Os doze projetos escolhidos são: *Netty*, *Mockito*, *ExoPlayer*, *Flink*, *Arduino*, *Dbeaver*, *Antlr4*, *Incubator-Shardingsphere*, *Druid*, *RxJava*, *Cat* e *Zaproxy*. A Tabela 4.1 apresenta uma descrição geral dos projetos selecionados. No total, os projetos possuem 13.143 *merges*.

Em seguida, foram selecionados para análise apenas os *merges* que possuem modificações em arquivos Java em ambos os ramos. Por exemplo, suponha um *merge* que possui em um de seus ramos três arquivos modificados, A1, B1 e C1, sendo que A1 e B1 são arquivos texto e C1 é um arquivo Java. Já no outro ramo, dois arquivos foram modificados, A2 e B2, sendo que A2 é um arquivo xml e B2 é um arquivo Java. Neste cenário, esse *merge* seria considerado na nossa análise e a extração dos acoplamentos seria feita somente entre o arquivo C1, do primeiro ramo, e o arquivo B2, do segundo ramo.

Por outro lado, alguns *merges* foram descartados da análise. Não foram considerados os *merges* que foram criados pelo uso da opção *no fast-forward*, pois possuem modificações em apenas um dos dois ramos, não sendo pertinente o cálculo de acoplamento ou esforço. Além disso, não foram considerados os *merges* com mais de dois pais, chamados *octopus*. Por definição, esses *merges* não podem ter conflitos e edições manuais e, portanto, não possuem esforço de *merge* [37].

Portanto, a Tabela 4.2 apresenta os dados resultantes da seleção dos *commits* de *merge*, onde foram eliminados 3.821 *merges no fast-forward* e 2.946 *merges* que não possuem arquivos Java em ambos os ramos. No entanto, não foram encontrados *merges* do tipo *octopus*. Então, após o processo de seleção dos *commits* de *merge* restaram 6.376 *merges*

para serem analisados. Cabe ressaltar que os projetos foram clonados em 1 de maio de 2019.

Tabela 4.2: Caracterização dos *merges*.

Projetos	**No *Fast-Forward***	**Sem Arquivos Java**	***Octopus***	**Analisados**
antlr/antlr4	541	300	0	584
arduino/Arduino	148	469	0	161
dianping/cat	486	444	0	994
dbeaver/dbeaver	120	295	0	498
alibaba/druid	276	503	0	611
google/ExoPlayer	127	59	0	256
apache/flink	24	72	0	557
apache/incubator-shardingspher	349	125	0	914
mockito/mockito	209	79	0	155
netty/netty	112	17	0	148
ReactiveX/RxJava	690	138	0	738
zaproxy/zaproxy	739	445	0	760
Totais	3.821	2.946	0	6.376

4.2.3 Análises

Para as QP1 e QP2, foram calculadas as correlações entre os três tipos de acoplamentos entre ramos e entre os três tipos de esforços de *merge*. Adicionalmente, foram avaliadas as interações entre os acoplamentos e entre os esforços de *merge*, com o intuito de identificar possíveis sobreposições.

Com relação à QP3, a análise de correlação entre os acoplamentos lógico e conceitual e os esforços de *merge* foi realizada variando-se o *threshold* de zero a um, com a etapa de 0,1. Isso porque não foram encontrados na literatura estudos que definam um *threshold* para esses dois tipos de acoplamento. Por outro lado, devido à métrica do acoplamento estrutural ser binária (i.e., ou tem acoplamento ou não tem, sem uma percepção de magnitude, como ocorre nos outros dois acoplamentos), a análise da correlação entre o acoplamento estrutural e os esforços de *merge* foi realizada sem variação de *threshold*. Para tal, com o intuito de analisar os padrões das correlações conforme a variação do *threshold*, optou-se por realizar as análises das correlações, agrupadas por *threshold*. Posteriormente, com base nos resultados das correlações, optou-se por um *threshold* único para realizar as análises de ambos os acoplamentos.

Em seguida, para calcular a correlação entre os tipos de acoplamento de *software* (estrutural, lógico e conceitual) e os tipos de esforço de *merge* (retrabalho, trabalho desperdiçado e trabalho extra), foi utilizado o coeficiente de correlação de *Spearman* [55], o qual permite a avaliação de relações monotônicas não lineares. A correlação foi interpretada conforme a tabela 4.3 e o nível de significância estabelecido para esta análise é de

95% (i.e, *alpha-valor* = 0,05). A interpretação é realizada para o valor absoluto e o sinal + indica se a correlação é direta, ou seja, quando um valor aumenta, o outro também aumenta. Por outro lado, o sinal − indica que a correlação é inversa, ou seja, quando um valor aumenta, o outro diminui.

Tabela 4.3: Interpretação da correlação de *Spearman* [22].

Correlação	**Min**	**Max**
Insignificante	0,00	0,29
Baixa	0,30	0,49
Moderada	0,50	0,69
Alta	0,70	0,89
Muito Alta	0,90	1,00

Um *merge* pode ter um ou mais valores de acoplamento, que dependem do número de métodos que foram modificados entre os ramos. O acoplamento é calculado para cada par de métodos, como já foi discutido no Capítulo 3. Por exemplo, para um *merge* que possui três métodos modificados em um ramo e dois métodos modificados em um outro ramo, é realizado o cálculo do acoplamento para as seis possibilidades de acoplamento entre os métodos do primeiro ramo com os métodos do segundo ramo (3×2), resultando em seis valores de acoplamento.

Com o intuito de agrupar os *merges* por *threshold*, foi realizada a soma dos valores de acoplamento dos pares de métodos de cada *merge* que são maiores ou iguais ao *threshold* definido. Desta forma, para o *threshold* zero, todos os valores de acoplamento dos pares de métodos dos *merges* são somados. Por outro lado, quando o *threshold* é maior que zero, serão somados somente os valores de acoplamento dos pares de métodos que são maiores ou iguais ao valor do *threshold* definido. Assim, caso o *merge* não possua valores de acoplamento dos pares de métodos maiores ou iguais ao *threshold*, o acoplamento do *merge* é definido como zero.

Para analisar os padrões das correlações conforme a variação do *threshold*, variou-se o *threshold*, iniciando com zero e aumentando-o até 1, em passos de 0,1. A variação do *threshold* foi realizada somente para as métricas dos acoplamentos lógico e conceitual, devido à métrica do acoplamento estrutural ser binária, conforme discutido na Seção 2.3. Por isso, as correlações referentes ao acoplamento estrutural estão representadas separadamente na Tabela 4.4, onde o *threshold* é zero. Além disso, foram realizadas todas as combinações possíveis desses valores e foram obtidas onze bases (i.e., bases com *threshold* variando de 0 até 1 com passos de 0,1) com os dados das métricas de acoplamento conceitual e onze bases com os dados das métricas do acoplamento lógico, resultando em

121 (11×11) bases de dados com todos os atributos binários, uma para cada combinação possível dos *thresholds*.

Para tal, foram extraídos os acoplamentos entre ramos e os esforços de *merge* para cada um dos 6.376 *merges* selecionados e, posteriormente, foi calculada a correlação entre os acoplamentos e os esforços de *merge*, variando o *threshold*. Logo, foram obtidas nove combinações entre as três métricas de acoplamento e as três métricas de esforço, totalizando nove valores de correlação para cada *threshold* (3×3).

Adicionalmente, alguns padrões de correlação entre os acoplamentos e os esforços de *merge* podem variar em função das características dos ramos, tendo em vista que os *merges* dos projetos analisados possuem características diferentes. Por exemplo, referente aos arquivos modificados entre ramos, um *merge*[1] do projeto *RxJava*, contém 368 arquivos modificados em um ramo e 17 arquivos modificados em outro ramo. Por outro lado, o mesmo projeto possui 16 *merges*[2] com somente um arquivo modificado em ambos os ramos. Com relação ao número de linhas modificadas entre ramos, o projeto *ANTLR* possui um *merge*[3] que contém 33.891 linhas de código modificadas em um ramo e 6.781 linhas modificadas em outro ramo. Entretanto, um outro *merge*[4] possui somente duas linhas de código modificadas em cada ramo. Referente ao número de *commits* realizados nos ramos, o projeto *DBEAVER* possui um *merge*[5] com 391 *commits* em um ramo e apenas um *commit* em outro ramo. No entanto, este projeto possui 33 *merges*[6] com apenas um *commit* em ambos os ramos. Por fim, com relação ao número de autores, dois *merges*[7] do projeto *ExoPlayer* contêm 27 autores em um ramo e 17 em outro ramo. Contudo, este projeto possui 39 *merges*[8] com apenas um autor em ambos os ramos.

Portanto, para cada *merge* do corpus apresentado na Seção 4.2.2, foram extraídos quatro atributos de cada ramo: número de arquivos modificados, número de linhas modificadas, número de *commits* e número de autores, a fim de analisar a influência destes atributos dos ramos na correlação entre os acoplamentos e os esforços de *merge*. De modo geral, os *merges* mais críticos são aqueles que em ambos os ramos existe uma ati-

[1] 4c38a296646718ef79e51818fb42bc07812b9038

[2] três *merges* como exemplo: d7d94ba97e7a17e276a9d560b0979da6acb9e8f4,9a35d798bb22052532af400b8b324d38dd
aea1e3befdd78d135be1d8bfa14e039b2cee9fa0

[3] ecc2a58615f63bd74d535c01127fb31c6dfbbdea

[4] bf4fa40ff91afbaad1a33f9623cd27abde22e097

[5] 25d0c811572338bfef9561e0d77cb84241098775

[6] como exemplo: 00b6ad3c5d5fb8bc444383087d76b4931374f91a, 0a614e93eeac951bcb760f16881be375c318fee1,
2336f577f60b7bad3e69761824715bb3572faa2f

[7] aa02df78ebd5a0eceee724f3fcfd6ba1f4b79044, 1f6607a9aa6a85eeccb52d64e4fdde89288a0bc9

[8] dd2921f9b27586d06bbc48fd930bd1f663663ce6, 04cead415ad7c7fd6427de65ecb7f94a0dc14367,
059835e357ec11f54f52831704fe9490da801bf3

vidade muito grande, ou seja, os valores dos atributos são elevados. Para tal, para cada atributo foi calculada a média harmônica entre os ramos, pois a média harmônica será forte somente quando a métrica do atributo for elevada em ambos os ramos, indicando as situações mais complexas. Logo, foi calculada a média harmônica entre:

- #Arquivos: o número de arquivos modificados no ramo *esquerdo* e o número de arquivos modificados no ramo *direito*;
- #Linhas: o número de linhas modificadas no ramo *esquerdo* e o número de linhas modificadas no ramo *direito*;
- #*Commits*: o número de *commits* realizados no ramo *esquerdo* e o número de *commits* realizados no ramo *direito*; e
- #Autores: o número de autores dos *commits* no ramo *esquerdo* e o número de autores dos *commits* no ramo *direito*.

A Figura 4.2 apresenta a distribuição dos atributos dos ramos em *boxplots*. Para melhor visualização, alguns *outliers* foram omitidos. Os atributos dos ramos possuem uma mediana de 3,87 Arquivos Modificados, 75,40 Linhas Modificadas, 1,88 *Commits* e 1,33 Autores. Além disso, os outliers máximos para cada atributo são 1.221,08 para Arquivos, 145.696,47 para Linhas, 296,99 para Commits e 22,86 para Autores. Portanto, devido à escala do atributo Linhas ser bem maior que a dos demais atributos, os *boxplots* foram separados para não prejudicar a visualização dos *boxplots* dos atributos Arquivos, *Commits* e Autores.

Adicionalmente, para cada um desse quatro atributos, foi calculada a mediana das médias harmônicas e os *merges* foram separados em dois grupos: *menos atributos* e *mais atributos*. O grupo *menos atributos* contém os *merges* que possuem os valores de média harmônica menores que a mediana e o grupo *mais atributos* contém os *merges* que possuem os valores de média harmônica maiores que a mediana. Por conseguinte, para cada um dos 12 projetos, foi calculada a correlação entre os acoplamentos e os esforços de *merge* nos grupos *menos atributos* e *mais atributos*. Com isso, foram obtidos 12 valores de correlação para cada grupo. A partir destes valores de correlação, foi analisado se algum dos quatro atributos influencia a correlação entre os acoplamentos e os esforços de *merge*.

É importante ressaltar que, para todas as análises, foi utilizada a base de dados com *threshold* igual a zero, ou seja, foram consideradas todas as possibilidades de acoplamento,

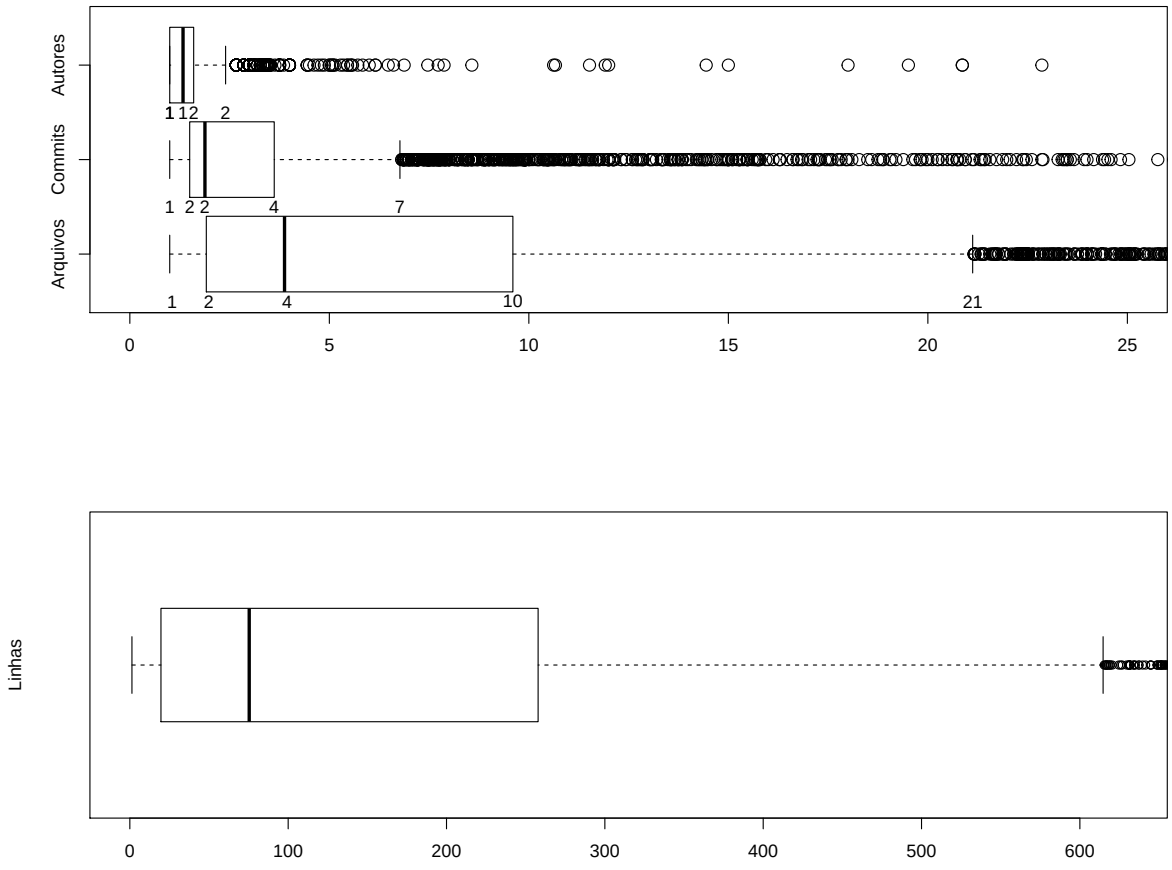


Figura 4.2: Distribuição da quantidade de atributos dos ramos do Corpus (alguns *outliers* omitidos).

sem nenhuma filtragem. Essa decisão foi tomada, porque mesmo que exista algum acoplamento falso positivo, a diferença de *merges* com acoplamento maior que zero entre um *threshold* e outro não é muito grande, conforme pode ser visto na Tabela 4.6. Portanto, a adoção de algum outro *threshold* não influenciaria no resultado final.

Para as análises, foi utilizado o teste de hipóteses não paramétrico de *Wilcoxon*[9]. De acordo com Warner [57], métodos não paramétricos devem ser utilizados para amostras pequenas. Por isso, optou-se por utilizar o teste pareado de Wilcoxon, devido à amostra deste estudo ser pequena (n=12), mesmo sem realizar os testes de normalidade e de homocedasticidade dos dados. Por último, em complementação aos testes de hipóteses, foi calculado o tamanho do efeito *Cliff's Delta* [10] (utilizado para métodos não paramétricos) dos quatro atributos (#Arquivos, #Linhas, #*Commits* e #Autores) de cada ramo. O tamanho do efeito *Cliff's Delta* pode ser negligenciável (Cliff's Delta $< 0{,}147$), pequeno ($0{,}147 \leq$ Cliff's Delta $< 0{,}33$), médio ($0{,}33 \leq$ Cliff's Delta $< 0{,}474$) ou grande (Cliff's Delta $\geq 0{,}474$) [48].

4.2.4 Implementação

Para extrair as métricas dos acoplamentos estrutural, lógico e conceitual, foram adaptadas três ferramentas existentes. Elas já detectam o acoplamento de *software* e foram adaptadas para detectar o acoplamento entre ramos. Além disso, para extrair as métricas dos esforços de *merge*: retrabalho, trabalho desperdiçado e trabalho extra, foi utilizada a ferramenta de esforço de *merge* desenvolvida por Moura e Murta [37]. Essa ferramenta calcula as métricas do esforço de *merge* para cada *commit* de *merge* de um repositório *Git*.

Para obter as dependências estruturais entre ramos, foram adaptados os *scripts* dos projetos *dependencies* e *resources* do repositório *merge-nature* [11]. Este projeto foi desenvolvido por Menezes et al. [34] para identificar a dependência estrutural entre os trechos conflitantes nos *commits* de *merge* e para entender como esses conflitos foram resolvidos. A adaptação da ferramenta consistiu primeiramente na inclusão da busca por *commits* de *merge* que possuem dois pais e na inclusão da extração da AST do *merge-base*, pois a extração das ASTs dos ramos *esquerdo* e *direito* já eram realizadas. Foi adaptada a função que realiza o reposicionamento das linhas para identificar a que método a linha alterada pertencente, por meio da comparação com o posicionamento das linhas da AST

[9] *wilcox.test(dados$menos atributos, dados$mais atributos, paired = TRUE)*, comando utilizado no R
[10] *cliff.delta(dados$menos atributos, dados$mais atributos, paired = TRUE)*, comando utilizado no R
[11] http://gems-uff.github.io/merge-nature/

do *merge-base*. Como também, foi alterada a função que faz o comparação dos *bindings* das ASTs para comparar os *bindings* em ASTs diferentes.

Além disso, o *TipMerge* [12] foi adaptado para identificar o acoplamento lógico entre os ramos, no grão do método. O TipMerge é baseado na ferramenta *Dominoes*, que organiza dados extraídos de repositórios de *software* em matrizes para identificar relacionamentos entre entidades de *software* [53]. Como discutido anteriormente, a métrica de confiança foi utilizada para calcular o acoplamento lógico para os métodos modificados entre os ramos. Para a adaptação da ferramenta *TIPMerge* foram reutilizados os códigos comuns para os três acoplamentos, que são: busca por *commits* de *merge* com dois pais, extração das ASTs e funções que identificam os métodos modificados a partir das informações das ASTs. Foi alterada a construção da matriz do *Dominoes*, de *Commit* × *Arquivo* para *Commit* × *Método*. Além do mais, foram modificadas todas as classes que tratam as informações dos arquivos editados para tratar as informações dos métodos alterados.

Finalmente, para identificar o acoplamento conceitual entre ramos, foi adaptada a ferramenta *Semantic Similarity Java* desenvolvida por Ajienka e Capiluppi [2], para calcular o acoplamento conceitual entre ramos. Esta ferramenta foi desenvolvida originalmente para calcular a similaridade semântica entre classes Java em projetos de *software*. A adaptação para a extração do acoplamento conceitual consistiu na criação de um *script* que reutilizou os códigos comuns para os três acoplamentos, conforme dito anteriormente. Além disso, o *script* extrai as linhas modificadas a partir do resultado do *diff*, cria um arquivo txt para cada método, contendo as linhas modificadas nos respectivos métodos. Portanto, foi necessário adaptar na ferramenta *Semantic Similarity Java* os diretórios de leitura dos arquivos como também os diretórios de escrita dos resultados, para que a ferramenta encontre os arquivos e realize a extração dos termos dos arquivos para calcular o acoplamento conceitual entre métodos.

Todas as adaptações estão disponíveis no repositório *merge-coupling* [13].

4.3 Resultados e Discussões

A seguir, são apresentados os resultados das análises das correlações entre os acoplamentos entre ramos; entre os esforços de *merge*; e entre os acoplamentos e esforços de *merge*. Além dos resultados da análise da influência dos atributos dos ramos na correlação entre

[12] http://gems-uff.github.io/tipmerge/
[13] https://gems-uff.github.io/merge-coupling/

os acoplamentos entre ramos e os esforços de *merge*. As análises foram realizadas sobre o corpus definido na subseção 4.2.2, para responder as questões de pesquisa QP1, QP2 e QP3 e QP4.

QP1: Quais acoplamentos entre ramos se correlacionam com o outro?

Na literatura, alguns estudos já analisaram o relacionamento entre os acoplamentos estrutural, lógico e conceitual, porém, não analisaram os acoplamentos entre ramos. Um estudo identificou que as dependências estruturais não levam a dependências lógicas, porém há evidências de que os pares de classes estruturalmente acoplados geralmente incluem dependências lógicas [38]. No entanto, outro estudo não identificou correlação significante entre acoplamento estrutural e lógico entre classes [3]. Por outro lado, um estudo [4] concluiu que há uma relação direcional entre os acoplamentos conceitual e lógico entre classes.

Diante do exposto e com intuito de analisar o relacionamento entre as métricas de acoplamento entre ramos, foi calculada a correlação entre os acoplamentos e notou-se que todas as correlações são significantes (p-valor $<$ 0,05). Identificou-se uma correlação moderada ($\rho = 0,61$) entre os acoplamentos lógico e conceitual e uma correlação fraca entre os acoplamentos estrutural e lógico ($\rho = 0,40$), assim como, entre os acoplamentos estrutural e conceitual ($\rho = 0,44$).

Para tal, a Figura 4.3 ilustra o relacionamento e algumas sobreposições dos acoplamentos entre ramos, em diagrama de *Venn*. Observou-se que há mais *merges* que possuem tanto o acoplamento lógico quanto o conceitual do que com os demais acoplamentos. Isso ocorre, porque foi identificado um maior número de *merges* com acoplamento lógico e conceitual, do que com acoplamento estrutural. No entanto, apesar do número pequeno de *merges* com acoplamento estrutural, notou-se que a maioria destes *merges* também possui os acoplamentos lógico (750) e conceitual (840). Vale ressaltar que dentre os 6.736 *merges* analisados, 730 possuem os três acoplamentos. De acordo com a Figura 4.3, dentre os 881 *merges* com acoplamento estrutural, 750 (85%) também possuem acoplamento lógico e 840 (95%) também possuem acoplamento conceitual. Além disso, dentre os 3.371 *merges* com acoplamento lógico, 750 (22%) também possuem acoplamento estrutural e 3.120 (92%) também possuem acoplamento conceitual. Por fim, dentre as 5.010 *merges* com acoplamento conceitual, 3.120 (62 %) também possuem acoplamento lógico e 840 (17%) também possuem acoplamento estrutural. Portanto, conclui-se que existe uma grande sobreposição direcional entre os acoplamentos estrutural e lógico, uma vez que 85% dos *merges* com acoplamento estrutural também possuem acoplamento lógico. Observou-se a

mesma situação entre os acoplamentos estrutural e conceitual, pois 95% dos *merges* com acoplamento estrutural também possuem acoplamento conceitual. Da mesma forma, há uma grande sobreposição direcional entre os acoplamentos lógico e conceitual, pois 92% dos *merges* com acoplamento lógico também possuem o acoplamento conceitual. Cabe ressaltar que, foram considerados os *merges* com acoplamento lógico ou conceitual, aqueles que possuem o valor de acoplamento maior que zero, por isso, o valor pode ser próximo de zero e assim, gerar um falso-positivo de acoplamento.

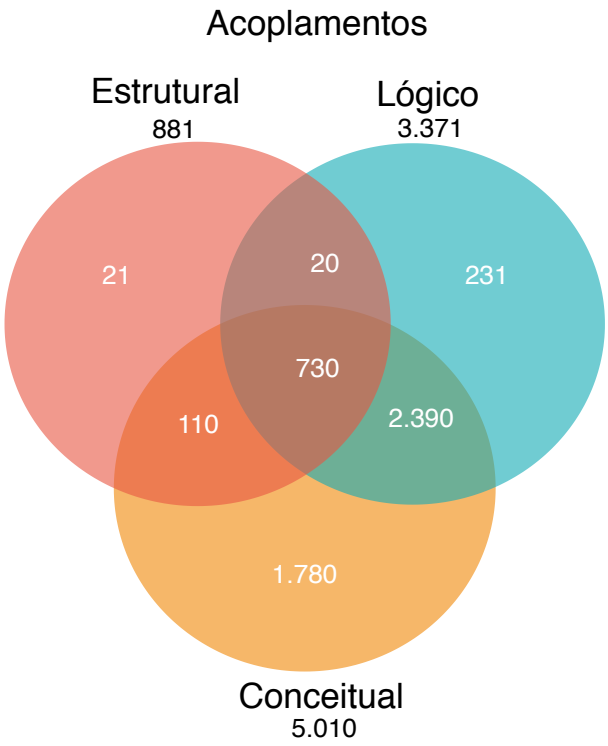


Figura 4.3: Diagrama de *Venn* ilustrando o relacionamento entre os acoplamentos entre ramos.

QP1: *Quais acoplamentos entre ramos se correlacionam com o outro?*
Resposta: Existe uma correlação moderada ($\rho = 0,61$) entre os acoplamentos lógico e conceitual. Além disso, existe uma correlação fraca entre os acoplamentos estrutural e lógico ($\rho = 0,40$), bem como entre os acoplamentos estrutural e conceitual ($\rho = 0,44$). Identificou-se uma grande sobreposição direcional entre o acoplamento estrutural e os acoplamentos lógico (85%) e conceitual (95%), bem como entre os acoplamentos lógico e conceitual (92%).
Implicações: Durante a execução dos experimentos, observou-se que a extração do acoplamento estrutural é três vezes mais rápida do que a do acoplamento conceitual e trinta vezes mais rápida do que a extração do acoplamento lógico. Além disso, o acoplamento estrutural pode indicar com aproximadamente 85% ($750 \div 881$) de confiança que um *merge* possui acoplamento lógico e 95% ($840 \div 881$) de confiança que um *merge* possui acoplamento conceitual. Ou seja, quando o acoplamento estrutural ocorre há 85% de chance do acoplamento lógico ocorrer e há 95% de chance do acoplamento conceitual ocorrer. Portanto, ferramentas que utilizam acoplamentos lógico ou conceitual e não são sensíveis a falsos positivos, podem calcular o acoplamento estrutural e usá-lo como preditor. Isso aceleraria a execução da ferramenta com apenas 7% a 15% de falsos positivos.

QP2: Quais esforços de *merge* se correlacionam com o outro?

Com o intuito de analisar o relacionamento entre as métricas de esforço, foi calculada a correlação entre os esforços de *merge* e notou-se que todas as correlações são significantes (p-valor $<$ 0,05). Foi identificado que há uma correlação forte ($\rho = 0,87$) entre o retrabalho e o trabalho desperdiçado. Relembrando que a métrica de retrabalho consiste no código idêntico criado em paralelo em ambos os ramos e o trabalho desperdiçado consiste no código que foi criado em um dos ramos, mas que foi descartado durante o *merge*. É razoável que exista uma forte correlação entre esses esforços, já que o esforço de retrabalho é consequência do código criado em ambos os ramos e, normalmente, um dos códigos criados é descartado ao realizar o *merge*. Então, na maioria das vezes quando ocorre o esforço de retrabalho, também ocorre o esforço de trabalho desperdiçado. No entanto, há alguns casos em que é criado código igual e em ambos os ramos, mas um dos códigos não é descartado ao realizar o *merge*, pois este é necessário no outro ramo também. Como exemplo, em um *merge* [14], do projeto *ANTLR*, quatro linhas foram criadas

[14]Merge SHA-1: da9186ed6342d75c6d771bb2d52fbfea0c42a166

com espaços em branco em ambos os ramos e, apesar dos códigos serem iguais, eles são necessários na versão final, e não foram descartados. Para este *merge* foi computado o valor 4 para o esforço de retrabalho e o valor 0 para os esforços de trabalho desperdiçado e trabalho extra. Identificou-se que cenários semelhantes a este também ocorreram para alguns outros *merges* analisados.

Além disso, observou-se que há uma correlação moderada entre o retrabalho e o esforço de trabalho extra ($\rho = 0,50$), assim como uma correlação moderada entre o esforço de trabalho desperdiçado e o esforço de trabalho extra ($\rho = 0,63$). De acordo com a Figura 4.4, identificou-se que 1.485 *merges* com esforço de retrabalho, 1.316 *merges* com esforço de trabalho desperdiçado e 645 *merges* com esforço de trabalho extra. Além disso, dentre 1.485 *merges* com esforço de retrabalho, 1.215 (81%) também possuem esforço desperdiçado e 557 (37%) também possuem esforço de trabalho extra. Além disso, dentre 1.316 *merges* com esforço de trabalho desperdiçado, 1.215 (92%) também possuem esforço de retrabalho e 621 (47%) também possuem esforço de trabalho extra. Finalmente, dentre os 645 *merges* com esforço de trabalho extra, 557 (86%) também possuem esforço de retrabalho e 621 (96%) também possuem esforço de trabalho desperdiçado. Portanto, conclui-se que há uma grande sobreposição direcional entre os esforços de *merge*, exceto entre os esforços de retrabalho e trabalho desperdiçado e o esforço de trabalho extra. Vale ressaltar que dentre os 6.736 *merges* analisados, 555 possuem os três esforços de *merge*.

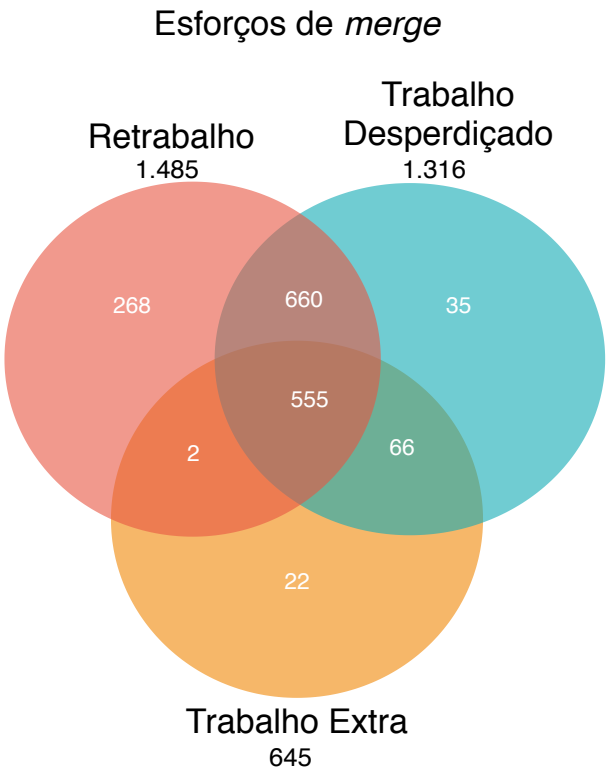


Figura 4.4: Diagrama de *Venn* ilustrando o relacionamento entre os esforços de *merge*.

QP2: *Quais esforços de merge se correlacionam com o outro?*
Resposta: A correlação é forte entre os esforços de retrabalho e trabalho desperdiçado, já entre o esforço de trabalho extra e os esforços de retrabalho e trabalho desperdiçado, a correlação é moderada.
Implicações: O esforço de retrabalho é um bom preditor para o esforço de trabalho desperdiçado ($\rho = 0,76$), mesmo quando os *merges* que têm zero de esforços de retrabalho e trabalho desperdiçado são descartados. Considerando que o retrabalho é uma métrica disponível antes do *merge* e o trabalho desperdiçado é uma métrica disponível logo após o *merge*, as ferramentas de percepção poderiam calcular o esforço de retrabalho para indicar a chance de haver um esforço de trabalho desperdiçado.

QP3: Quais acoplamentos entre ramos se correlacionam com esforço de *merge*?

Com o intuito de contrastar as métricas absoluta (i.e., $Total_{tipo}$) e normalizada (i.e., $Normalizado_{tipo}$), que foram calculadas conforme discutido nas Seções 3.3.4 e 3.3.5, foram analisados os padrões dessas métricas. A Tabela 4.4 mostra as correlações entre o acoplamento estrutural e os esforços de *merge*. A coluna **TEst** mostra as correlações utili-

zando a métrica $Total_{estrutural}$. A coluna **NEst** mostra as correlações utilizando a métrica $Normalizado_{estrutural}$. A Tabela 4.5 mostra as correlações entre os acoplamentos lógico e conceitual e os esforços de *merge*. As colunas **TLog** e **TCon** mostram as correlações utilizando as métricas $Total_{logico}$ e $Total_{conceitual}$, respectivamente. As colunas **NLog** e **NCon** mostram as correlações utilizando as métricas $Normalizado_{logico}$ e $Normalizado_{conceitual}$, respectivamente. De acordo com as Tabelas 4.4 e 4.5, notou-se que as correlações referentes à métrica normalizada são frequentemente menores do que as correlações referentes à métrica absoluta. Além disso, referente ao acoplamento conceitual, a diferença entre as métricas absoluta e normalizada é maior para os *thresholds* mais baixos. No entanto, quando o *threshold* se aproxima de 1, não ocorre mais diferença entre elas, exceto para o esforço de trabalho extra. Já para o acoplamento lógico, a diferença entre as métricas se mantém semelhante na variação do *threshold*. Por fim, pode-se dizer que a maior diferença entre essas métricas acontece para as correlações entre os acoplamentos lógico e conceitual e o esforço de trabalho extra.

Tabela 4.4: Correlações entre o acoplamento estrutural e os esforços de *merge*.

Esforço de *merge*	**TEst**	**NEst**
Retrabalho	0,51	0,49
Desperdiçado	0,52	0,49
Extra	0,39	0,36

Tabela 4.5: Correlações entre os acoplamentos (lógico e conceitual) e os esforços de *merge*.

Th	**Retrabalho**				**Desperdiçado**				**Extra**			
	TLog	**NLog**	**TCon**	**NCon**	**TLog**	**NLog**	**TCon**	**NCon**	**TLog**	**NLog**	**TCon**	**NCon**
0	0,44	0,38	0,43	0,39	0,42	0,35	0,43	0,38	0,30	0,22	0,30	0,23
0,1	0,44	0,38	0,46	0,42	0,43	0,36	0,47	0,42	0,31	0,23	0,32	0,25
0,2	0,45	0,38	0,50	0,46	0,43	0,36	0,50	0,46	0,31	0,23	0,34	0,27
0,3	0,46	0,39	0,53	0,50	0,44	0,37	0,54	0,51	0,31	0,23	0,35	0,30
0,4	0,46	0,39	0,56	0,54	0,45	0,37	0,58	0,55	0,32	0,24	0,37	0,32
0,5	0,46	0,39	0,59	0,57	0,45	0,38	0,61	0,58	0,32	0,24	0,37	0,33
0,6	0,47	0,41	0,61	0,59	0,46	0,39	0,63	0,61	0,33	0,26	0,38	0,34
0,7	0,47	0,41	0,63	0,61	0,46	0,39	0,65	0,64	0,33	0,26	0,37	0,34
0,8	0,48	0,41	0,63	0,62	0,47	0,39	0,66	0,65	0,33	0,26	0,36	0,34
0,9	0,48	0,41	0,63	0,62	0,47	0,40	0,66	0,65	0,33	0,26	0,34	0,33
1	0,48	0,41	0,48	0.48	0,47	0,40	0,50	0,50	0,33	0,26	0.22	0.21

Além disso, foi identificado que há uma correlação forte ($\rho = 0,73$) entre as métricas absoluta e normalizada do acoplamento conceitual, e há uma correlação muito forte ($\rho = 0,91$) entre as métricas absoluta e normalizada do acoplamento lógico. Por isso, a escolha por qualquer uma das duas métricas é razoável para continuar com as análises, já que ambas apontam para a mesma direção. Diante disso, optou-se por utilizar somente as correlações referentes à métrica absoluta, ou seja, as métricas representadas pelas colunas **TEst**, **TLog** e **TCon**, para todas as análises seguintes.

Tabela 4.6: Quantidade de *merges* com acoplamento > 0, de acordo com a variação do *threshold*.

Threshold	**Estrutural**	**Lógico**	**Conceitual**
0	881	3.371	5.010
0,1	-	3.293	4.213
0,2	-	3.193	3.519
0,3	-	3.060	2.906
0,4	-	2.956	2.435
0,5	-	2.892	1.998
0,6	-	2.708	1.632
0,7	-	2.592	1.327
0,8	-	2.543	1.077
0,9	-	2.476	872
1	-	2.431	448

Por conseguinte, antes de fazer uma análise mais aprofundada, notou-se que, no geral, quanto maior é o *threshold*, maior é a correlação entre os acoplamentos e os esforços de *merge*. Isso é esperado, tendo em vista que à medida que o threshold aumenta, a tendência é selecionar aqueles que estão mais fortemente acoplados, ou seja, que possuem um valor de acoplamento maior. Além disso, devido ao aumento de *merges* com valor de acoplamento igual a zero à medida que *threshold* aumenta, a tendência é ter uma correlação maior também. Com isso, à medida que o *threshold* aumenta, a quantidade de *merges* com valor de acoplamento igual ao do *threshold* diminui, como pode ser visto na Tabela 4.6. Para o *threshold* 0,1 há 3.293 *merges* com acoplamento lógico maior ou igual a 0,1 e 4.213 *merges* com acoplamento conceitual maior ou igual a 0,1. No entanto, para o *threshold* 0,9, os valores diminuem, chegando a 2.476 *merges* com acoplamento lógico maior ou igual a 0,9 e 872 com acoplamento conceitual maior ou igual a 0,9. Além disso, é possível perceber que, para o acoplamento lógico, a diferença de *merges* entre um *threshold* e outro é menor do que para o acoplamento conceitual, indicando assim que o acoplamento lógico possui valores maiores de acoplamento do que o acoplamento conceitual.

É importante ressaltar que todas as correlações são significantes (p-valor $<$ 0,05) e de acordo com a Tabela 4.5, pôde-se observar que, a partir do *threshold* 0,2, a correlação é moderada ($\rho = 0,50$) entre o esforço de retrabalho e o acoplamento conceitual, e a partir do *threshold* 0,2 a correlação também é moderada ($\rho = 0,50$) entre o acoplamento conceitual e o esforço de trabalho desperdiçado. Por outro lado, independentemente do valor do *threshold*, as Tabelas 4.4 e 4.5 mostram uma correlação fraca ($0,22 \leq \rho \leq 0,39$) entre o esforço de trabalho extra e os acoplamentos estrutural, lógico e conceitual. Provavelmente, este resultado é devido ao número elevado de *commits* de *merge* (89%)

que não possuem esforço de trabalho extra. Outra descoberta é que independentemente do *threshold*, a correlação é fraca entre o acoplamento lógico e todos os esforços de *merge*, onde entre o acoplamento lógico e o esforço de retrabalho a correlação varia entre 0,44 $\leq \rho \leq$ 0,48, entre o acoplamento lógico e o esforço de trabalho desperdiçado varia entre 0,42 $\leq \rho \leq$ 0,47, e entre o acoplamento lógico e esforço de trabalho extra, varia entre 0,30 $\leq \rho \leq$ 0,33.

Além disso, observou-se que, o acoplamento estrutural possui correlação moderada com os esforços de retrabalho ($\rho = 0,51$) e trabalho desperdiçado ($\rho = 0,52$), mas à medida que o *threshold* aumenta (*threshold* maior ou igual a 0,3), o acoplamento conceitual começa a correlacionar mais com estes dois esforços de *merge*, resultando em uma correlação moderada (0,53 $\leq \rho \leq$ 0,66). Logo, os valores de correlação referentes ao acoplamento conceitual passam a ser maiores que os valores de correlação referentes ao acoplamento estrutural. No entanto, não foi identificado que quando o threshold é 0,3 ele fica mais seletivo, ou seja, tem um aumento significativo de *merges* com acoplamento conceitual igual a zero. O aumento de *merges* com acoplamento igual a zero é gradativo, à medida que o *threshold* aumenta.

Cabe ressaltar que, nos *thresholds* 0,8, 0,9 e 1 notou-se uma queda do valor da correlação entre o acoplamento conceitual e o esforço de trabalho extra, sendo que mais acentuada no *threshold* 1. Apesar do número de *merges* com acoplamento igual a zero ser maior para os *thresholds* 0,8, 0,9 e 1 do que para o *threshold* 0,7. Assim como, observou-se um decréscimo nos valores de correlação entre o acoplamento conceitual e os esforços de retrabalho e de trabalho desperdiçado para o *threshold* 1.

Por fim, para ilustrar a variação dos valores de correlação de acordo com o *threshold*, as Figuras 4.5, 4.6 e 4.7 representam, em *boxplots*, a distribuição dos valores de correlação de cada um dos doze projetos de acordo com a variação do *threshold*. Conforme a Figura 4.5, notou-se que os valores de correlação entre acoplamento conceitual e retrabalho aumentam à medida que o *threshold* aumenta. Já os valores de correlação entre o acoplamento lógico e o esforço de retrabalho variam pouco à medida que o *threshold* aumenta.

De forma semelhante como ocorre com o esforço de retrabalho, a Figura 4.6 também mostra que há um aumento gradativo dos valores de correlação entre o acoplamento conceitual e o esforço de trabalho desperdiçado à medida que o *threshold* aumenta, exceto, quando o *threshold* é 1, pois há uma queda da mediana das correlações. No entanto, a variação entre os valores de correlação entre acoplamento lógico e esforço de trabalho desperdiçado é bem pequena, ou seja, o acoplamento lógico não é muito sensível ao

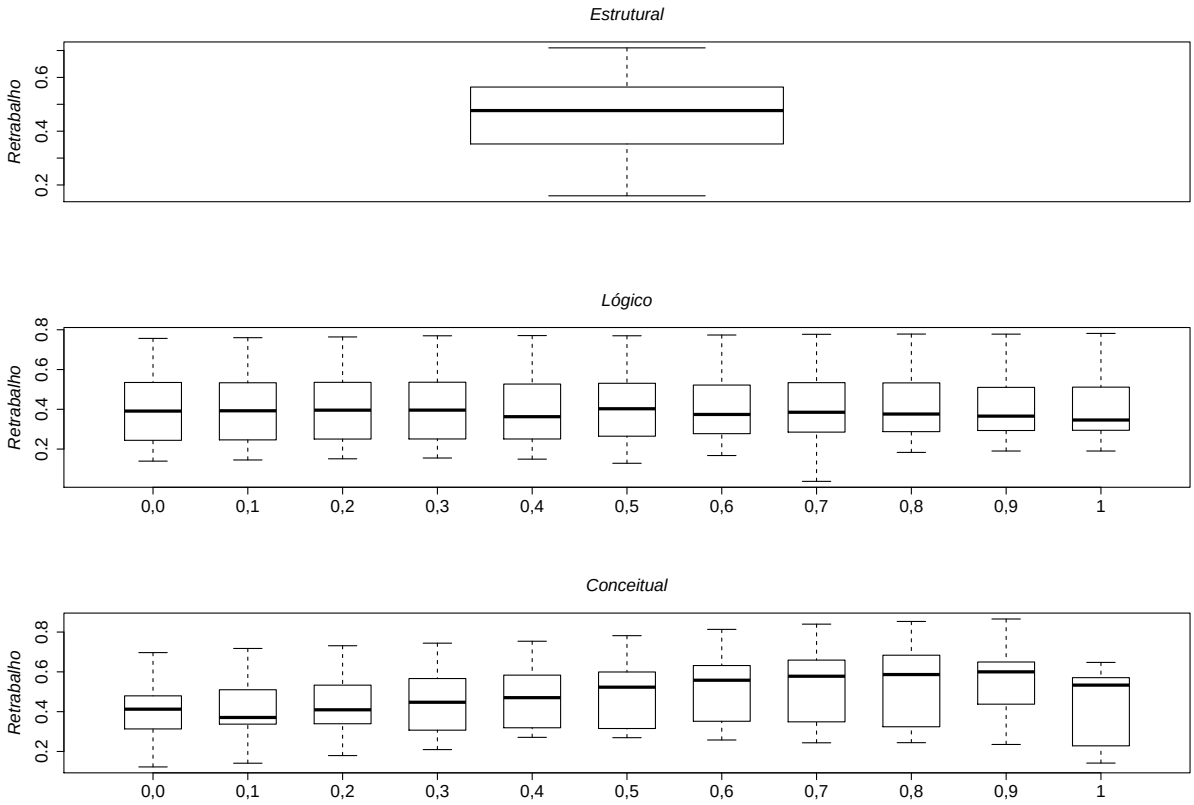


Figura 4.5: Distribuição dos valores de correlação entre os acoplamentos e o esforço de retrabalho.

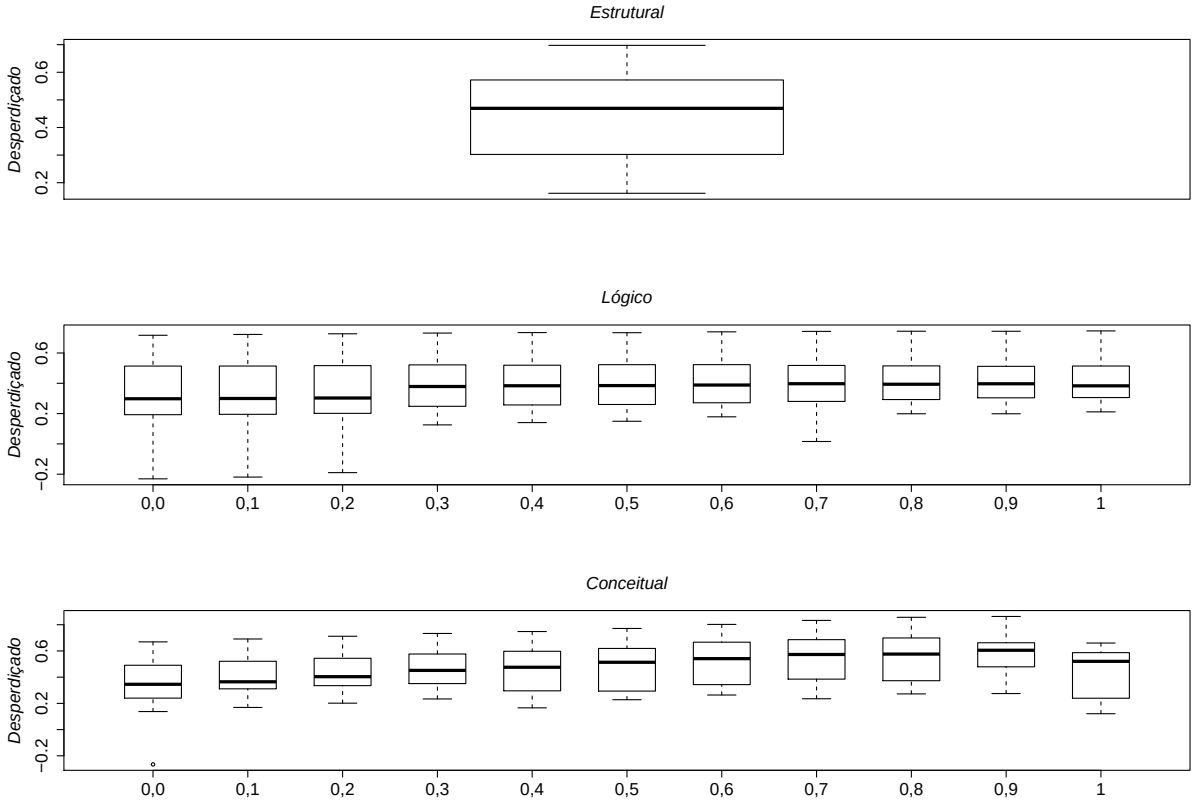


Figura 4.6: Distribuição dos valores de correlação entre os acoplamentos e o esforço de trabalho desperdiçado.

threshold.

Além disso, a Figura 4.7 mostra uma oscilação dos valores de correlação entre o acoplamento conceitual e o esforço de trabalho extra, à medida que o *threshold* aumenta. Para os *thresholds* 0,0, 0,1 e 0,2, a mediana das correlações entre o acoplamento conceitual e o esforço de trabalho extra variam pouco e há uma queda das medianas para os *thresholds* 0,4, 0,5 e 0,6. Já para os *thresholds* 0,7, 0,8 e 0,9 há um aumento gradativo das medianas, porém, para o *threshold* 1 há uma queda da mediana das correlações para um valor semelhante às medianas das correlações para os *thresholds* 0,0 e 0,1. Por outro lado, as medianas dos valores de correlação entre o acoplamento lógico e o esforço de trabalho extra variam pouco à medida que o *threshold* aumenta.

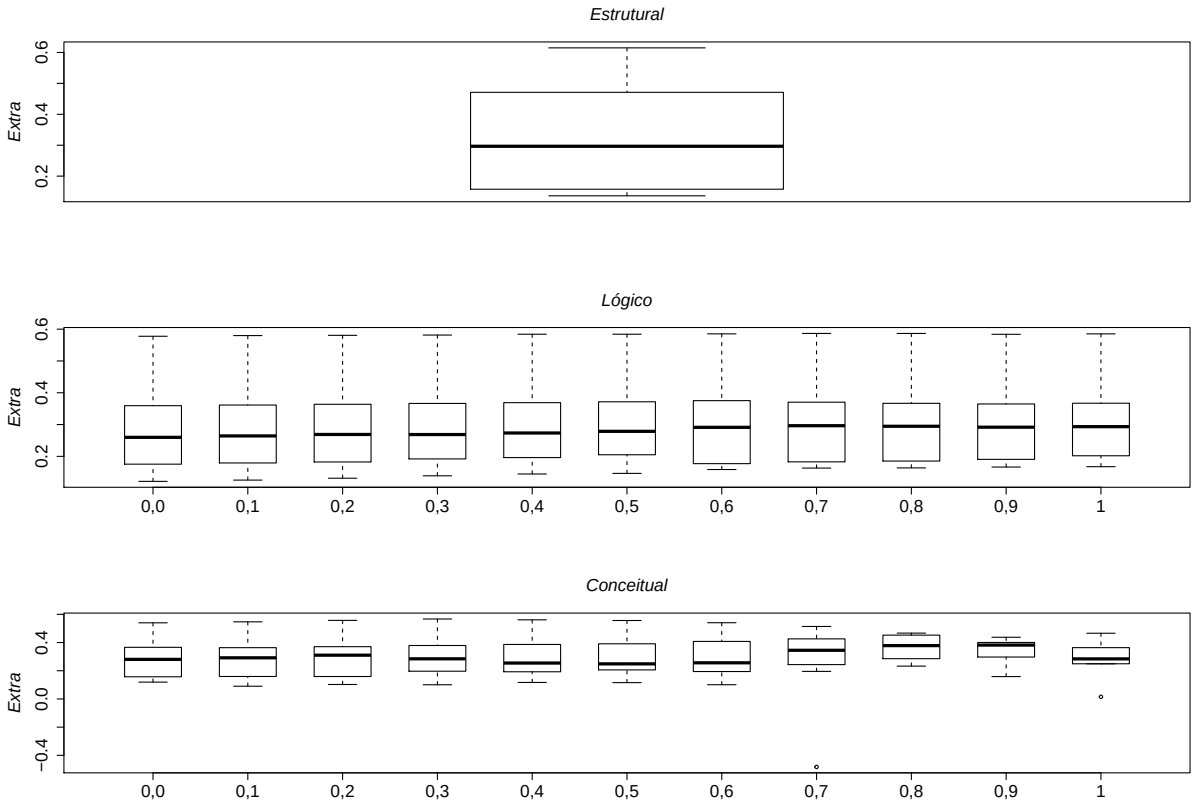


Figura 4.7: Distribuição dos valores de correlação entre os acoplamentos e o esforço de trabalho extra.

QP3: *Quais acoplamentos entre ramos se correlacionam com esforço de merge?*
Resposta: O acoplamento estrutural possui correlação moderada com os esforços de retrabalho e trabalho desperdiçado. De forma equivalente, o acoplamento conceitual possui correlação moderada com os esforços de retrabalho (a partir do *threshold* = 0,2) e trabalho desperdiçado (a partir do *threshold* = 0,2). Por outro lado, independentemente do valor do *threshold*, a correlação é fraca entre o esforço de trabalho extra e todos os acoplamentos, assim como, a correlação é fraca entre o acoplamento lógico e todos os esforços de *merge*. A correlação entre o acoplamento estrutural e o esforço de trabalho extra é maior que para os demais acoplamentos. Estudos anteriores [52, 17] mostraram que o acoplamento estrutural leva a conflitos indiretos e portanto, de acordo com os resultados deste estudo, pode ser um melhor preditor para tais conflitos.
Implicações: Os gerentes de projeto, ao escolher as tarefas a serem implementadas em paralelo, podem focar na redução do acoplamento estrutural entre as tarefas. Isso potencialmente diminuiria o esforço de trabalho extra durante a reintegração de tais tarefas. Complementarmente, os gerentes de projeto também podem focar na redução do acoplamento conceitual entre as tarefas para diminuir os esforços de retrabalho e trabalho desperdiçado durante a reintegração.

QP4: Quais características influenciam a correlação entre os acoplamentos entre ramos e os esforços de *merge*?

Nesta questão de pesquisa, o objetivo é identificar se alguns atributos dos ramos: #Arquivos, #Linhas, #*Commits* e #Autores podem influenciar a correlação entre acoplamentos e esforços de *merge*. Portanto, esta questão foi subdividida em outras quatro subquestões mais específicas (QP4.1, QP4.2, QP4.3 e QP4.4).

As Figuras 4.8, 4.9, 4.10 e 4.11 resumem a influência dos atributos #Arquivos, #Linhas, #*Commits* e #Autores, respectivamente, nas correlações entre os acoplamentos e os esforços de *merge* utilizando *boxplots*. Para tal, foram consideradas somente as correlações estatisticamente significantes (p-valor $<$ 0,05) e deste modo, foram descartadas as duas correlações de um projeto, caso este possua correlação significativa em um grupo, mas possua correlação insignificante em outro grupo. Cada *boxplot* representa a variação dos valores de correlação entre um tipo de acoplamento e um tipo de esforço, agrupado pelos atributos #Arquivos, #Linhas, #*Commits* e #Autores. Por exemplo, de acordo com a Figura 4.8, o grupo *menos arquivos*, apresenta as correlações entre um tipo de acoplamento e um tipo de esforço de *merge*, referentes ao grupo de *merges* que possuem a média

harmônica entre os ramos menor que a mediana. Já o grupo *mais arquivos*, representa as correlações entre um tipo de acoplamento e um tipo de esforço de *merge*, referentes ao grupo de *merges* que possuem a média harmônica entre os ramos maior que a mediana.

As Tabelas 4.7, 4.8, 4.9 e 4.10 mostram os projetos selecionados para a análise da influência dos atributos #Arquivos, #Linhas, #*Commits* e #Autores, respectivamente, nas correlações entre os acoplamentos e os esforços de *merge*. Os projetos selecionados estão marcados com **X** e os demais projetos foram descartados por apresentarem correlações insignificantes. Por exemplo, para a tabela 4.7, o projeto *Antlr* foi selecionado para a análise da influência do atributo #Arquivos na correlação entre o acoplamento estrutural e esforço de *merge* de trabalho desperdiçado. No entanto, o projeto *Netty* foi descartado da análise da influência do atributo #Arquivos na correlação entre o acoplamento estrutural e esforço de *merge* de trabalho desperdiçado.

QP4.1: O número de arquivos modificados entre ramos influencia a correlação entre os acoplamentos e os esforços de *merge*?

Em geral, conforme a Figura 4.8, observou-se que os *merges* com *mais arquivos* possuem uma correlação maior entre os acoplamentos e os esforços de *merge* do que os *merges* com *menos arquivos*.

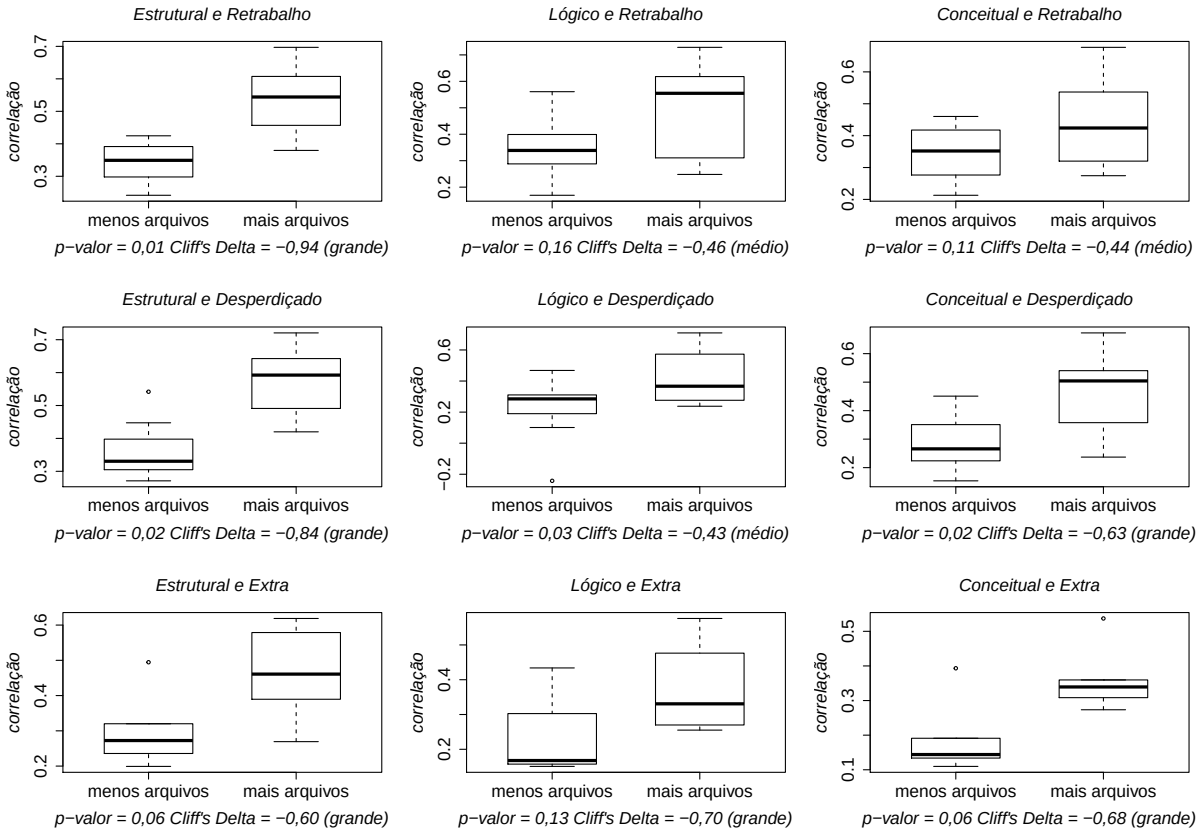


Figura 4.8: *Boxplots* comparando a correlação entre os acoplamentos e os esforços de *merge* de forma agrupada por #Arquivos.

Tabela 4.7: Projetos selecionados para avaliação da influência do atributo #Arquivos.

Projeto	**Retrabalho**			**Desperdiçado**			**Extra**		
	Est	**Log**	**Con**	**Est**	**Log**	**Con**	**Est**	**Log**	**Con**
Antlr	x	x	x	x	x	x	x	x	x
Arduino				x		x			
Cat	x	x	x	x	x	x	x	x	x
Dbeaver			x			x			
Druid	x	x	x	x	x	x			x
Exoplayer	x		x	x	x				
Flink	x	x	x	x	x	x	x	x	x
Incubator	x			x		x	x		
Mockito	x	x	x	x		x	x		
Netty									
RxJava		x		x	x	x		x	x
Zaproxy	x	x			x				

Para os projetos restantes, foi aplicado o teste pareado de *Wilcox* e observou-se uma diferença estatisticamente significante na correlação entre o acoplamento estrutural e o esforço de retrabalho (p-valor = 0,01), e também, na correlação entre o esforço de trabalho

desperdiçado e os acoplamentos estrutural (p-valor = 0,02), lógico (p-valor = 0,03) e conceitual (p-valor = 0,02), com nível de confiança de 95%, conforme pode ser visto na Figura 4.8. No entanto, não foi possível observar diferença estatisticamente significante na correlação entre o esforço de trabalho extra e os acoplamentos estrutural (p-valor = 0,06), lógico (p-valor = 0,13) e conceitual (p-valor = 0,06), bem como na correlação entre o acoplamento lógico e o esforço de retrabalho (p-valor = 0,16), e na correlação entre o acoplamento conceitual e o esforço de trabalho desperdiçado (p-valor = 0,05), com nível de confiança de 95 %.

Para complementar esta análise, foi identificado um tamanho de efeito de *Cliff's Delta* grande para as correlações entre o acoplamento estrutural e os esforços de retrabalho (delta = -0,94) e trabalho desperdiçado (delta = -0,84). De forma equivalente, também foi observado um tamanho de efeito de *Cliff's Delta* grande para a correlação entre o acoplamento conceitual e o esforço de trabalho desperdiçado (delta = -0,63). Além disso, notou-se um tamanho de efeito de *Cliff's Delta* médio para a correlação entre o acoplamento lógico e o esforço de trabalho desperdiçado (delta = -0,43), conforme visto na Figura 4.8.

QP4.1: *O número de arquivos modificados entre ramos influencia a correlação entre os acoplamentos e os esforços de merge?*
Resposta: Em geral, pôde-se observar correlações mais altas para *merges* com mais arquivos modificados, especialmente para a correlação entre acoplamento estrutural e esforços de retrabalho e trabalho desperdiçado, bem como, entre acoplamento conceitual e esforço de trabalho desperdiçado.
Implicações: Os padrões discutidos na QP3 têm uma validade maior para *merges* maiores, onde muitos arquivos são alterados.

QP4.2: O número de linhas modificadas entre ramos influencia a correlação entre os acoplamentos e os esforços de *merge*?

Em geral, conforme a Figura 4.9, observou-se que os *merges* com *mais linhas* possuem uma correlação maior entre os acoplamentos e os esforços de *merge* do que os *merges* com *menos linhas*.

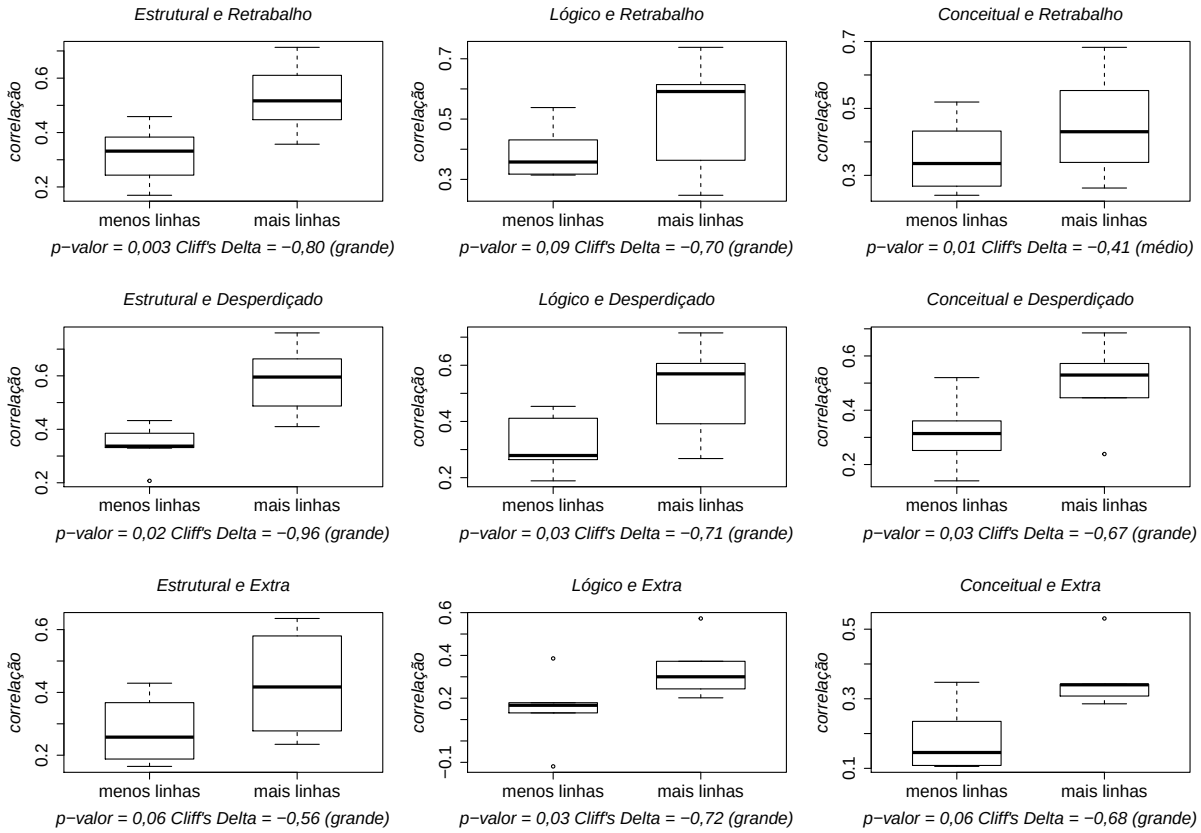


Figura 4.9: *Boxplots* comparando a correlação entre os acoplamentos e os esforços de *merge* de forma agrupada por #Linhas.

Tabela 4.8: Projetos selecionados para avaliação da influência do atributo #Linhas.

Projeto	**Retrabalho**			**Desperdiçado**			**Extra**		
	Est	**Log**	**Con**	**Est**	**Log**	**Con**	**Est**	**Log**	**Con**
Antlr	x	x	x	x	x	x	x	x	x
Arduino									
Cat	x	x	x	x	x	x	x	x	x
Dbeaver	x		x			x			
Druid	x	x	x	x	x	x	x	x	x
Exoplayer	x		x	x					x
Flink	x	x	x	x	x	x	x	x	x
Incubator	x			x			x	x	
Mockito	x	x	x	x	x	x	x		
Netty									
RxJava		x	x		x			x	x
Zaproxy	x								

Para os projetos restantes, foi aplicado o teste pareado de *Wilcox* e notou-se uma diferença estatisticamente significante na correlação entre o acoplamento estrutural e os esforços de retrabalho (p-valor = 0,003) e trabalho desperdiçado (p-valor = 0,02), e tam-

bém, na correlação entre o acoplamento lógico e os esforços de trabalho desperdiçado (p-valor = 0,03) e trabalho extra (p-valor = 0,03). Igualmente, observou-se uma diferença estatisticamente significante na correlação entre o acoplamento conceitual e os esforços de retrabalho (p-valor = 0,01) e trabalho desperdiçado (p-valor = 0,03). No entanto, não foi observada uma diferença estatisticamente significante na correlação entre o esforço de trabalho extra e os acoplamentos estrutural (p-valor = 0,06) e conceitual (p-valor = 0,06), e na correlação entre o acoplamento lógico e o esforço de retrabalho, como pode ser visto na Figura 4.9.

Para complementar a análise, identificou-se um tamanho de efeito de *Cliff's Delta* grande para a correlação entre o acoplamento estrutural e os esforços de retrabalho (delta = -0,80) e trabalho desperdiçado (delta = -0,96), como também, para a correlação entre o acoplamento lógico e os esforços de trabalho desperdiçado (delta = 0,71) e trabalho extra (delta = -0,72). Além disso, observou-se um tamanho de efeito de *Cliff's Delta* grande para a correlação entre o acoplamento conceitual e o esforço de trabalho desperdiçado (delta = -0,67). No entanto, notou-se um tamanho de efeito de *Cliff's Delta* médio para a correlação entre o acoplamento conceitual e o esforço de retrabalho (delta = -0,41). conforme descrito na Figura 4.9.

QP4.2: *O número de linhas modificadas entre ramos influencia a correlação entre os acoplamentos e os esforços de merge?*
Resposta: O atributo #Linhas influencia a correlação entre o acoplamento estrutural e os esforços de retrabalho e trabalho desperdiçado. De forma equivalente, influencia também, a correlação entre o acoplamento lógico e os esforços de trabalho desperdiçado e trabalho extra. Finalmente, o atributo #Linhas influencia a correlação entre o acoplamento conceitual e o esforço de trabalho desperdiçado.
Implicações: Os resultados corroboram os do atributo #Arquivos (QP4.1), indicando que os padrões observados neste trabalho são válidos, em especial para *merges* maiores, ou seja, para *merges* que possuem maior quantidade de linhas alteradas.

QP4.3: O número de *commits* entre ramos influencia a correlação entre os acoplamentos e os esforços de *merge*?

Em geral, observou-se que os *merges* com *mais commits* possuem uma correlação maior entre os acoplamentos e os esforços de *merge* do que os *merges* com *menos commits*.

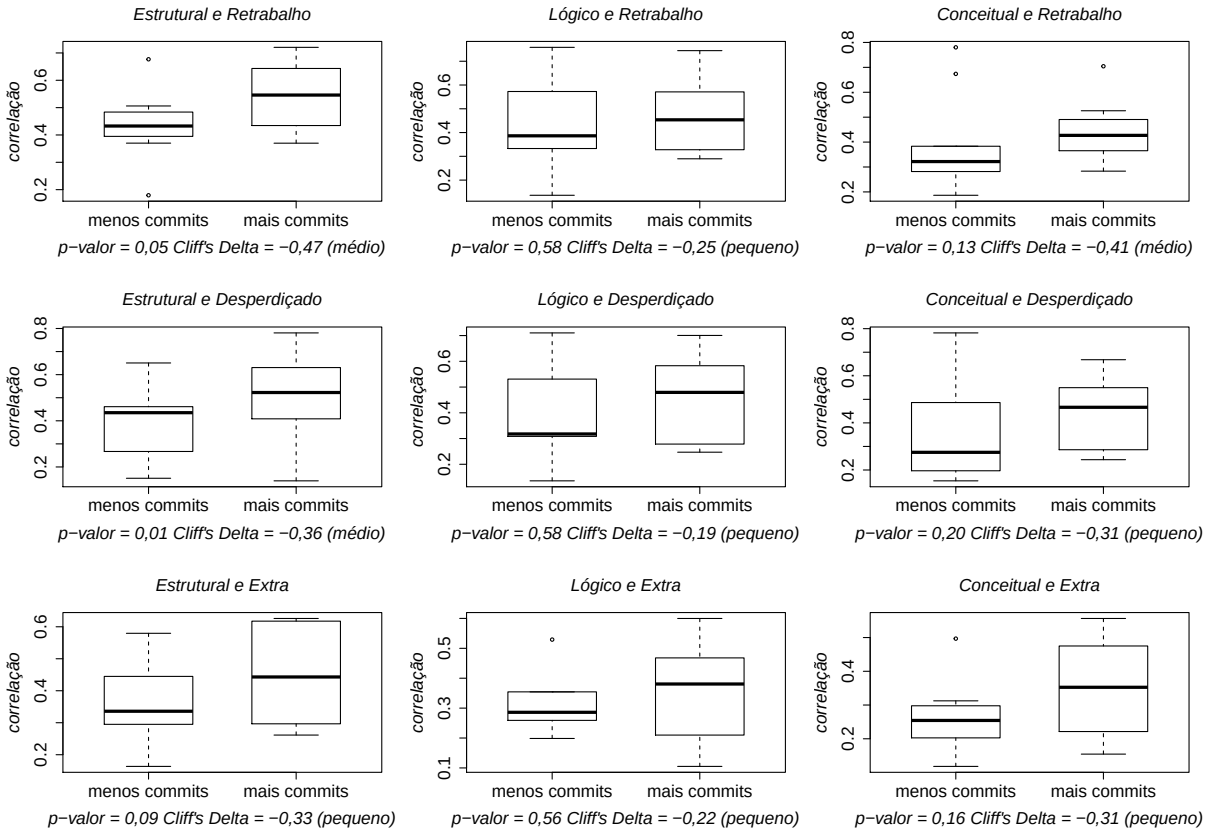


Figura 4.10: *Boxplots* comparando a correlação entre os acoplamentos e os esforços de *merge* de forma agrupada por *#Commits*.

Tabela 4.9: Projetos selecionados para avaliação da influência do atributo *#Commits*.

Projeto	**Retrabalho**			**Desperdiçado**			**Extra**		
	Est	**Log**	**Con**	**Est**	**Log**	**Con**	**Est**	**Log**	**Con**
Antlr	x	x	x	x	x	x	x	x	x
Arduino									
Cat	x	x	x	x	x	x	x	x	x
Dbeaver	x		x			x			
Druid	x	x	x	x	x	x	x	x	x
Exoplayer			x	x					
Flink	x	x	x	x	x	x	x	x	x
Incubator	x	x	x	x	x	x	x		x
Mockito	x	x	x	x	x	x	x	x	x
Netty	x								
RxJava		x	x		x	x		x	x
Zaproxy									

Para os projetos restantes, foi aplicado o teste de *Wilcoxon* e percebeu-se uma diferença estatisticamente significativa somente entre o acoplamento estrutural e o esforço desperdiçado (p-valor = 0,01). Para as demais correlações percebeu-se uma diferença es-

tatisticamente não significante. Cabe ressaltar que, para a correlação entre o acoplamento estrutural e o esforço de retrabalho foi identificada uma diferença insignificante, com p-valor $>$ 0,05 (p-valor = 0,05469). Em complemento ao teste de hipóteses, foi identificado um tamanho de efeito de *Cliff's Delta* médio entre o acoplamento estrutural e o esforço de trabalho desperdiçado (delta = -0,36), conforme pode ser visto na Figura 4.10.

QP4.3: *O número de commits entre ramos influencia a correlação entre os acoplamentos e os esforços de merge?*
Resposta: O número de *commits* entre ramos influencia a correlação entre o acoplamento estrutural e o esforço de trabalho desperdiçado.
Implicações: Como discutido anteriormente, a complexidade das mudanças fortalece os padrões observados neste trabalho (QP4.1 e QP4.2), entretanto, a dispersão dessas mudanças em vários *commits* não é relevante na maioria dos casos, exceto para a correlação entre o acoplamento estrutural e o esforço de trabalho desperdiçado.

QP4.4: O número de autores entre ramos influencia a correlação entre os acoplamentos e os esforços de *merge*?

Observou-se que os *merges* com *mais autores* ou *menos autores* não influenciam a correlação entre os acoplamentos e os esforços de *merge*.

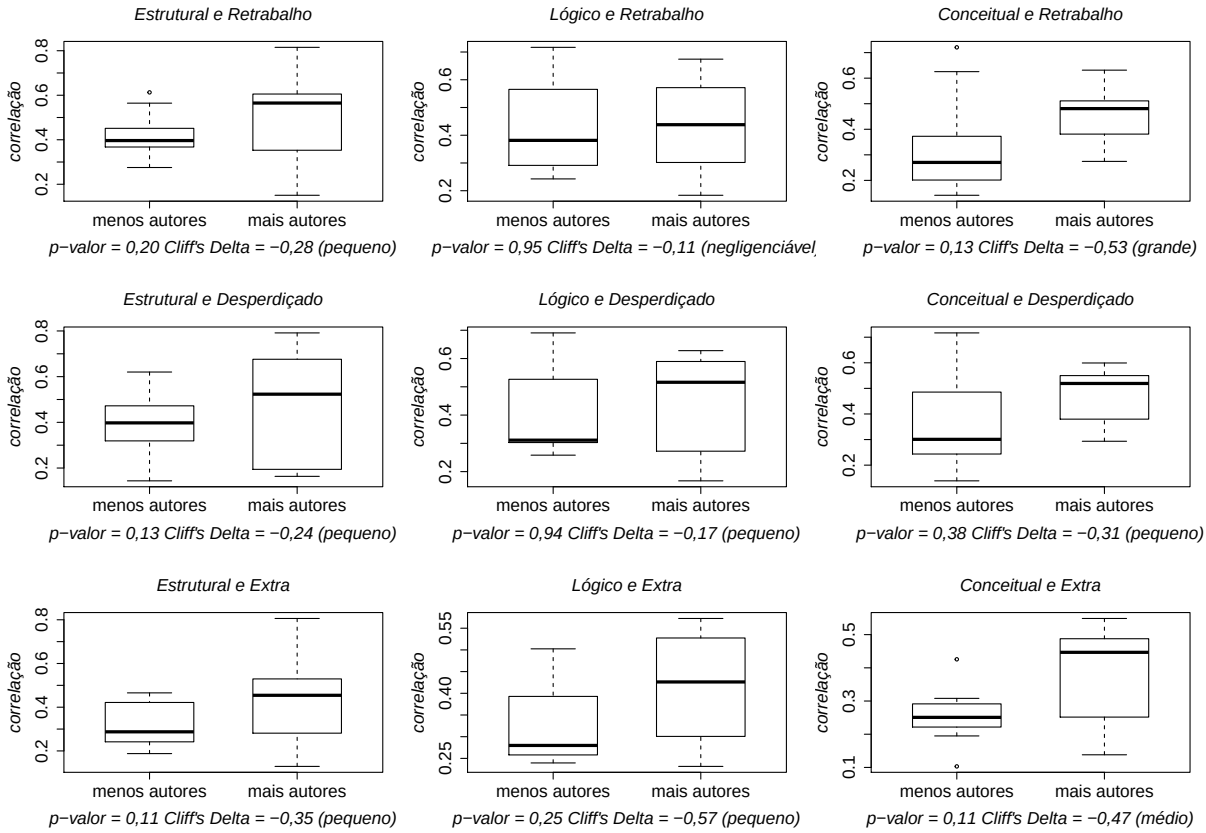


Figura 4.11: *Boxplots* comparando a correlação entre os acoplamentos e os esforços de *merge* de forma agrupada por #Autores.

Tabela 4.10: Projetos selecionados para avaliação da influência do atributo #Autores.

Projeto	**Retrabalho**			**Desperdiçado**			**Extra**		
	Est	**Log**	**Con**	**Est**	**Log**	**Con**	**Est**	**Log**	**Con**
Antlr	x	x	x	x	x	x	x	x	x
Arduino									
Cat	x	x	x	x	x	x	x	x	x
Dbeaver	x		x						
Druid	x	x	x	x	x	x	x		x
Exoplayer		x							
Flink	x	x	x	x	x	x	x	x	x
Incubator	x		x		x	x		x	x
Mockito	x	x	x	x	x	x	x	x	x
Netty	x								
RxJava		x	x	x	x	x	x	x	x
Zaproxy	x	x		x			x		

Para os projetos restantes, foi aplicado o teste de *Wilcoxon* e identificou-se uma diferença estatisticamente não significante para todas as correlações, de acordo com a Figura 4.11.

QP4.4: *O número de autores entre ramos influencia a correlação entre os acoplamentos e os esforços de merge?*
Resposta: Diferentemente dos demais atributos dos ramos, o atributo #Autores não influencia a correlação entre acoplamentos e esforços de *merge*.
Implicações: Os resultados corroboram os do atributo *#Commits* (QP4.3), indicando que os padrões encontrados neste trabalho estão relacionados a fatores intrínsecos à mudança.

Cabe ressaltar que, foram identificados tamanho de efeito médio ou grande e p-valor insignificante para algumas correlações entre acoplamentos e esforços de *merge*. Isso pode ter ocorrido devido ao pequeno número de amostras analisadas.

4.4 Ameaças à Validade

Embora o objetivo tenha sido minimizar as ameaças à validade desse estudo, alguns fatores podem ter influenciado os resultados desse estudo.

Uma ameaça potencial à validade de construção refere-se ao cálculo do acoplamento lógico que pode ser impreciso para os primeiros *merges* do projeto, pois eles possuem pouco histórico de *commits*. Da mesma forma, refatorações ao longo do histórico podem remover acoplamentos entre métodos, gerando falsos positivos para o acoplamento lógico. Outra ameaça à validade de construção diz respeito à maneira como o desenvolvedor faz o esforço de trabalho extra quando não há conflitos diretos. O desenvolvedor pode adiar a resolução do conflito indireto, mantendo o *commit* de *merge* original e criando um novo *commit* subsequente com os ajustes. No entanto, o *commit* de *merge* foi analisado sem considerar os *commits* subsequentes. Contudo, Menezes et al. [34] estudaram esses *commits* subsequentes e descobriram que eles raramente mudam o código-fonte modificado na resolução de *commits* de *merge*. Portanto, a probabilidade de perder ajustes de *merge* em *commits* subsequentes é baixa. Além disso, outra ameaça diz respeito à extensão em que a métrica absoluta representa corretamente os acoplamentos entre os ramos. Para mitigar essa ameaça, foi calculada uma métrica normalizada. No entanto, notou-se que as correlações obtidas a partir da métrica normalizada, são frequentemente menores do que as correlações obtidas a partir da métrica absoluta. Além disso, observou-se uma correlação muito forte entre as duas métricas. Consequentemente, optou-se por utilizar apenas a métrica absoluta.

Uma potencial ameaça à validade interna refere-se ao fato de que foi indicada a influência dos atributos nas correlações entre acoplamentos e esforços de *merge*. Embora tenham sido adotados testes estatísticos para avaliar tal influência, outros fatores não analisados nesse estudo podem ter influenciado tanto os atributos como as correlações.

Uma potencial ameaça à validade externa é a seleção de projetos na linguagem Java. Os estudos concentraram-se em apenas uma linguagem de programação, porque a análise dos acoplamentos estrutural, lógico e conceitual entre os métodos é dependente da linguagem, conforme discutido na Seção 4.2. Assim, não é possível garantir que os resultados observados sejam válidos para projetos em outras linguagens de programação. Além disso, foram analisados poucos *merges* de poucos projetos devido à grande demanda de processamento e tempo para extrair os acoplamentos lógico e conceitual. Como exemplo, se a extração do acoplamento estrutural de um projeto durou em torno de 6 horas, então a extração dos acoplamentos lógico e conceitual durou em torno de 180 horas e 18 horas, respectivamente. Consequentemente, os resultados podem variar se um número maior de *merges* de vários projetos for analisado. Além disso, os resultados desse estudo são aplicáveis apenas para projetos de código aberto, uma vez que os projetos de *software* da indústria podem ter características que não foram consideradas nessas análises.

Por último, uma ameaça potencial à validade de conclusão refere-se à avaliação da normalidade dos dados que não foi realizada. Devido à pequena amostra desse estudo (doze projetos), optou-se por utilizar testes não paramétricos independentemente da observância da normalidade. No entanto, alguns autores Warner [57] apoiam esta decisão.

4.5 Considerações Finais

Neste capítulo, foram apresentados os materiais e métodos e os resultados das quatro questões de pesquisa. As QP1 e QP2 tem a finalidade de avaliar a relação entre os acoplamentos, e também, entre os esforços de *merge*. A QP3 tem o objetivo de entender se há correlação entre os acoplamentos e os esforços de *merge*. Já a QP4 visa entender a influência de alguns atributos dos ramos na correlação entre os acoplamentos e os esforços de *merge*.

Para a QP1 foi calculada a correlação entre os acoplamentos e foi observada uma correlação moderada entre os acoplamentos lógico e conceitual. Além disso, foi observada uma correlação fraca entre os acoplamentos estrutural e lógico, assim como, entre os acoplamentos estrutural e conceitual. Para a QP2, foi calculada a correlação entre os

esforços de *merge* e foi observada uma correlação forte entre os esforços de retrabalho e trabalho desperdiçado. No entanto, foi identificada uma correlação fraca entre o esforço extra e os outros dois esforços de *merge*. Já para a QP3, na análise da correlação, variou-se o *threshold* de zero a um, em passos de 0,1, somente para as métricas de acoplamento lógico e conceitual. Então, foi calculada a correlação entre acoplamentos e esforços de *merge* para cada *threshold* e foi observado que geralmente, à medida que o *threshold* aumenta, a correlação também aumenta. Portanto, a partir desta análise, é possível notar que há uma correlação moderada entre os acoplamentos estrutural e conceitual e os esforços de retrabalho e trabalho desperdiçado. No entanto, há uma correlação fraca entre o acoplamento lógico e os esforços de retrabalho e trabalho desperdiçado. Além disso, foi identificada uma correlação fraca entre todas as métricas de acoplamento e o esforço de trabalho extra. Para a QP4, foi observado que o número de arquivos modificados e o número de linhas modificadas influenciam algumas correlações entre acoplamentos e esforços de *merge*. Além disso, o número de *commits* influencia somente a correlação entre o acoplamento estrutural e o esforço de trabalho desperdiçado. No entanto, o número de autores não foi estatisticamente significativo nesta análise.

As respostas das análises fornecem informações importantes para auxiliar os construtores de ferramentas a conceber melhores ferramentas de percepção para predizer *merges* difíceis. Como também, pode ajudar os gerentes de projeto a designar melhor os desenvolvedores para tarefas ou os desenvolvedores a sincronizarem entre si, visando reduzir o esforço de *merge*.

Cabe ressaltar que, Oliva e Gerosa [38] concluíram que o acoplamento lógico não envolve artefatos estruturalmente relacionados e Ajienka e Capiluppi [3] identificaram que não há correlação significativa entre os acoplamentos estrutural e lógico entre as classes. Portanto, essas descobertas não estão de acordo com os resultados apresentados, pois foi identificada uma grande sobreposição direcional entre os acoplamentos estrutural e lógico entre ramos (85%), e 22% dos *merges* com acoplamento lógico entre ramos, também possuem acoplamento estrutural entre ramos. Além disso, foi identificada uma correlação fraca entre os acoplamentos estrutural e lógico entre ramos. Por outro lado, Ajienka, Capiluppi e Counsell [4] identificaram uma relação bidirecional entre os acoplamentos conceitual e lógico e por isso, está de acordo com os resultados apresentados neste estudo. Tendo em vista que foi identificado que 62% dos *merges* com acoplamento conceitual também possuem acoplamento lógico, assim como, há uma grande sobreposição direcional entre os acoplamentos lógico e conceitual.

No entanto, deve-se contrastar com cautela os estudos da literatura com os resultados apresentados. Esses estudos detectam acoplamentos dentro de uma versão do repositório, analisando todas as classes de um projeto em um momento específico, para identificar os acoplamentos entre elas. Por outro lado, neste estudo foram analisados acoplamentos entre alterações feitas em paralelo em diferentes ramos.

Capítulo 5

Conclusão

5.1 Contribuições

Neste trabalho, foram apresentados alguns conceitos sobre acoplamento de *software* e esforços de *merge*, e foram realizadas algumas análises referentes à correlação entre acoplamentos e esforços de *merge*. Portanto, as principais contribuições deste estudo são:

(1) a avaliação da interação entre os acoplamentos entre ramos, onde identificou-se que a correlação entre os acoplamentos lógico e conceitual é moderada, e é fraca entre os acoplamentos estrutural e lógico, bem como entre os acoplamentos estrutural e conceitual. Observou-se que há uma grande sobreposição direcional entre o acoplamento estrutural e os acoplamentos lógico e conceitual, bem como entre os acoplamentos lógico e conceitual;

(2) a quantificação do grau em que as alterações realizadas em paralelo são acopladas, de acordo com três diferentes métricas: estrutural, lógica e conceitual;

(3) a avaliação da interação entre os esforços de *merge*, onde foi observada uma correlação forte entre os esforços de retrabalho e trabalho desperdiçado, e uma correlação moderada entre os esforços de trabalho extra e os esforços de retrabalho e de trabalho desperdiçado. Além disso, conclui-se que há uma grande sobreposição direcional entre os esforços de *merge*, exceto entre os esforços de retrabalho e trabalho desperdiçado e o esforço de trabalho extra;

(4) a avaliação da correlação entre as métricas de acoplamento das mudanças realizadas entre ramos e os esforços de *merge*, onde pôde-se observar uma correlação moderada entre os acoplamentos estrutural e conceitual, e os esforços de retrabalho e trabalho desperdiçado. No entanto, foi encontrada uma fraca correlação entre o acoplamento lógico e os esforços de retrabalho e de trabalho desperdiçado. Além disso, foi identificada uma

fraca correlação entre todas as métricas de acoplamento e o esforço de trabalho extra. Contudo, alinhada à literatura, a correlação entre o acoplamento estrutural e o esforço de trabalho extra foi maior dentre os três tipos de acoplamento; e

(5) a investigação das características dos ramos para identificar se elas influenciam a correlação entre acoplamentos e esforços de *merge*. Observou-se que, o número de arquivos modificados influencia as correlações entre o acoplamento estrutural e os esforços de retrabalho e de trabalho desperdiçado, e as correlações entre o acoplamento conceitual e o esforço de trabalho desperdiçado, com tamanho de efeito grande. Além disso, o número de arquivos modificados influencia a correlação entre o acoplamento lógico e o esforço de trabalho desperdiçado, com tamanho de efeito médio. Por outro lado, o número de linhas modificadas entre ramos influencia a correlação entre o acoplamento estrutural e os esforços de retrabalho e de trabalho desperdiçado, assim como, a correlação entre o acoplamento lógico e os esforços de trabalho desperdiçado e de trabalho extra, com tamanho de efeito grande. Além do mais, o número de linhas modificadas influencia também a correlação entre o acoplamento conceitual e o esforço de trabalho desperdiçado, com tamanho de efeito grande. Contudo, o número de linhas modificadas influencia a correlação entre o acoplamento conceitual e o esforço de retrabalho, com tamanho de efeito médio. Por fim, o número de *commits* influencia somente a correlação entre o acoplamento estrutural e o esforço de trabalho desperdiçado, com tamanho de efeito médio, e o número de autores não influencia a correlação entre acoplamentos e esforços de *merge*.

5.2 Trabalhos Futuros

Como dito anteriormente, este estudo se limitou a compreender a interação entre os acoplamentos entre ramos, a relação entre os esforços de *merge*, a correlação entre os acoplamentos e os esforços de *merge* e as características dos ramos que podem influenciar esta correlação. Então, de forma a aprimorar e ampliar o estudo realizado, sugere-se alguns trabalhos futuros. Primeiramente, sugere-se executar as análises em um número maior de projetos, a fim de identificar se uma amostra maior de projetos corrobora as descobertas encontradas, principalmente com relação ao estudo da influência das características dos ramos nas correlações. Visto que, devido ao número reduzido de projetos ($N = 12$), a análise pode ter sido comprometida. Além disso, pretende-se executar novas análises em projetos desenvolvidos em outras linguagens de programação, como Python ou C++, tendo em vista que também são linguagens de programação popularmente utilizadas [1].

[1] https://www.tiobe.com/tiobe-index/

O objetivo é verificar se os resultados obtidos neste estudo se mantêm mesmo em outras linguagens de programação e os resultados das análises podem auxiliar os desenvolvedores nestas linguagens.

Além disso, pretende-se inspecionar manualmente os projetos em que não foi encontrada relação de influência dos atributos nas correlações, por exemplo, os projetos *Arduino* e *Netty*. Como também, remover este projetos da análise da influência dos atributos e executar novamente a análise, para verificar se há significância estatística para os outros casos.

Outro ponto importante é realizar uma análise mais aprofundada sobre os autores dos *commits*, como também avaliar a influência de métricas sociais nas correlações entre acoplamentos e esforços de *merge*, como a experiência do desenvolvedor no projeto. Assim como, quantificar o número de desenvolvedores diferentes que contribuíram para o projeto a fim de avaliar se esta quantidade exerce alguma influência nos acoplamentos entre ramos. Além do mais, verificar se existe alguma correlação entre as características dos ramos e realizar uma regressão, a fim de avaliar o fator de confusão entre essas características.

Outra possibilidade de trabalho futuro consiste em utilizar o teste não paramétrico de *Kruskall-Wallis* para comparar as amostras entre os acoplamentos e os esforços de *merge*, variando por *threshold*, a fim de identificar se há uma diferença estatística entre as amostras, e calcular o tamanho de efeito para avaliar o tamanho da diferença. Além disso, investigar o motivo pelo qual a correlação entre os acoplamentos lógico e conceitual não aumentou quando o *threshold* é 1.

Outro campo de pesquisa interessante é o de testes de *software*. Alguns estudos [7, 23] mostram que as técnicas dos testes de integração de *software* são baseadas no acoplamento estrutural entre as entidades do código. Portanto, pretende-se estudar os acoplamentos lógico e conceitual em casos de teste de integração de software, com o intuito de identificar outras dependências entre as entidades que não são identificadas somente com a análise do acoplamento estrutural. Com isso, poderia auxiliar a equipe de testes na elaboração de planos de testes com base em outros tipos de acoplamento.

Finalmente, pretende-se incluir nesse estudo outro tipo de acoplamento, denominado *acoplamento de uso*. Este tipo de acoplamento existe quando as entidades são frequentemente utilizadas juntas e podem ser identificadas por meio da mineração de sequência sobre os corpos de método [24, 54]. Com isso, a partir da identificação do *acoplamento de uso*, é sugerido um trecho de código para ser utilizado no método que está sendo codificado.

Bibliografia

[1] Rakesh Agrawal, Ramakrishnan Srikant et al. "Fast algorithms for mining association rules". Em: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. 1994, pp. 487–499.

[2] Nemitari Ajienka e Andrea Capiluppi. "Semantic coupling between classes: Corpora or identifiers?" Em: *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM. London: ACM, 2016, p. 40.

[3] Nemitari Ajienka e Andrea Capiluppi. "Understanding the interplay between the logical and structural coupling of software classes". Em: *Journal of Systems and Software* 134 (2017), pp. 120–137.

[4] Nemitari Ajienka, Andrea Capiluppi e Steve Counsell. "An empirical study on the interplay between semantic coupling and co-change of software classes". Em: *Empirical Software Engineering* 23.3 (2018), pp. 1791–1825.

[5] Nemitari Ajienka, Andrea Capiluppi e Steve Counsell. "Managing hidden dependencies in OO software: a study based on open source projects". Em: *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE. 2017, pp. 141–150.

[6] Mamdouh Khalaf Alenezi. "A New Coupling Metric: Combining Structural and Semantic Relationships". Tese de doutoramento. North Dakota State University, 2014.

[7] Roger T Alexander, Jeff Offutt e Andreas Stefik. "Testing coupling relationships in object-oriented programs". Em: *Software Testing, Verification and Reliability* 20.4 (2010), pp. 291–327.

[8] Stephen P. Berczuk e Brad Appleton. *Software Configuration Management Patterns: Effective Teamwork, Practical Integration*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN: 0201741172.

[9] Christian Bird e Thomas Zimmermann. "Assessing the Value of Branches with What-if Analysis". Em: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. FSE '12. Cary, North Carolina: ACM, 2012, 45:1–45:11. ISBN: 978-1-4503-1614-9. DOI: 10.1145/2393596.2393648. URL: http://doi.acm.org/10.1145/2393596.2393648.

[10] Christian Bird, Thomas Zimmermann e Alex Teterev. "A theory of branches as goals and virtual teams". Em: *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*. ACM. 2011, pp. 53–56.

[11] Lionel C. Briand, John W. Daly e Jurgen K Wust. "A unified framework for coupling measurement in object-oriented systems". Em: *IEEE Transactions on software Engineering* 25.1 (1999), pp. 91–121.

[12] Lionel C Briand, Jurgen Wust e Hakim Lounis. "Using coupling measurement for impact analysis in object-oriented systems". Em: *Proceedings IEEE International Conference on Software Maintenance-1999 (ICSM'99).'Software Maintenance for Business Change'(Cat. No. 99CB36360)*. IEEE. Germany, Canada: IEEE, 1999, pp. 475–482.

[13] Yuriy Brun et al. "Proactive detection of collaboration conflicts". Em: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 2011, pp. 168–178.

[14] Marcelo Cataldo et al. "Software dependencies, work dependencies, and their impact on failures". Em: *IEEE Transactions on Software Engineering* 35.6 (2009), pp. 864–878.

[15] Catarina Costa et al. "TIPMerge: recommending experts for integrating changes across branches". Em: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM. Niteroi, Brazil: ACM, 2016, pp. 523–534.

[16] Marco D'Ambros. "Supporting software evolution analysis with historical dependencies and defect information". Em: *2008 IEEE International Conference on Software Maintenance*. IEEE. University of Lugano, Switzerland: IEEE, 2008, pp. 412–415.

[17] Prasun Dewan e Rajesh Hegde. "Semi-synchronous conflict detection and resolution in asynchronous software development". Em: *ECSCW 2007*. Springer, 2007, pp. 159–178.

[18] Harald Gall, Karin Hajek e Mehdi Jazayeri. “Detection of logical coupling based on product release history”. Em: *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*. IEEE. Austria, Europe: IEEE, 1998, pp. 190–198.

[19] Malcom Gethers, Amir Aryani e Denys Poshyvanyk. “Combining conceptual and domain-based couplings to detect database and code dependencies”. Em: *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation*. IEEE. USA, Australia: IEEE, 2012, pp. 144–153.

[20] Gui Gui e Paul D Scott. “Coupling and cohesion measures for evaluation of component reusability”. Em: *Proceedings of the 2006 international workshop on Mining software repositories*. ACM. ACM, 2006, pp. 18–21.

[21] Youssef Hassoun, Steve Counsell e Roger Johnson. “Dynamic coupling metric: proof of concept”. Em: *IEE Proceedings-Software* 152.6 (2005), pp. 273–279.

[22] Dennis E Hinkle, William Wiersma e Stephen G Jurs. *Applied statistics for the behavioral sciences*. Vol. 663. Houghton Mifflin College Division, 2003.

[23] Zhenyi Jin e A Jefferson Offutt. “Coupling-based criteria for integration testing”. Em: *Software Testing, Verification and Reliability* 8.3 (1998), pp. 133–154.

[24] Luiz Laerte Nunes Silva Junior et al. “Vertical code completion: Going beyond the current ctrl+ space”. Em: *2012 Sixth Brazilian Symposium on Software Components, Architectures and Reuse*. IEEE. 2012, pp. 81–90.

[25] Huzefa Kagdi, Malcom Gethers e Denys Poshyvanyk. “Integrating conceptual and logical couplings for change impact analysis in software”. Em: *Empirical Software Engineering* 18.5 (2013), pp. 933–969.

[26] Huzefa Kagdi et al. “Blending conceptual and evolutionary couplings to support change impact analysis in source code”. Em: *2010 17th Working Conference on Reverse Engineering*. IEEE. Winston-Salem, Williamsburg, Akron: IEEE, 2010, pp. 119–128.

[27] Bakhtiar Khan Kasi e Anita Sarma. “Cassandra: Proactive conflict minimization through optimized task scheduling”. Em: *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press. Lincoln, NE, USA: IEEE, 2013, pp. 732–741.

[28] Noureddine Kerzazi e Foutse Khomh. "Factors impacting rapid releases: an industrial case study". Em: *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM. 2014, p. 61.

[29] Jonathan Knudsen e John Director-Zukowski. *Wireless Java: developing with J2ME*. APress LP, 2003.

[30] Hoai Le Nguyen e Claudia-Lavinia Ignat. "An Analysis of Merge Conflicts and Resolutions in Git-Based Open Source Projects". Em: *Computer Supported Cooperative Work (CSCW)* 27.3-6 (2018), pp. 741–765.

[31] Olaf Leßenich et al. "Indicators for merge conflicts in the wild: survey and empirical study". Em: *Automated Software Engineering* 25.2 (2018), pp. 279–313.

[32] Gleydson de Azevedo Ferreira Lima. "Uma abordagem para evolução e reconciliação de linhas de produtos de software clonadas". Em: (2014), pp. 1–186.

[33] Shane McKee et al. "Software practitioner perspectives on merge conflicts and resolutions". Em: *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE. 2017, pp. 467–478.

[34] Gleiph Ghiotto Lima Menezes et al. "On the Nature of Merge Conflicts: a Study of 2,731 Open Source Java Projects Hosted by GitHub". Em: *IEEE Transactions on Software Engineering* (2018), pp. 1–25.

[35] Tom Mens. "A state-of-the-art survey on software merging". Em: *IEEE transactions on software engineering* 28.5 (2002), pp. 449–462.

[36] Manishankar Mondal, Chanchal K Roy e Kevin A Schneider. "Insight into a method co-change pattern to identify highly coupled methods: An empirical study". Em: *2013 21st International Conference on Program Comprehension (ICPC)*. IEEE. 2013, pp. 103–112.

[37] Tayane Silva Fernandes de Moura e Leonardo Gresta Paulino Murta. "Uma técnica para a quantificação do esforço de merge". Em: *arXiv preprint arXiv:1811.09176* (2018), pp. 1–10.

[38] Gustavo Ansaldi Oliva e Marco Aurelio Gerosa. "On the interplay between structural and logical dependencies in open-source software". Em: *2011 25th Brazilian Symposium on Software Engineering*. IEEE. São Paulo, Brazil: IEEE, 2011, pp. 144–153.

[39] Moein Owhadi-Kareshk, Sarah Nadi e Julia Rubin. "Predicting Merge Conflicts in Collaborative Software Development". Em: *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE. 2019, pp. 1–11.

[40] Linda Dailey Paulson. "Developers shift to dynamic programming languages". Em: *Computer* 40.2 (2007), pp. 12–15.

[41] Dewayne E Perry, Harvey P Siy e Lawrence G Votta. "Parallel changes in large-scale software development: an observational case study". Em: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 10.3 (2001), pp. 308–337.

[42] Denys Poshyvanyk e Andrian Marcus. "The conceptual coupling metrics for object-oriented systems". Em: *2006 22nd IEEE International Conference on Software Maintenance*. IEEE. Detroit Michigan: IEEE, 2006, pp. 469–478.

[43] Denys Poshyvanyk et al. "Using information retrieval based coupling measures for impact analysis". Em: *Empirical software engineering* 14.1 (2009), pp. 5–32.

[44] Roger Pressman e Bruce Maxim. *Engenharia de Software-8ª Edição*. New York: McGraw Hill Brasil, 2016.

[45] João Gustavo Prudêncio, Leonardo Murta e Cláudia Werner. "On the Selection of Concurrency Control Policies for Configuration Management". Em: *XXIII Brazilian Symposium on Software Engineering*. 2009, pp. 155–164.

[46] João Gustavo Prudêncio et al. "To lock, or not to lock: That is the question". Em: *Journal of Systems and Software* 85.2 (2012), pp. 277–289.

[47] Romain Robbes, Damien Pollet e Michele Lanza. "Logical coupling based on fine-grained change information". Em: *2008 15th Working Conference on Reverse Engineering*. IEEE. University of Lugano, Switzerland: IEEE, 2008, pp. 42–46.

[48] Jeanine Romano et al. "Exploring methods for evaluating group differences on the NSSE and other surveys: Are the t-test and Cohen'sd indices the most appropriate choices". Em: *annual meeting of the Southern Association for Institutional Research*. Citeseer. 2006, pp. 1–51.

[49] R. d. S. Santos e L. G. P. Murta. "Evaluating the Branch Merging Effort in Version Control Systems". Em: *26th Brazilian Symposium on Software Engineering*. 2012, pp. 151–160. DOI: 10.1109/SBES.2012.16.

[50] R Santos e L Murta. "Evaluating the Branch Merging Effort in Version Control Systems". Em: *2012 26th Brazilian Symposium on Software Engineering*. 2012, pp. 151–160.

[51] Anita Sarma, Gerald Bortis e André Van Der Hoek. "Towards supporting awareness of indirect conflicts across software configuration management workspaces". Em: *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM. University of California, Irvine: ACM, 2007, pp. 94–103.

[52] Anita Sarma, David F Redmiles e Andre Van Der Hoek. "Palantir: Early detection of development conflicts arising from parallel code changes". Em: *IEEE Transactions on Software Engineering* 38.4 (2011), pp. 889–908.

[53] Jose Ricardo da Silva Junior et al. "Dominoes: An Interactive Exploratory Data Analysis tool for Software Relationships". Em: *IEEE Transactions on Software Engineering* (2020).

[54] Luiz Laerte Nunes da Silva Junior, Alexandre Plastino e Leonardo Gresta Paulino Murta. "What Should I Code Now?" Em: *J. UCS* 20.5 (2014), pp. 797–821.

[55] Charles Spearman. "Correlation calculated from faulty data". Em: *British Journal of Psychology, 1904-1920* 3.3 (1910), pp. 271–295.

[56] Bela Ujhazi et al. "New conceptual coupling and cohesion metrics for object-oriented systems". Em: *2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*. IEEE. Hungary, USA: IEEE, 2010, pp. 33–42.

[57] Rebecca M Warner. *Applied statistics: From bivariate through multivariate techniques*. Sage Publications, 2012.

[58] Zhifeng Yu e Václav Rajlich. "Hidden dependencies in program comprehension and change propagation". Em: *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001*. IEEE. 2001, pp. 293–299.

[59] Thomas Zimmermann et al. "Mining version histories to guide software changes". Em: *IEEE Transactions on Software Engineering* 31.6 (2005), pp. 429–445.