Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Bacharelado em Ciência da Computação

Uma análise de alterações externas em conflitos de merge

João Pedro de Carvalho Lima

JUIZ DE FORA
SETEMBRO, 2025

Uma análise de alterações externas em conflitos de merge

João Pedro de Carvalho Lima

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientador: Gleiph Ghiotto Lima De Menezes
Coorientador: Heleno de Souza Campos Junior

JUIZ DE FORA
SETEMBRO, 2025

UMA ANÁLISE DE ALTERAÇÕES EXTERNAS EM CONFLITOS DE MERGE

João Pedro de Carvalho Lima

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

Gleiph Ghiotto Lima De Menezes
Doutor em Computação

Heleno de Souza Campos Junior
Doutor em Computação

André Luiz de Oliveira
Doutor em Ciência da Computação

Fabricio Martins Mendonça
Doutor em Ciência da Informação

JUIZ DE FORA

16 DE SETEMBRO, 2025

Aos meus amigos e irmãos.

Aos pais, pelo apoio e sustento.

Resumo

O desenvolvimento de software moderno depende cada vez mais da programação concorrente para lidar com a crescente complexidade das bases de código. Embora os sistemas de controle de versão atuais ofereçam suporte à colaboração por meio de rastreamento de mudanças e mecanismos de *merge*, os conflitos de *merge* ainda representam um desafio significativo — ocorrendo em 10% a 20% dos *merges*, especialmente quando modificações sobrepostas afetam a mesma região de um arquivo. Pesquisas anteriores concentraram-se na análise, prevenção, predição ou resolução automática desses conflitos. Entretanto, tais abordagens desconsideram modificações realizadas fora das regiões de conflito, o que pode resultar em uma subestimação da real complexidade envolvida na resolução de conflitos de merge, conforme já discutido na literatura especializada. Este trabalho busca preencher essa lacuna ao investigar o papel das modificações não conflitantes no processo de resolução de conflitos. Para isso, foram analisados 80 repositórios, abrangendo oito linguagens de programação — C, C#, C++, Java, JavaScript, PHP, Python e Ruby — com o objetivo de compreender melhor a relevância das alterações não conflitantes durante a resolução de conflitos. Nos dados analisados, descobriu-se que, em média, 28,05% dos *merges* em conflito dos projetos analisados possuem alterações fora das regiões conflito durante o processo de resolução.

Palavras-chave: Git, sistemas de controle de versão, merge de software, conflitos de merge, alterações.

Abstract

Modern software development increasingly relies on concurrent programming to handle the growing complexity of codebases. While current version control systems support collaboration through change tracking and merge mechanisms, merge conflicts remain a significant challenge—occurring in 10% to 20% of merges, especially when overlapping modifications affect the same region of a file. Previous research has focused on analyzing, preventing, predicting, or automatically resolving such conflicts. However, these approaches often disregard modifications made outside the conflict regions, which can lead to an underestimation of the actual complexity involved in merge conflict resolution, as already noted in the specialized literature. This work aims to address this gap by investigating the role of non-conflicting changes in the merge conflict resolution process. To this end, we analyzed 80 repositories covering eight programming languages—C, C#, C++, Java, JavaScript, PHP, Python, and Ruby—in order to better understand the relevance of non-conflicting changes during conflict resolution. In the analyzed data, we found that, on average, 28.05% of conflicting merges in the studied projects include changes outside the conflict regions during the resolution process.

Palavras-chave: Git, version control systems, software merging, merge conflicts, changes.

Agradecimentos

Aos meus pais pelo sustento e o apoio.

Ao professor Gleiph e ao co-orientador Heleno pela orientação e principalmente, pela paciência, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

"Sim, existem dois caminhos que você pode seguir, mas a longo prazo
Ainda há tempo para mudar a estrada em que você está".
Robert Plant / James Patrick "Jimmy" Page (Stairway To Heaven)

Conteúdo

Lista de Figuras **8**

Lista de Tabelas **9**

Lista de Abreviações **10**

1 Introdução **11**
1.1 Apresentação do Tema . . . 11
1.2 Contextualização . . . 11
1.3 Descrição do Problema . . . 12
1.4 Justificativa/Motivação . . . 13
1.5 Trabalhos Relacionados . . . 14
1.6 Objetivos . . . 16
1.7 Organização do Documento . . . 17

2 Fundamentação Teórica **18**
2.1 Introdução . . . 18
2.2 Sistemas de Controle de Versão . . . 18
2.3 Conflitos de *Merge* . . . 19
2.4 *Chunks* . . . 20
2.5 Tipos de arquivo em conflito . . . 22
2.6 Decisão de desenvolvedor . . . 23
2.7 Considerações Finais . . . 24

3 Algoritmo *Merge Nature* **26**
3.1 Introdução . . . 26
3.2 Camada de Projeto . . . 27
3.3 Camada de *Merge* . . . 28
3.4 Camada de Arquivo em Conflito . . . 29
3.5 Camada de *Chunk* . . . 29
3.6 Camada de Alterações Externas . . . 29
3.7 Considerações Finais . . . 32

4 Estudo Prático **33**
4.1 Introdução . . . 33
4.2 Questões de Pesquisa . . . 33
4.3 Seleção do Projetos . . . 35
4.4 Resultados . . . 36
4.4.1 RQ1: Com que frequência os conflitos de *merge* são resolvidos com pelo menos uma alteração externa? . . . 36
4.4.2 RQ2: Quantas alterações externas, medidas em número de linhas, são feitas durante o processo de resolução? . . . 40
4.4.3 RQ3: Como as alterações externas estão distribuídas entre arquivos com e sem conflito durante a resolução de conflitos de *merge*? . . . 41

4.4.4 RQ4: Com que frequência uma alteração externa corresponde ao conteúdo de um *chunk*? 45
4.5 Discussão 47
4.6 Ameaças à Validade 48
4.7 Considerações Finais 51

5 Conclusão **52**

Bibliografia **53**

Lista de Figuras

2.1 Diagrama de histórico de *commits*. 20

4.1 Distribuição das proporções de EAO. 37
4.2 Distribuição do número de alterações externas por *merge* conflitante. . . . 41
4.3 Distribuição do número de alterações externas por *merge* conflitante. . . . 42
4.4 *Boxplot* de OAE em arquivos com e sem conflito. 43
4.5 Taxas de locais de alterações externas. 44
4.6 *Boxplot* de RCC. 46

Lista de Tabelas

3.1 Trecho de *diff* com *chunk*. 30

4.1 Estatísticas resumidas de *merges*, arquivos em conflito e *chunks* por linguagem. 36
4.2 Trecho de *diff* com *marcadores de conflito*. 38
4.3 Os 5 projetos com maiores OAE. 39
4.4 Os 5 projetos com menores OAE. 39
4.5 Os 5 projeto com menores OAI. 39

Lista de Abreviações

DCC	Departamento de Ciência da Computação
UFJF	Universidade Federal de Juiz de Fora
SCV	Sistema de Controle de Versão
LLM	*Large Language Models*
OAE	Ocorrência de Alterações Externas
OAI	Ocorrência de Alterações Internas
RCC	Razão de Correspondência com *Chunk*

1 Introdução

1.1 Apresentação do Tema

Devido ao crescimento acelerado da indústria de *software* e à complexidade crescente dos projetos, as empresas têm adotado equipes de desenvolvimento cada vez maiores e mais diversificadas. Esse aumento no número de desenvolvedores trabalhando simultaneamente em um mesmo projeto intensifica a necessidade de ferramentas que garantam a organização, integridade e rastreamento das modificações no código. Nesse contexto, Sistemas de Controle de Versão (SCV), como *Git*[1] e Subversion[2], tornaram-se indispensáveis. Esses sistemas permitem que várias pessoas trabalhem em paralelo, realizando mudanças no código coordenadamente e garantindo que todas as alterações possam ser revertidas ou auditadas, se necessário. Além disso, SCVs possibilitam gerenciar diferentes versões de *software*, viabilizando a colaboração em larga escala.

1.2 Contextualização

Uma das características que fazem os SCVs essenciais para o desenvolvimento de *software* é a funcionalidade de ramos. Também chamadas de ramificações, ela é uma funcionalidade essencial em SCVs, como *Git*, que permitem criar linhas de desenvolvimento paralelas do código. Cada ramo é uma linha independente de trabalho, na qual os desenvolvedores podem implementar novas funcionalidades, corrigir *bugs* ou experimentar ideias sem afetar o código principal, geralmente mantido no ramo *main* (MENS, 2002). Outra vantagem dos ramos é a possibilidade de realizar mudanças isoladas, mantendo o desenvolvimento organizado e evitando interferências no código de outros ramos. Quando as modificações em um ramo estão prontas, elas podem ser integradas de volta ao projeto principal por meio de um processo chamado *merge* (MENS, 2002). Isso proporciona um fluxo de trabalho no qual múltiplas tarefas podem ser executadas simultaneamente por diferentes membros

[1] ⟨https://git-scm.com/⟩
[2] ⟨https://subversion.apache.org/⟩

da equipe.

Quando chega o momento de integrar as mudanças de volta ao ramo principal, podem surgir conflitos de *merge*. Esses conflitos ocorrem quando as alterações em dois ou mais ramos são realizadas em linhas adjacentes, resultando em modificações incompatíveis que o SCV não consegue conciliar automaticamente (MENS, 2002). Por exemplo, se dois desenvolvedores editarem a mesma linha de código de maneira diferente, o SCV sinaliza um conflito que precisa ser resolvido manualmente. Esse processo exige que os desenvolvedores revisem as alterações feitas e tomem decisões sobre qual versão manter ou como combinar as diferentes contribuições.

Os SCVs criam regiões para demarcar os conflitos de *merge* conhecidas como *chunks*. Esses *chunks* são blocos de texto que o sistema insere automaticamente no arquivo quando ele detecta alterações conflitantes feitas em paralelo em um mesmo trecho de código. Esses blocos marcam claramente as regiões onde o conflito ocorreu, dividindo o código em diferentes partes.

Os *chunks* são delimitados por marcas que servem como um alerta visual para os desenvolvedores, indicando que eles precisam intervir manualmente para resolver o conflito. A resolução de um *chunk* envolve revisar cuidadosamente as diferentes versões do código e decidir qual delas deve ser mantida ou, em alguns casos, combinar as mudanças para preservar o funcionamento correto do *software*. Embora os *chunks* sejam uma ferramenta útil para sinalizar problemas, eles também podem interromper o fluxo de trabalho, exigindo tempo e esforço para serem resolvidos (VALE et al., 2021).

1.3 Descrição do Problema

Abordagens existentes para auxiliar na resolução de conflitos de merge, como o *AutoMerge*(ZHU; HE, 2018) e o *Almost Rerere*(GONZALEZ; FRATERNALI, 2022), normalmente assumem que os desenvolvedores irão modificar apenas as regiões de código explicitamente marcadas como conflitantes durante o processo de *merge*. A premissa subjacente é que a tarefa do desenvolvedor se limita a resolver apenas essas áreas destacadas, sem alterar partes não relacionadas do arquivo. No entanto, estudos recentes e evidências empíricas (VALE et al., 2021; BOLL et al., 2024) sugerem que, na prática, os desenvol-

vedores fazem alterações fora dos *chunks* durante o processo de resolução dos conflitos. Neste trabalho, essas alterações são chamadas de "alterações externas".

Essas alterações externas podem ocorrer por diversas razões. Por exemplo, ao tentar resolver um conflito, o desenvolvedor percebe a necessidade de ajustes adicionais em outras partes do código para garantir que o sistema continue funcionando corretamente após a fusão das mudanças (VALE et al., 2021; BRINDESCU et al., 2020a). Isso pode envolver a refatoração de código, correção de erros não relacionados diretamente ao conflito, ou mesmo a melhoria da consistência e legibilidade do código. Essa prática desafia as suposições tradicionais dos algoritmos automáticos de resolução de *merge*, que não consideram a possibilidade de alterações externas e, portanto, podem subestimar a complexidade do processo real de resolução.

1.4 Justificativa/Motivação

O conhecimento sobre alterações externas no processo de resolução de conflitos de *merge* ainda é escasso. Na literatura, Vale et al. (2021) demonstraram algum interesse nesse fenômeno, embora não tenha sido o foco principal de sua pesquisa. A contribuição mais significativa de Vale et al. (2021) para o tema veio por meio de uma enquete aplicada a desenvolvedores, na qual foram incluídas duas perguntas diretamente relacionadas às alterações externas: "Eu olho para mudanças não conflitantes para resolver conflitos?" e "Eu mudo o código não conflitante para resolver conflitos de *merge* e evitar a introdução de comportamento inesperado para o projeto?". Os resultados dessa pesquisa fornecem uma visão inicial sobre o papel dessas alterações no processo de resolução de conflitos.

Segundo os dados coletados, 50,7% dos desenvolvedores afirmaram que frequentemente ou quase sempre examinam o código não conflitante durante a resolução de conflitos. Além disso, 25,7% dos participantes indicaram que frequentemente ou quase sempre alteram o código não conflitante. Esses resultados são indicativos que as alterações externas não são meros eventos isolados ou irrelevantes no processo de *merge*. Pelo contrário, elas parecem ser uma prática comum entre os desenvolvedores, podendo desempenhar um papel significativo na garantia de que a fusão ocorra sem comprometer a funcionalidade do *software*.

Se houver fatores decisivos que levam à realização de alterações externas durante a resolução de conflitos de *merge*, sua identificação será fundamental para o aprimoramento de estratégias automatizadas de resolução mais elaboradas como a *ChatMerge* (SHEN et al., 2023b) que utiliza aprendizado de máquina juntamente com uma LLM (*Large Language Model*), que neste caso é o *ChatGPT*[3]. Essas ferramentas de resolução automatizada são extremamente úteis, especialmente considerando que entre 10% e 20% dos *merges* realizados em projetos de *software* acabam gerando conflitos (BRINDESCU et al., 2020a; BRUN et al., 2011; KASI; SARMA, 2013).

1.5 Trabalhos Relacionados

O *merge* de software é uma atividade fundamental no desenvolvimento colaborativo, mas os conflitos de *merge* associados a esse processo representam desafios significativos. Consequentemente, uma grande quantidade de pesquisas tem se concentrado em entender, prevenir e resolver esses conflitos.

Diversos estudos empíricos buscaram caracterizar a natureza e a frequência dos conflitos de *merge*. Pesquisas iniciais, geralmente baseadas na mineração de repositórios públicos como os disponíveis no *GitHub*, estabeleceram que conflitos são eventos comuns, afetando aproximadamente 10% a 20% dos *merges* (CUNHA; ACCIOLY; BORBA, 2022; AMARAL et al., 2020; RIBEIRO; COSTA; SANTOS, 2022). Estudos como os de Menezes et al. (2020) e Dias, Borba e Barreto (2020) investigaram fatores relacionados à ocorrência de conflitos, identificando atributos como número de contribuidores, número de *commits*, número de arquivos alterados e a duração do *branch* como influentes. Menezes et al. (2020), em particular, constatou que os atributos do *branch* sendo integrado geralmente têm mais influência do que os do *branch* de destino. Já Ribeiro, Costa e Santos (2022) confirmaram, por meio de entrevistas e questionários, que os desenvolvedores percebem a duração dos *branches* e a falta de comunicação como os principais fatores que levam a conflitos. Outros trabalhos aprofundaram-se nos tipos específicos de conflito, indo além da simples sobreposição textual para incluir conflitos de *build* e de testes, os quais frequentemente exigem análises mais complexas para serem detectados e resolvi-

[3] ⟨https://chatgpt.com⟩

dos (SHEN et al., 2023a). O próprio processo de resolução também tem sido estudado, examinando estratégias comuns utilizadas por desenvolvedores (GHIOTTO et al., 2020; YUZUKI; HATA; MATSUMOTO, 2015), as dificuldades enfrentadas durante o processo de entendimento (BRINDESCU et al., 2020b), e se a resolução de conflitos é particularmente propensa a erros (BRINDESCU et al., 2020a). Curiosamente, Amaral et al. (2020) constataram que *merges* conflitantes não têm uma probabilidade significativamente maior de introduzir *bugs* do que *commits* regulares. A literatura também explorou estratégias específicas de *merge*, como o *rebase* (JI et al., 2020), e a diferença entre conflitos ocorridos localmente e aqueles visíveis nos repositórios remotos (CUNHA; ACCIOLY; BORBA, 2022), revelando que as taxas de conflito locais podem ser substancialmente mais altas.

Embora os estudos mencionados forneçam *insights* valiosos sobre as causas dos conflitos e padrões gerais de resolução, menos atenção tem sido dada às modificações específicas realizadas pelos desenvolvedores fora das regiões marcadas como conflitantes durante o processo de resolução — o que denominamos de alterações externas. Estudos mais recentes, no entanto, começaram a abordar esse aspecto de maneira implícita ou explícita.

Vale et al. (2021) analisaram quantitativamente 66 projetos no *GitHub* para identificar fatores que afetam o tempo de resolução dos conflitos. Seu modelo de regressão revelou que dependências entre o código conflitante e alterações externas (arquivos não conflitantes no cenário do *merge*) prolongam significativamente o esforço de resolução. Em uma pesquisa com 140 participantes, os desenvolvedores relataram gastar tempo considerável compreendendo como o código conflitante interage com essas modificações externas, sugerindo que dependências contextuais — e não apenas o tamanho do conflito — determinam a complexidade da resolução. Embora o trabalho evidencie o impacto do código externo a partir da percepção dos desenvolvedores, ele não formaliza métodos para aproveitar esse contexto durante a resolução, nem apresenta evidências empíricas das modificações externas nos projetos analisados.

Boll et al. (2024) propuseram uma resolução semi-automatizada de conflitos com base em padrões independentes de linguagem. Sua análise de cerca de 131.000 *merges* do *GitHub* mostrou que, embora 87.9% dos trechos individuais possam ser resolvidos

usando padrões simples aplicados apenas dentro dos limites do conflito (derivabilidade), a resolução do *merge* completo frequentemente falha porque os desenvolvedores também modificam o contexto externo (código fora dos *chunks*) ou porque alguns trechos (os 12.1% restantes) exigem edições personalizadas não cobertas pelos padrões simples. Consequentemente, apenas 34.5% dos *merges* foram totalmente deriváveis. A abordagem proposta com base em padrões não aborda essas edições personalizadas nem as modificações no contexto externo. Essa lacuna evidencia a necessidade de estratégias que incorporem alterações externas, especialmente quando padrões simples não são suficientes. Os autores reconhecem que dependências contextuais reduzem a eficácia de métodos de resolução puramente textuais.

1.6 Objetivos

O objetivo geral da pesquisa é explorar as ocorrências das alterações externas no processo de resolução de conflitos de *merge*, analisando frequência que as alterações externas acontecem e quantidade de linhas, de modo a avaliar sua relevância. Um primeiro objetivo específico seria determinar com que frequência essas modificações ocorrem durante o desenvolvimento colaborativo. Entender a incidência dessas alterações é essencial para avaliar a relevância do fenômeno e seu impacto no fluxo de trabalho dos desenvolvedores. Além disso, a pesquisa também planeja investigar em quais locais essas alterações são feitas: se elas ocorrem predominantemente em arquivos que apresentam conflitos ou se se estendem a arquivos que, a princípio, não estão diretamente envolvidos no conflito.

Outro objetivo específico é identificar com que frequência as alterações externas possuem conteúdo idêntico ao conteúdo interno a algum *chunk* do conflito.

Como temos as seguintes questões as quais serão aprofundadas na Seção 4.2:

- **RQ1**: Com que frequência os conflitos de *merge* são resolvidos com pelo menos uma alteração externa?
- **RQ2**: Quantas alterações externas, medidas em número de linhas, são feitas durante o processo de resolução?

- **RQ3**: Como as alterações externas são distribuídas entre arquivos com e sem conflito durante a resolução de conflitos de *merge*?

- **RQ4**: Com que frequência uma alteração externa corresponde ao conteúdo de um *chunk*?

1.7 Organização do Documento

Este trabalho está organizado em quatro capítulos. No Capítulo 2 são apresentados os conceitos teóricos essenciais encontrados na literatura, que são fundamentais para a compreensão do estudo. O Capítulo 3 descreve em detalhe o algoritmo utilizado para a extração dos dados. O Capítulo 4 expõe o estudo prático, bem como suas principais implicações. Finalmente, no Capítulo 5, são apresentadas as conclusões e propostas de trabalhos futuros.

2 Fundamentação Teórica

2.1 Introdução

Neste capítulo, são apresentados os conceitos fundamentais para a compreensão deste trabalho. Na Seção 2.2 é feita uma análise aprofundada das funcionalidades dos Sistemas de Controle de Versão (SCVs), com foco no papel essencial que esses sistemas desempenham no gerenciamento de projetos colaborativos, além de uma explicação detalhada sobre o funcionamento dos ramos e sua importância para o desenvolvimento paralelo. Na Seção 2.3, são abordados os conflitos de *merge*, explorando como esses conflitos surgem a partir do uso de ramos.

Na Seção 2.4, o foco será nos *chunks*, que são as estruturas que demarcam as áreas de conflito dentro dos arquivos, indicando em quais linhas ocorreram as divergências entre as versões do código. Além disso, é explicado como os *chunks* facilitam a visualização e resolução dos conflitos de *merge*. Em seguida, na Seção 2.5, serão explicados os Tipos de Arquivos em Conflito, uma classificação feita pelo nossso algoritmo proposto. Por fim, as Decisões de Desenvolvedor, um conceito introduzido por Ghiotto et al. (2018) que indicam como um *chunk* foi solucionado, serão aprofundadas na Seção 2.6.

2.2 Sistemas de Controle de Versão

Sistemas de Controle de Versão (SCVs), como o *Git* e o Subversion, são ferramentas essenciais no desenvolvimento de *software* que permitem o gerenciamento e a rastreabilidade das alterações feitas em arquivos ao longo do tempo (PERRY; SIY; VOTTA, 2001). Esses sistemas possibilitam que os desenvolvedores acompanhem o histórico de modificações, revertam a versões anteriores do código e colaborem em projetos compartilhados. Ao registrar cada alteração e fornecer uma visão detalhada do progresso do projeto, os SCVs são indispensáveis para a Gerência de Configuração.

Na Gerência de Configuração, os SCVs desempenham um papel crucial ao as-

segurar que todos os artefatos de *software* (por exemplo, código-fonte e documentação) sejam gerenciados de forma organizada e controlada. A gerência de configuração envolve a identificação, controle e documentação de todos os itens de configuração do projeto, como código-fonte, documentação e arquivos de *build* (CONRADI; WESTFECHTEL, 1998). Os sistemas de controle de versão ajudam a implementar essas práticas, fornecendo um repositório onde as mudanças são registradas, facilitando o gerenciamento de versões e a coordenação entre diferentes equipes e fases do desenvolvimento.

Em sistemas de controle de versão, ramos são uma funcionalidade que permite criar linhas independentes de desenvolvimento a partir do código principal (PERRY; SIY; VOTTA, 2001). Cada ramo representa uma linha de desenvolvimento do projeto no qual mudanças podem ser realizadas de forma isolada. Essa abordagem é útil para implementar novas funcionalidades, corrigir *bugs* ou experimentar alterações sem comprometer a estabilidade do ramo principal, geralmente chamado de *main*. Após as mudanças serem concluídas e testadas no ramo, elas podem ser integradas de volta ao código principal por meio de um processo de *merge*, permitindo que múltiplas linhas de desenvolvimento sejam unidas.

2.3 Conflitos de *Merge*

Conflitos de *merge* surgem quando diferentes ramos de um projeto de *software*, desenvolvidos de forma independente, são combinados e encontram alterações conflitantes (MENS, 2002). Em um fluxo de trabalho de desenvolvimento que utiliza ramos, cada ramificação permite que desenvolvedores trabalhem em funcionalidades paralela e isoladamente. No entanto, quando essas ramificações são posteriormente integradas, o SCV deve integrar as mudanças feitas em cada ramo. Se duas ramificações alterarem o mesmo trecho de código, o sistema pode não conseguir decidir automaticamente qual versão deve prevalecer, resultando em um conflito de *merge*.

A Figura 2.1 ilustra um exemplo simples que cria dois ramos e realiza a *merge* de seus conteúdos. Cada círculo na figura representa um *commit* o qual representa uma versão do software contendo os estados dos arquivos em determinado momento do desenvolvimento. O *commit* C_1 serve como base para os ramos *feature* e *main*, sendo

denominado **merge base**. Em contraste, um **merge commit**, denotado como C_6, integra contribuições de dois ou mais ramos. Os *commits* que são integrados, C_4 e C_5, são referidos como os pais de C_6. O pai do *merge commit* que recebe as contribuições é convencionalmente chamado de P_1, enquanto o outro é denominado P_2. Neste caso, as contribuições estão vindo do ramo *feature* para o ramo *main*, o que resulta em C_5 sendo o P_1 e C_4 sendo o P_2.

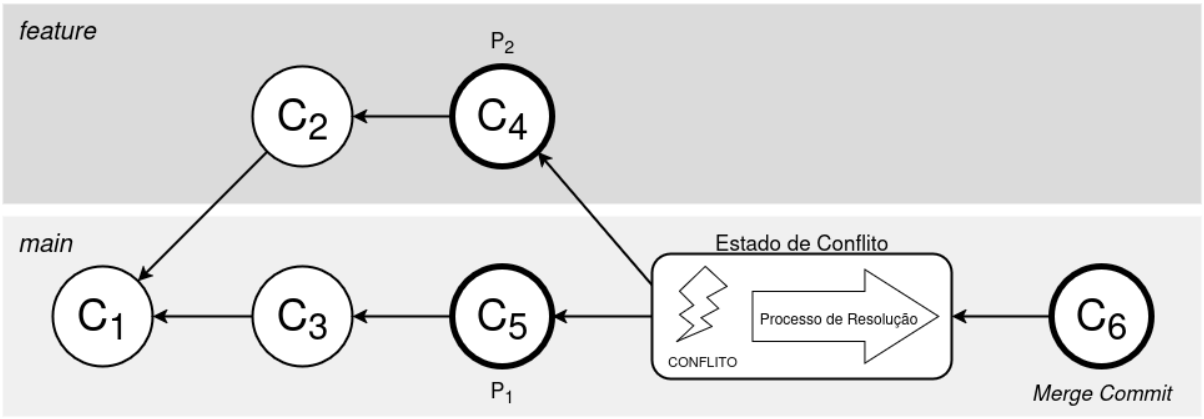

Figura 2.1: Diagrama de histórico de *commits* ilustrando um *merge* com conflito entre as *branches* '*main*' e '*feature*', destacando os pais do *merge*, o *merge base* e a resolução do conflito.

Durante a execução de um *merge*, conflitos podem ocorrer. Neste trabalho, examinamos os conflitos físicos (ou textuais) que surgem quando desenvolvedores modificam simultaneamente a mesma região do código (MENS, 2002). Se um arquivo apresenta esse tipo de conflito em seu conteúdo, ele é chamado de **arquivo em conflito**. Quando essa situação ocorre, o *merge* entra em um **estado de conflito**, o que exige um **processo de resolução**. Esse *processo de resolução* é um esforço manual realizado pelos desenvolvedores para tratar e resolver o trecho conflitante, que será referido daqui em diante como *chunk*.

2.4 *Chunks*

Chunks são estruturas que surgem como resultado de conflitos de *merge*, sendo utilizados para indicar as áreas específicas do código onde houve desacordo entre as alterações feitas em diferentes ramos. Durante o processo de *merge*, quando o sistema detecta que as

mudanças feitas em paralelo não podem ser integradas automaticamente, ele insere marcas no código para indicar as regiões conflitantes. Esses *chunks* incluem as versões conflitantes do código e servem como um guia para os desenvolvedores, que precisam revisar essas regiões e decidir como resolver as discrepâncias.

A Listagem 2.1 apresenta um exemplo de um *chunk* extraído do *merge* `b23de45` do projeto *open-source realm-java*[4], no arquivo `Realm.java`. Esse *chunk* contém três marcadores de conflito padrão: o marcador de **begin** (`<<<<<<<`) na linha 424, o **separator** (`=======`) na linha 431 e o marcador de **end** (`>>>>>>>`) na linha 436. Esses marcadores são essenciais para delimitar as regiões em conflito e para distinguir entre as duas versões divergentes do código. As linhas que aparecem antes do marcador de *begin* (linhas 421–423) e após o marcador de *end* (linhas 437–439) são chamadas de **context**, representando código sem conflito. A linha imediatamente anterior ao marcador de *begin* (linha 423) é chamada de **prefix**, enquanto a linha imediatamente posterior ao marcador de *end* (linha 437) é chamada de **suffix**. O código localizado entre o marcador de *begin* e o *separator* é referido como V_1 (linhas 425–430), enquanto o código entre o *separator* e o marcador de *end* é chamado de V_2 (linhas 432–435).

Listagem 2.1: Exemplo de um chunk extraído do *merge* b23de45 do projeto open-source realm-java no arquivo Realm.java

```
421                 throw e;
422             } finally {
423                 if (!syncAvailable) {
424                     if (commitNeeded) {
425                         realm.commitTransaction();
426                     } else {
427                         realm.cancelTransaction();
428                     }
429                 }
430             }
431         }
```

[4]⟨https://github.com/realm/realm-java⟩

2.5 Tipos de arquivo em conflito

Os **tipos de arquivo em conflito** são classificações baseadas em como o conflito aconteceu. Eles são derivados do que o código-fonte do *Git* denomina *reasons*[5] (razões), embora essa terminologia não seja amplamente documentada. Essas razões são exibidas nas mensagens de conflito apresentadas após a execução de um *merge*. De forma geral, um conflito ocorre quando um mesmo artefato de código é alterado paralelamente de forma diferente. Assim, os tipos de arquivo em conflito indicam quais são os artefatos alterados, quais foram as alterações e se possui *chunks*. A ferramenta implementada para este trabalho é capaz de analisar 11 tipos de arquivo em conflito:

1. **Conteúdo**: Ocorre quando apenas as linhas de um arquivo são alteradas paralelamente de forma diferente. Esse tipo sempre gera *chunks* no arquivo.

2. **Conteúdo com Renomeamento Unilateral**: Ocorre da mesma forma que o tipo Conteúdo, porém o nome do arquivo é alterado somente em um dos pais, resultando em um arquivo com *chunks*, mas com o nome alterado.

3. **Adição de Coincidência**: Ocorre quando o arquivo em questão não existe no Merge Base, ou seja, quando um arquivo é adicionado com o mesmo nome paralelamente, mas com diferenças em seu conteúdo. Esse tipo sempre gera *chunks* no arquivo.

4. **Renomeamento de Arquivo**: Ocorre quando o nome de um arquivo é alterado paralelamente de forma diferente. Esse tipo pode gerar *chunks* no arquivo caso ocorram alterações divergentes em seu conteúdo.

5. **Renomeamento de Diretório**: Ocorre quando o nome do diretório de um arquivo é alterado paralelamente de forma diferente. Esse tipo não gera *chunks*.

6. **Modificar e Deletar**: Ocorre quando um arquivo é deletado em um ramo e modificado em seu conteúdo em outro. Esse tipo nunca gera *chunks*.

[5] O código do *Git* prevê uma grande quantidade de *reasons*, porém, a ferramenta *Merge Nature* não é capaz de identificar todas. Para saber mais sobre as *reasons*, veja o algoritmo de *merge* no código fonte do *Git*: ⟨https://github.com/git/git/blob/master/merge-ort.c⟩.

7. **Renomear e Deletar**: Ocorre quando um arquivo é deletado em um ramo e renomeado em outro. Esse tipo nunca gera *chunks*.

8. **Renomear e Adicionar**: Ocorre quando um arquivo é renomeado em um ramo para um nome de um arquivo que foi adicionado no outro ramo. Esse tipo nunca gera *chunks*.

9. **Local de Arquivo**: Ocorre quando um arquivo é adicionado ou renomeado em um ramo e o diretório desse arquivo é renomeado no outro ramo. Quando isso acontece, o *Git* automaticamente coloca o arquivo em questão no diretório renomeado, não sendo necessário realizar alterações para resolver o conflito. Esse tipo nunca gera *chunks*.

10. **Submódulo**: Os submódulos[6] são estruturas que permitem que você mantenha um repositório *Git* como um subdiretório de outro repositório *Git*. Esse tipo ocorre quando um mesmo submódulo é adicionado ou atualizado para *commits* diferentes nos dois ramos. Esse tipo não gera *chunks*.

11. **Divisão por Renomeação de Diretório**: Ocorre quando um arquivo é adicionado ou renomeado em um ramo e o diretório desse arquivo é severamente alterado no outro ramo, em um nível em que o *Git* não consiga identificar a alteração como uma renomeação de diretório. Esse tipo nunca gera *chunks*.

2.6 Decisão de desenvolvedor

A **decisão de desenvolvedor** é uma classificação de *chunks* introduzida por Ghiotto et al. (2018) que indica como o *chunk* foi solucionado. A solução de um *chunk* é obtida procurando as linhas de contexto no *Merge Commit*. Quando encontradas, o código entre elas é a solução do *chunk* adotada pelo desenvolvedor. Cabe ressaltar que essa é uma análise *post-hoc*. Ou seja, ela é feita a partir do histórico do repositório. Em seguida, os códigos do *chunk* e de sua solução são comparados removendo seus caracteres em branco

[6] ⟨https://git-scm.com/book/en/v2/Git-Tools-Submodules⟩

para determinar a decisão de desenvolvedor. A ferramenta *Merge Nature*(CIRIBELLI et al., 2022) classifica as soluções dos *chunks* em 9 decisões de desenvolvedor:

1. **Versão 1**: O desenvolvedor mantém apenas o código de V1.
2. **Versão 2**: O desenvolvedor mantém apenas o código de V2.
3. **Concatenação**: O desenvolvedor concatena V1 e V2 em qualquer ordem. Quando V1 ou V2 são linhas em branco, a concatenação deles também é a Decisão de Desenvolvedor V1 ou V2. Nesse caso, V1 e V2 têm prioridade na classificação.
4. **Combinação**: O desenvolvedor cria um solução usando linhas de V1 e V2 em qualquer ordem. As Decisões de Desenvolvedor V1, V2 e Concatenação também são uma Combinação, porém no algoritmo *Merge Nature*, elas têm prioridade.
5. **Novo Código**: O desenvolvedor cria um solução que não usa apenas os códigos de V1 e V2, ou seja, escreve um novo código.
6. **Vazio**: O desenvolvedor apaga completamente o *chunk*, ou seja, na solução, não há código entre o prefixo e o sufixo.
7. **Impreciso**: Ocorre quando o desenvolvedor altera as linhas de contexto durante a resolução de um *merge*. Neste caso, o algoritmo da ferramenta *Merge Nature* não consegue definir com precisão a solução do *chunk*.
8. **Arquivo Deletado**: O desenvolvedor apaga o arquivo com o *chunk*.
9. **Postergado**: O desenvolvedor mantém pelo menos um dos marcadores de conflito do *chunk* (*Begin*, *Separator* e *End*). Isso pode indicar que o desenvolvedor postergou a solução do *chunk*.

2.7 Considerações Finais

Este capítulo apresentou os principais conceitos necessários para o entendimento deste trabalho. Inicialmente, foi discutido o papel dos Sistemas de Controle de Versão (SCVs)

e suas funcionalidades, destacando a importância dos ramos para o desenvolvimento colaborativo e o versionamento do código. Em seguida, foram descritos os conflitos de *merge*. Também foram explorados os *chunks*, que marcam as áreas de conflito nos arquivos e fornecendo aos desenvolvedores uma visualização clara das linhas afetadas.

Depois, foram introduzidos e explicados os tipos de arquivo em conflito, os quais indicam como o conflito aconteceu. Por fim, foi explicada a classificação introduzida por Ghiotto et al. (2018) e expandida por Ciribelli et al. (2022) de decisões de desenvolvedor, que indica como o desenvolvedor resolveu um *chunk*.

3 Algoritmo *Merge Nature*

3.1 Introdução

Para analisar os 80 projetos selecionados, adaptamos a ferramenta Merge Nature (CIRIBELLI et al., 2022) originalmente projetada para identificar todos os *merge commits* que resultam em conflitos, juntamente com os arquivos conflitantes e os respectivos *chunks* de conflito. O funcionamento da ferramenta é descrito no Algoritmo 1. Inicialmente, todos os *merge commits* do repositório são recuperados. Para cada *commit*, a versão pai P_1 é extraída, e o *merge* com P_2 é reexecutado. Caso o *merge* resulte em conflito, o *Git* identifica os arquivos afetados. A ferramenta então processa cada um desses arquivos para extrair os *chunks* de conflito. Para cada *chunk*, a ferramenta localiza sua resolução rastreando o prefixo e o sufixo no *merge commit* (que será descrito na Seção 3.6), de modo a determinar sua resolução.

Nesse capítulo, será explicado o funcionamento do algoritmo *Merge Nature* cuja principal funcionalidade é extrair múltiplas informações a respeito de conflitos de *merge* de um repositório *Git*. Caso o repositório do projeto esteja local, o algoritmo recebe apenas o caminho para o projeto. Caso contrário, o algoritmo deve receber o *link* do repositório no *GitHub* e um diretório local para o projeto ser clonado nesse diretório.

Os dados gerados pelo algoritmo podem ser organizados conforme sua granularidade. No nível mais alto, encontram-se os dados que identificam o repositório, processados pelo algoritmo na camada de projeto, detalhada na Seção 3.2. Em seguida, em uma granularidade mais refinada, estão os dados referentes aos *merges* do repositório, obtidos na camada de *merge* explicada na Seção 3.3.

A granularidade seguinte abrange os dados sobre os arquivos em conflito presentes em *merges* conflitantes, extraídos na camada de arquivos em conflito, abordada na Seção 3.4. Por fim, no nível mais detalhado, estão os dados associados aos *chunks* dentro de arquivos em conflito, extraídos na camada de *chunks*, descrita na Seção 3.5.

As alterações externas são informações associadas ao *merge*, mas só podem ser

Algoritmo 1: Extrator de *chunks* de conflito a partir de *merge commits*

Entrada: Repositório Git.
Saída: Todos os *merge commits* que resultam em conflitos, com seus arquivos conflitantes e *chunks* de conflito.

```
1  início
2      Recuperar todos os merge commits do repositório;
3      para cada merge commit C faça
4          Identificar as versões pai P1 e P2 de C;
5          Fazer checkout de P1;
6          Executar merge de P2 em P1;
7          se o merge resultar em conflito então
8              Identificar os arquivos conflitantes a partir da saída do comando
               de merge;
9              para cada arquivo conflitante f faça
10                 Analisar f para extrair os chunks de conflito;
11                 para cada chunk c faça
12                     Rastrear o prefixo e o sufixo de c no merge commit;
13                     Extrair a resolução de c com base no prefixo e sufixo;
14                 fim para cada
15             fim para cada
16         fim se
17     fim para cada
18 fim
```

identificadas após a extração de todos os *chunks*. Isso ocorre porque uma alteração só pode ser classificada como externa se não estiver presente em nenhum *chunk* do *merge* em conflito. Assim, na última camada do algoritmo, denominada camada de alterações externas, detalhada na Seção 3.6, essas alterações são extraídas.

3.2 Camada de Projeto

A camada de projeto é responsável por obter os dados que identificam o repositório *Git*. Dentre esses dados, estão o nome do repositório, a organização que o criou no *GitHub* e a *url* na qual ele foi obtido. Todos esses dados podem ser obtidos através da *url* caso ela seja fornecida como parâmetro. Caso contrário, a *url* pode, em repositórios *Git* oriundos do *GitHub*, ser obtida no arquivo "*.git/config*".

Outra responsabilidade da camada de projeto é obter a lista de *Merge Commits* na história do repositório. O algoritmo obtêm essa lista através do comando *Git* exibido na Listagem 3.1. Cada *commit* dessa lista será processado para que possam ser obtidos

os dados de cada *merge* do repositório.

Listagem 3.1: Comando usado para mostrar todos os commits de um repositório que possuem pelo menos dois pais, ou seja, que são merges.

```
# git log:              Exibe o historico de commits do
#                       repositorio.
# --all:                Mostra o historico de commits de todas
#                       os branches, e nao apenas do branch atual.
# --min-parents=2:      Filtra os commits exibidos para mostrar
#                       apenas aqueles que tem pelo menos dois
#                       pais. Ou seja, apenas os Merge Commits.
$ git log --all --min-parents=2
```

3.3 Camada de *Merge*

A camada de *merge* é responsável por obter os dados dos *commits* envolvidos no *merge* (*Merge Commit*, os Pais e o *Merge Base*) e por reexecutar o *merge* para identificar se ele é um *merge* com conflito. Ela recebe um *Merge Commit* da lista de *commits* obtida na camada de projeto.

Em seguida, é iniciado o processo de *merge*. Para isso, o algoritmo executa um *checkout*[7] do P_1 e o *merge* com o P_2 através do comando na Listagem 3.2. Quando o *merge* não é conflitante, o algoritmo processa o próximo *Merge Commit* da lista na camada de *merge*. Quando o *merge* é conflitante, o termo "CONFLICT" irá aparecer como resposta ao comando, como mostrado na Listagem 3.2. Ele aparece a cada arquivo com conflito, ou seja, a quantidade de aparições desse termo na mensagem do *merge* é igual à quantidade de arquivos em conflito.

Listagem 3.2: Exemplo de um conflito de *merge* no *Git* ao tentar executar o *merge* na *branch* "*whitespace*". Retirado de ⟨https://git-scm.com/book/pt-br/v2/Git-Tools-Advanced-Merging⟩.

```
$ git merge whitespace
Auto-merging hello.rb
CONFLICT (content): Merge conflict in hello.rb
Automatic merge failed; fix conflicts and then commit the result.
```

[7] No *Git*, *checkout* é o comando usado para alternar entre versões, restaurar arquivos específicos ou até mesmo criar novas *branches*. Ele pode ter diferentes comportamentos dependendo do contexto em que é usado, outros detalhes podem ser explorados na documentação do Git em ⟨https://git-scm.com/docs/git-checkout⟩.

3.4 Camada de Arquivo em Conflito

A camada de arquivo em conflito é responsável por obter o conteúdo do arquivo em conflito e por descobrir o nome do arquivo no P_1 e no P_2 através da mensagem após o termo "CONFLICT" que aparece na Listagem 3.2. A classificação do arquivo em tipo de arquivo em conflito é principalmente baseada nessa mensagem que mostra a razão do conflito conforme explicado na Seção 2.5. No caso da Listagem 3.2, a razão do conflito aparece como "content", classificando o arquivo como tipo de arquivo em conflito Conteúdo. Caso o arquivo em conflito seja de um tipo que não gera *chunks*, o algoritmo processa a próxima mensagem que possui o termo "CONFLICT", obtida na camada de *merge*.

3.5 Camada de *Chunk*

A camada de *chunk* é responsável por encontrar e processar os *chunks* de um arquivo em conflito, iterando sobre todas as linhas do arquivo. Quando um *chunk* é encontrado, seu conteúdo é separado nas regiões divididas pelos marcadores.

Outra responsabilidade dessa camada é encontrar a solução do *chunk* e consequentemente a decisão de desenvolvedor. A solução do *chunk* é o código entre o prefixo e o sufixo no *merge commit*. Para encontrar a solução, assume-se que o desenvolvedor não altera o prefixo e o sufixo durante o processo de resolução, caso contrário, não é possível determinar com precisão a solução do *chunk*, resultando na decisão de desenvolvedor Impreciso, introduzida na Seção 2.6.

3.6 Camada de Alterações Externas

Para resolver conflitos, o desenvolvedor pode modificar o arquivo durante o estado de conflito. Chamamos de **alterações** as mudanças realizadas durante o processo de resolução. A Tabela 3.1 apresenta um *diff* do *chunk* para o arquivo `Realm.java` do projeto *realm-java*, comparando a versão do *merge commit* `b23de45` com seu respectivo estado de conflito. O comando `diff` exibe as diferenças entre duas versões. Cada linha da Tabela 3.1 corresponde a uma linha do arquivo `Realm.java`, com a coluna "EC" indicando

EC	MC	Tipo	Conteúdo
...	...	...	...
365	365		`    if (commitChanges) {`
	366	+	`      realm.commitTransaction();`
366		-	`      realm.commitTransaction(false);`
367	367		`    } else {`
368	368		`      realm.cancelTransaction();`
369	369		`    }`
...	...	...	...
421	421		`      commitChanges = false;`
422	422		`      throw e;`
423	423		`    } finally {`
424		-	`<<<<<<< HEAD`
425		-	`      if (!syncAvailable) {`
426		-	`        if (commitNeeded) {`
427		-	`          realm.commitTransaction();`
428		-	`        } else {`
429		-	`          realm.cancelTransaction();`
430		-	`        }`
431		-	`=======`
432	424		`      if (commitChanges) {`
	425	+	`        realm.commitTransaction();`
433		-	`        realm.commitTransaction(false);`
434	426		`      } else {`
435	427		`        realm.cancelTransaction();`
436		-	`>>>>>>> 5b70044d0`
437	428		`      }`
438	429		`    }`
439	430		`  }`
...	...	...	...

Tabela 3.1: Trecho de *diff* entre o *merge commit* `b23de45` do projeto *realm-java* e seu *estado de conflito*. O trecho mostra as alterações realizadas durante o processo de resolução no arquivo `Realm.java`.

a posição da linha no estado de conflito, e a coluna "MC" indicando a posição da linha no *merge commit* final. A coluna "Tipo" indica se a linha foi adicionada ou removida durante o processo de resolução. Se a coluna "Tipo" estiver vazia, a linha não foi nem adicionada nem removida. Linhas adicionadas não possuem valor correspondente na coluna "EC", pois não existiam no estado de conflito; de modo análogo, linhas removidas não têm valor na coluna "MC", pois não aparecem no *merge commit*.

Para identificar essas alterações, nossa abordagem realiza uma comparação entre o *merge commit* e o estado de conflito, conforme ilustrado na Tabela 3.1. Uma alteração pode ser classificada como interna ou externa, dependendo de sua localização em relação aos limites de um *chunk*. Uma alteração é classificada como interna quando ocorre dentro

dos limites do *chunk*. Por exemplo, a remoção da linha 433 no EC, apresentada na Tabela 3.1, é considerada uma alteração interna, assim como a adição da linha 425 no MC correspondente. Por outro lado, uma alteração é considerada externa quando ocorre fora dos limites do *chunk*. Tanto a remoção da linha 366 no EC quanto a adição da linha 366 no MC são exemplos de alteração externa.

O processo de classificação de uma alteração como interna ou externa depende do tipo da modificação, ou seja, se é uma linha removida ou adicionada. No caso de linhas removidas, elas estavam presentes no estado de conflito, mas não aparecem no *merge commit*. Assim, uma linha é considerada uma alteração externa ao *chunk* se estiver fora do intervalo entre o prefixo e o sufixo. Qualquer modificação dentro desse intervalo é classificada como uma alteração interna ao *chunk*.

É importante observar que, neste trabalho, a remoção dos marcadores de *chunk* não é considerada uma alteração interna. Essa decisão foi fundamentada em análises preliminares de 80 projetos, que revelaram que, em média, 99% dos *merge commits* continham pelo menos uma alteração interna. Esse percentual elevado se deve ao fato de que os marcadores de conflito são inseridos automaticamente pelo *Git* para sinalizar divergências entre versões e não representam código intencionalmente adicionado pelo desenvolvedor. Portanto, sua remoção é uma etapa natural no processo de resolução, já que o código resultante poderia ser sintaticamente inválido caso esses marcadores fossem mantidos. O 1% restante corresponde a casos em que os desenvolvedores deixaram, acidentalmente, os marcadores de conflito no código final mesclado.

Por outro lado, linhas adicionadas não existem no estado de conflito, apenas no *merge commit*. Para classificá-las, é necessário rastrear o prefixo e o sufixo no *merge commit*. Esse rastreamento é realizado utilizando o comando `diff`, como ilustrado na Tabela 3.1. Neste exemplo, a linha 423 no estado de conflito—referida como prefixo—corresponde à linha 423 no *merge commit*. De forma semelhante, a linha 437 no estado de conflito—o sufixo—corresponde à linha 428 no *merge commit*. Esse rastreamento pode falhar se o desenvolvedor tiver removido as linhas de prefixo ou sufixo, impossibilitando a classificação das alterações. Caso contrário, a lógica de classificação segue o mesmo princípio utilizado para as linhas removidas, usando as linhas rastreadas no *merge commit*.

3.7 Considerações Finais

Neste capítulo foram apresentadas as camadas de execução do algoritmo *Merge Nature*, cada uma responsável por obter e processar um tipo de dado com base em sua granularidade. Inicialmente, foram descritos os dados utilizados para identificar o projeto e o método de obtenção dos *commits* que são *merges*. Em seguida, foram abordados os dados relacionados ao *merge* e o processo de execução do *merge*.

Na sequência, foi detalhada a obtenção dos arquivos em conflito e sua classificação. Posteriormente, aprofundou-se na extração dos *chunks* nos arquivos em conflito. Por fim, foram apresentados a definição das alterações externas e o processo de extração dessas modificações.

4 Estudo Prático

4.1 Introdução

Este capítulo descreve o estudo prático conduzido neste estudo, juntamente com seus respectivos resultados. A Seção 4.2 apresenta as questões de pesquisa que nortearam o trabalho. A Seção 4.3 detalha o processo de seleção dos repositórios Git utilizados como amostra. Na Seção 4.4, são descritas as respostas a cada questão de pesquisa, bem como os resultados obtidos. A Seção 4.5 discute os principais achados do estudo. Por fim, a Seção 4.6 analisa as potenciais ameaças à validade interna, externa, de construção e à conclusão estatística da pesquisa.

4.2 Questões de Pesquisa

O principal objetivo deste trabalho é explorar o quão comuns são as alterações externas no processo de resolução de conflitos de *merge*, por meio de uma análise retrospectiva do histórico dos projetos. As seguintes perguntas de pesquisa orientam o estudo:

- **RQ1: Com que frequência os conflitos de *merge* são resolvidos com pelo menos uma alteração externa?**

  Esta pergunta investiga com que frequência os conflitos de *merge* são resolvidos com pelo menos uma modificação feita fora dos limites do *chunk*. As abordagens existentes para resolução de conflitos normalmente concentram seus esforços no próprio *chunk*, muitas vezes ignorando o fato de que os desenvolvedores podem introduzir mudanças em outras partes do código durante o processo de resolução. De fato, Vale et al. (2021) conduziram uma pesquisa com 140 desenvolvedores, que relataram que 25% deles normalmente analisam e modificam regiões não conflitantes. Apesar dessa evidência, nenhum trabalho anterior analisou sistematicamente repositórios para determinar com que frequência esse comportamento ocorre. Se tais ocorrências forem

comuns, os desenvolvedores de ferramentas de resolução de conflitos podem precisar projetar funcionalidades que considerem o código além da fronteira dos *chunks*.

- **RQ2: Quantas alterações externas, medidas em número de linhas, são feitas durante o processo de resolução?**

  Esta pergunta examina o número de linhas adicionadas ou removidas fora dos chunks durante a resolução de um *merge*. Compreender esse impacto ajuda a avaliar a complexidade do processo de resolução fora do código conflitante. Um volume elevado de alterações externas por *merge* sugere que ferramentas automatizadas de resolução de conflitos podem precisar analisar um contexto mais amplo do código, já que podem existir dependências significativas entre código conflitante e não conflitante.

- **RQ3: Como as alterações externas são distribuídas entre arquivos com e sem conflito durante a resolução de conflitos de *merge*?**

  Esta pergunta investiga se as alterações externas tendem a ocorrer com mais frequência em arquivos com conflitos ou em arquivos sem conflitos. Compreender essa distribuição é importante, pois uma maior prevalência de mudanças em arquivos sem conflitos sugeriria que abordagens automatizadas de resolução de conflitos podem precisar analisar códigos além dos arquivos diretamente envolvidos no conflito, destacando a presença de possíveis dependências entre arquivos durante o processo de resolução.

- **RQ4: Com que frequência uma alteração externa corresponde ao conteúdo de um *chunk*?**

  Esta pergunta investiga a frequência com que alterações externas correspondem a trechos de código que originalmente apareciam dentro dos *chunks*, sugerindo possíveis movimentos ou cópias de código durante a resolução do conflito. Se essas correspondências forem frequentes, isso pode indicar que ferramentas automatizadas de resolução de conflitos poderiam se beneficiar da análise do conteúdo dos *chunks* para inferir alterações externas, reduzindo a necessidade de considerar muitas regiões não conflituosas.

4.3 Seleção do Projetos

Os projetos de código aberto analisados neste estudo foram obtidos a partir do trabalho de Menezes et al. (2020), que selecionou 80 projetos do GitHub em 8 linguagens de programação diferentes, ou seja, 10 projetos para cada linguagem (C, C#, C++, Java, JavaScript, PHP, Python e Ruby). Esse estudo foi selecionado por possuir uma quantidade de projetos que poderiam ser analisados em tempo hábil para este Trabalho de Conclusão de Curso e por apresentar uma diversidade de linguagens de programação.

Durante o processo de extração dos projetos, foi feita uma reavaliação da linguagem principal de cada repositório. A linguagem principal de um projeto foi determinada com base na linguagem com a maior porcentagem de código, conforme informado pelo GitHub[8]. Como resultado dessa reavaliação, o projeto *borg*[9], anteriormente classificado como um projeto em C por Menezes et al. (2020), foi reclassificado como um projeto em Python. Essa reclassificação resultou em um desequilíbrio entre as linguagens, com 11 projetos agora classificados como Python e 9 como C. Além disso, o projeto *Directus*, anteriormente classificado como baseado em PHP, passou por uma migração[10] para TypeScript como sua linguagem principal. Como o TypeScript não está incluído no escopo desta análise, optou-se por selecionar um projeto alternativo em PHP da mesma organização, especificamente o *v8-archive*[11].

A Tabela 4.1 apresenta um resumo estatístico dos principais indicadores relacionados a *merges* com conflito, arquivos em conflito e *chunks* para diferentes linguagens de programação analisadas. Esses dados fornecem uma visão geral do volume de conflitos enfrentados em projetos que utilizam essas linguagens.

Observa-se que a linguagem PHP apresenta os maiores valores médios para todos os três indicadores: *merges* com conflito (379,40), arquivos em conflito (835,40) e *chunks* (2020,50). Além disso, seus valores máximos também são expressivos, especialmente para *chunks* (8410). Já linguagens como C e *Python* apresentam valores médios menores em comparação, com o C tendo, por exemplo, uma média de 100,44 *merges* com conflito e

[8] Esta análise foi realizada em 12/12/2024.
[9] ⟨https://github.com/borgbackup/borg⟩
[10] ⟨https://directus.io/blog/introducing-directus-9⟩
[11] ⟨https://github.com/directus/v8-archive⟩

Tabela 4.1: Estatísticas resumidas de *merges*, arquivos em conflito e *chunks* por linguagem.

Linguagem	Dado	Média	Desvio Padrão	Mediana	Mínimo	Máximo
Java	*Merges* com conflito	252,10	247,69	95,00	22,00	637,00
	Arquivos em conflito	499,40	499,89	228,50	43,00	1352,00
	Chunks	749,60	788,35	322,50	65,00	2124,00
C	*Merges* com conflito	**100,44**	44,52	107,00	36,00	175,00
	Arquivos em conflito	**194,22**	115,24	142,00	51,00	355,00
	Chunks	**296,67**	208,25	173,00	57,00	693,00
C#	*Merges* com conflito	374,30	384,18	228,00	35,00	1183,00
	Arquivos em conflito	941,40	1193,81	395,50	49,00	3633,00
	Chunks	1413,00	1779,64	592,50	66,00	4956,00
JavaScript	*Merges* com conflito	288,20	241,34	210,00	61,00	811,00
	Arquivos em conflito	601,80	643,95	428,00	107,00	2316,00
	Chunks	1009,90	867,65	788,50	207,00	3113,00
PHP	*Merges* com conflito	**379,40**	363,17	203,50	153,00	1258,00
	Arquivos em conflito	**835,40**	939,53	492,00	192,00	3191,00
	Chunks	**2020,50**	2610,15	783,00	222,00	**8410,00**
Python	*Merges* com conflito	182,18	116,58	118,00	68,00	378,00
	Arquivos em conflito	331,55	239,79	171,00	133,00	872,00
	Chunks	652,09	631,38	317,00	178,00	2117,00
Ruby	*Merges* com conflito	199,80	196,20	123,50	44,00	654,00
	Arquivos em conflito	353,30	389,54	229,00	61,00	1337,00
	Chunks	605,30	651,09	377,00	82,00	2203,00

194,22 arquivos em conflito, e o *Python* com médias de 182,18 e 331,55 respectivamente.

4.4 Resultados

4.4.1 RQ1: Com que frequência os conflitos de *merge* são resolvidos com pelo menos uma alteração externa?

Para responder a essa pergunta, foram calculadas, para cada projeto, a proporção de *merges* com conflitos que envolvem ao menos uma alteração externa — referida como uma *ocorrência de alteração externa* (**OAE**, na sigla em inglês) — e a proporção que envolve ao menos uma alteração interna — referida como uma *ocorrência de alteração interna* (**OAI**, na sigla em inglês). A Figura 4.1 mostra um boxplot que ilustra a distribuição dessas proporções para os 80 projetos analisados.

A média de OAE é de 28,05%, com um desvio padrão (σ) de 9,48%. Conforme a Figura 4.1, o valor da mediana de OAE é de 26,44%, mas o primeiro quartil indica que 25% dos projetos possuem pelo menos 20,78% de seus *merges* com conflito resolvidos

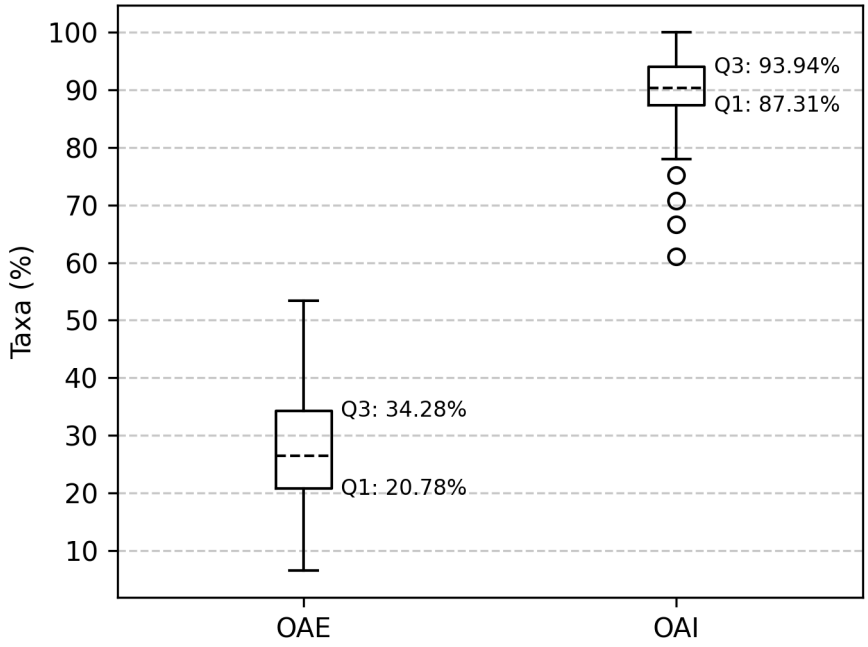

Figura 4.1: Distribuição das proporções de *merges* com conflito que possuem ao menos uma alteração externa por projeto.

com ao menos uma alteração externa. Isso sugere que alterações externas não são raras a ponto de serem consideradas desprezíveis.

Além disso, segundo a Figura 4.1, o valor mediano de OAI é de 90,36%. Um valor elevado de OAI já era esperado, visto que a resolução de conflitos normalmente exige que os desenvolvedores modifiquem os *chunks*. Instâncias sem alterações internas geralmente ocorrem quando a resolução envolve apenas a remoção dos marcadores de conflito ou quando as modificações se limitam a linhas em branco, que são desconsideradas. A Tabela 4.2 mostra um exemplo em que houve somente remoções de marcadores como alteração dentro do trecho que são representadas pelas linhas destacadas com fundo vermelho.

É importante notar que OAE e OAI não são sempre complementares. Por exemplo, se um projeto possui uma OAE de 20%, a OAI não será necessariamente de 80%, já que um único *merge* pode conter tanto alterações externas quanto internas ao mesmo tempo.

Para investigar as características dos projetos com os maiores e menores percentuais de OAE, é apresentada a Tabela 4.3 e a Tabela 4.4.

A Tabela 4.3 apresenta os cinco projetos com os maiores valores de OAE. É

EC	MC	Tipo	Conteúdo
...	...	...	...
83	83		`public function getArgument($name);`
84		-	`<<<<<<< HEAD`
85	84		
...	...	...	...
102	103		`public function hasArgument($name);`
103		-	`=======`
104		-	`>>>>>>> 457f9f6b`
105	104		
...	...	...	...
118	121		`public function getOption($name);`
119		-	`<<<<<<< HEAD`
120	122		
...	...	...	...
137	141		`public function hasOption($name);`
138		-	`=======`
139		-	`>>>>>>> 457f9f6b`
140	142		
...	...	...	...
144	150		`public function isInteractive();`
145		-	`<<<<<<< HEAD`
146	151		
...	...	...	...
151	158		`public function setInteractive($interactive);`
152		-	`=======`
153		-	`>>>>>>> 457f9f6b`
144	150		`}`
...	...	...	...

Tabela 4.2: *Diff* no arquivo Input/InputInterface.php do projeto console durante o processo de resolução do *merge commit* 9df10ae1

Nome do Projeto	Linguagem Principal	Merges em Conflito	OAE (%)	OAI (%)
bootstrap	JavaScript	352	53.41	97.16
vert.x	Java	92	52.17	92.39
druid	Java	98	45.92	92.86
OpenRA	C#	35	45.71	91.43
pinpoint	Java	22	45.45	86.36

Tabela 4.3: Os 5 projetos com maiores OAE.

Nome do Projeto	Linguagem Principal	Merges em Conflito	OAE (%)	OAI (%)
laravel	PHP	153	6.54	95.42
tmux	C	107	11.21	80.37
qemu	C	121	11.57	75.21
v8-archive	PHP	194	13.40	92.27
abp	C#	766	15.67	92.17

Tabela 4.4: Os 5 projetos com menores OAE.

importante notar que quatro desses projetos foram desenvolvidos principalmente em linguagens orientadas a objetos nativas: três em Java e um em C#. Por outro lado, a Tabela 4.4 lista os cinco projetos com os menores valores de OAE. Nesse grupo, não há uma predominância clara de linguagens orientadas a objetos nativas, com apenas um projeto usando C#. Isso pode sugerir que o uso de linguagens orientadas a objetos pode induzir alterações externas, devido, por exemplo, às dependências entre métodos.

A Tabela 4.5 apresenta os cinco projetos com os menores valores de OAI, cujos quatro primeiros são *outliers* apresentados na Figura 4.1. Ao contrário da análise feita para OAE, a análise quantitativa não apresenta padrões ou similaridades evidentes entre esses projetos, indicando a necessidade de uma investigação mais aprofundada que pode será feita em trabalhos futuros.

Nome do Projeto	Linguagem Principal	Merges em Conflito	OAE (%)	OAI (%)
emscripten	C++	149	27.52	61.07
systemd	C	36	19.44	66.67
cucumber-ruby	Ruby	212	38.21	70.75
qemu	C	121	11.57	75.21
dbeaver	Java	68	19.12	77.94

Tabela 4.5: Os 5 projeto com menores OAI.

Resultado 1: A análise revelou que, em média, 28,05% dos *merges* com conflito são resolvidos por meio de pelo menos uma alteração externa, indicando que esse tipo de modificação, embora menos frequente que as alterações internas, não é pode ser negligenciado no processo de resolução de conflitos.

4.4.2 RQ2: Quantas alterações externas, medidas em número de linhas, são feitas durante o processo de resolução?

Para responder a esta questão de pesquisa, foram examinados o número de alterações externas associadas a cada *merge* com conflito, medidas em linhas. É importante destacar que as linhas em branco não são consideradas. Para esta análise, foram utilizados apenas os *merges* que possuem ao menos uma modificação externa, totalizando 5.647 *merges*, ou 26,00% do conjunto de dados. Foi utilizado esse subconjunto porque, como visto na Seção 4.4.1, a média de OAE dos 80 projetos é 28,05%, ou seja, a maior parte dos *merges* em conflito de cada projeto não possui alterações externas. Portanto, se esse dados fossem considerados, a quantidade de alterações externas por *merge* em conflito tenderia a zero.

A Figura 4.2 apresenta um gráfico de barras ilustrando a quantidade de *merges* agrupados pela quantidade de alterações externas neste subconjunto. Os dados revelam que 65,50% desses *merges* foram resolvidos com a quantidade de alterações externas variando de 1 a 10. Do lado oposto, 7,60% dos *merges* são resolvidos com mais de 101 alterações externas.

A Figura 4.3 mostra os mesmos dados usando um *boxplot*. Outliers com valores muito discrepantes da distribuição geral não são mostrados para facilitar a visualização. A média de alterações externas é de 181,89 ($\sigma = 3284,54$). Essa média é fortemente influenciada por um pequeno número de *merges* que envolveram deleções em larga escala, sendo que o maior *merge* contabilizou 163.760 alterações externas. Por outro lado, o valor da mediana é 5, indicando que 50% dos *merges* no conjunto de dados são resolvidos com até 5 alterações externas.

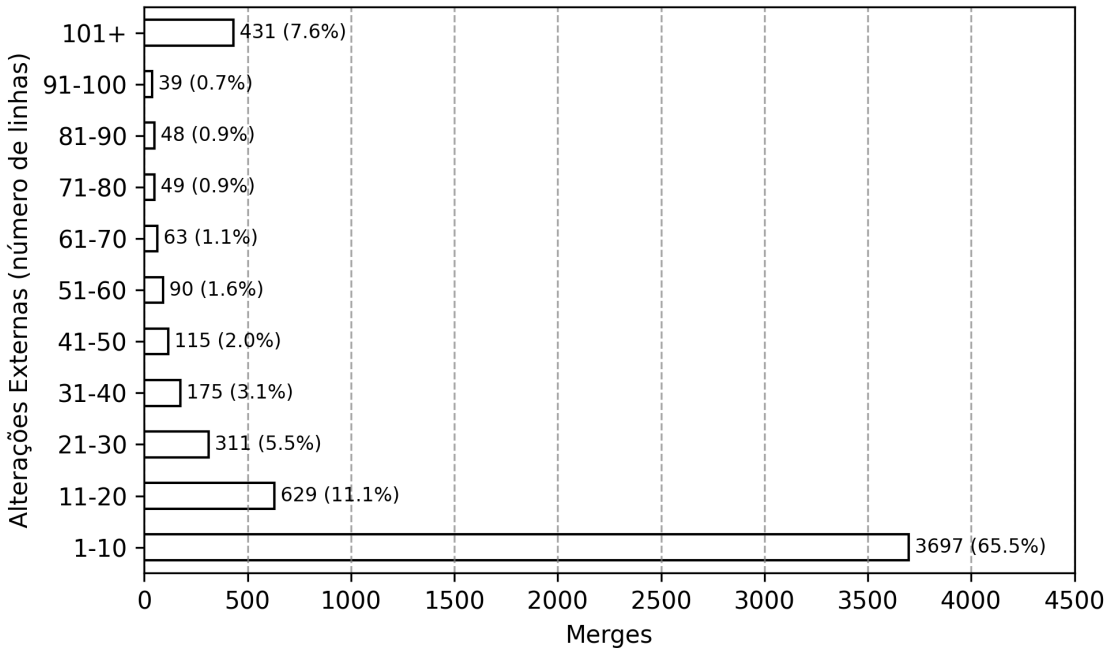

Figura 4.2: Distribuição do número de alterações externas por *merge* conflitante.

Resultado 2: A análise das alterações externas, medidas em número de linhas, revelou que, considerando apenas os *merges* com pelo menos uma modificação externa — correspondendo a 26,00% do total — a mediana de alterações externas por *merge* é de 5 linhas, indicando que metade dos casos envolve um número relativamente baixo de mudanças externas. Contudo, a média de 181,89 alterações, fortemente influenciada por alguns *merges* com alterações em larga escala. Ainda, a distribuição mostra que 65,50% dos *merges* com alterações externas apresentam entre 1 e 10 linhas modificadas, enquanto uma parcela menor, de 7,60%, demanda mais de 101 alterações externas.

4.4.3 RQ3: Como as alterações externas estão distribuídas entre arquivos com e sem conflito durante a resolução de conflitos de *merge*?

Para responder a esta pergunta, foram calculados, para cada projeto, a proporção de *merges* com conflito que envolvem ao menos uma alteração externa em um arquivo com conflito — chamada de **OAE em arquivo com conflito** — e a proporção de *merges* com conflito que envolvem ao menos uma alteração externa em um arquivo sem conflito

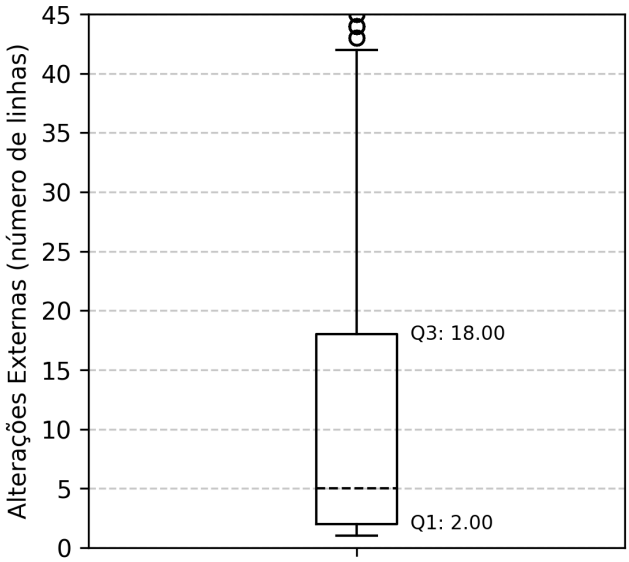

Figura 4.3: Distribuição do número de alterações externas por *merge* conflitante.

— **chamada de OAE em arquivo sem conflito**. A Figura 4.4 mostra dois *boxplots* ilustrando a distribuição dessas proporções nos 80 projetos.

A Figura 4.4 mostra que o número de alterações externas em arquivos com conflito é maior do que em arquivos sem conflito. A média de OAE em arquivos com conflito é 20,79% ($\sigma = 7,89$), enquanto em arquivos sem conflito é 11,90% ($\sigma = 7,20$). De forma semelhante, a mediana de OAE é maior para arquivos com conflito (19,50%) em comparação com arquivos sem conflito (10,16%).

Para avaliar formalmente a diferença e determinar o teste estatístico apropriado, foi verificada a normalidade das distribuições dos dados utilizando o teste de Shapiro-Wilk. O teste indicou que a distribuição da OAE em arquivos sem conflito desvia-se significativamente da normalidade ($p < 0,001$), enquanto a distribuição da OAE em arquivos com conflito não apresentou evidência forte contra a normalidade no nível convencional ($\alpha = 0,05$) ($p = 0,073$). Dada a não normalidade em um dos grupos, utilizamos o teste não paramétrico de Mann-Whitney U para comparar as distribuições.

O teste de Mann-Whitney U confirmou uma diferença estatisticamente significativa entre as proporções de OAE em arquivos com conflito e arquivos sem conflito ($U = 5277,0$, $p < 0,001$). Esse resultado indica fortemente que a diferença observada nas

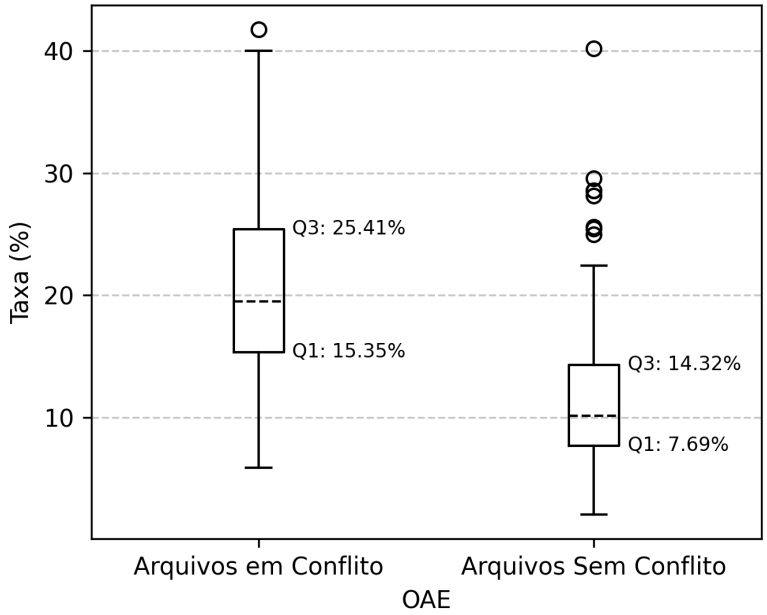

Figura 4.4: Distribuição de OAE em arquivos de conflito e em arquivos sem conflito de todos os projetos.

distribuições é dificilmente devido ao acaso.

Além disso, para avaliar a magnitude dessa diferença, foi calculado o *Common Language Effect Size* (CLES) (MCGRAW; WONG, 1992). O CLES resultante é 0,82. Esse valor pode ser interpretado da seguinte forma: se selecionado aleatoriamente um valor de OAE do grupo de arquivos com conflito e outro do grupo de arquivos sem conflito para qualquer comparação entre projetos, há uma probabilidade de 82% de que a proporção do primeiro grupo seja maior. De acordo com a convenção (VARGHA; DELANEY, 2000), isso indica um efeito de grande magnitude.

Portanto, com base tanto na significância estatística quanto no tamanho de efeito elevado, conclui-se que as alterações externas durante a resolução de conflitos de *merge* ocorrem significativamente e substancialmente com mais frequência em arquivos que contêm conflitos, em comparação com aqueles que não contêm, entre os projetos analisados.

A Figura 4.5 categoriza os *merges* pelo número de alterações externas, seguindo a mesma abordagem da Figura 4.2. Para cada grupo, mostra a taxa de *merges* resolvidos exclusivamente por alterações externas em arquivos com conflito, exclusivamente em arquivos sem conflito e em ambos.

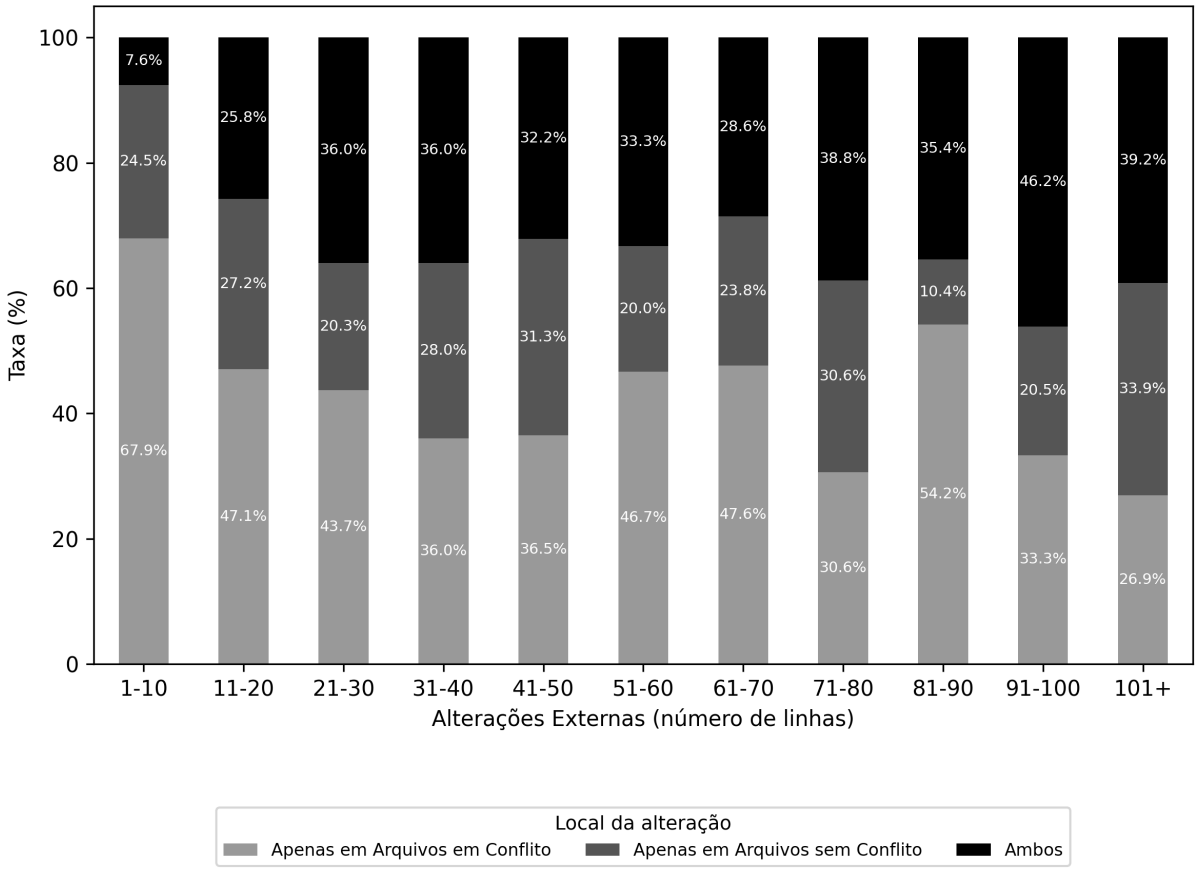

Figura 4.5: Taxas de locais de alterações externas em *merges* conflitantes.

Para *merges* envolvendo de 1 a 10 alterações externas, há uma predominância clara de resoluções baseadas exclusivamente em alterações externas em arquivos com conflito. À medida que o número de alterações externas aumenta, essa taxa tende a diminuir, enquanto a proporção de *merges* envolvendo alterações externas em ambos os tipos de arquivo parece aumentar. Em contraste, a taxa de *merges* resolvidos exclusivamente por mudanças em arquivos sem conflito permanece relativamente estável em diferentes níveis de alterações externas.

Resultado 3: A análise da distribuição das alterações externas entre arquivos com e sem conflito mostrou que, nos projetos analisados, essas modificações ocorrem com mais frequência em arquivos que contêm conflitos. Em média, 20,79% dos *merges* com conflito apresentam ao menos uma alteração externa em arquivos com conflito, enquanto esse percentual é de 11,90% para arquivos sem conflito. Além disso, a análise por faixas de quantidade de alterações externas revelou que, para *merges* com até 10 alterações, a resolução tende a ocorrer majoritariamente por mudanças em arquivos com conflito; à medida que o número de alterações aumenta, cresce também a proporção de *merges* com alterações em ambos os tipos de arquivo, enquanto a taxa de resoluções exclusivamente em arquivos sem conflito permanece relativamente constante.

4.4.4 RQ4: Com que frequência uma alteração externa corresponde ao conteúdo de um *chunk*?

Uma alteração externa é considerada correspondente a um *chunk* quando o conteúdo da alteração aparece dentro dele. Para verificar isso, todas as linhas de V_1 são concatenadas e os espaços são removidos. Em seguida, as linhas em branco são removidas do conteúdo da alteração. Se o conteúdo resultante da alteração estiver contido no V_1 concatenado, ele é considerado uma correspondência. O mesmo é feito para o conteúdo de V_2. Se a alteração corresponder a V_1 ou V_2, considera-se que o *chunk* é a origem da alteração, partindo do pressuposto de que essa alteração externa resulta de um reposicionamento de linha cujo ponto de partida é o próprio *chunk*.

Não foram consideradas alterações que consistem apenas em linhas em branco ou apenas em símbolos. No caso da alteração externa ser um conteúdo em branco, isso provavelmente se trata de uma formatação de código e, no caso das linhas apenas com símbolos, provavelmente se trata de abertura e fechamento de blocos de código. Apenas alterações que contenham ao menos um número ou letra são incluídas pois é o que é considerado, nesse trabalho, mínimo para aquela linha ser um código, diferenciando-a de elementos puramente estruturais ou estéticos, como espaços em branco ou símbolos isolados.

Para responder à pergunta, foram analisados os *merges* que possuem ao menos uma alteração externa. Foi calculada a razão entre o número de alterações externas que correspondem ao conteúdo de um *chunk* e o número total de alterações externas naquele *merge*. Essa métrica foi chamada de *razão de correspondência com chunk* (**RCC**). A distribuição do RCC entre todos os *merges* no conjunto de dados filtrado está representada no boxplot da Figura 4.6.

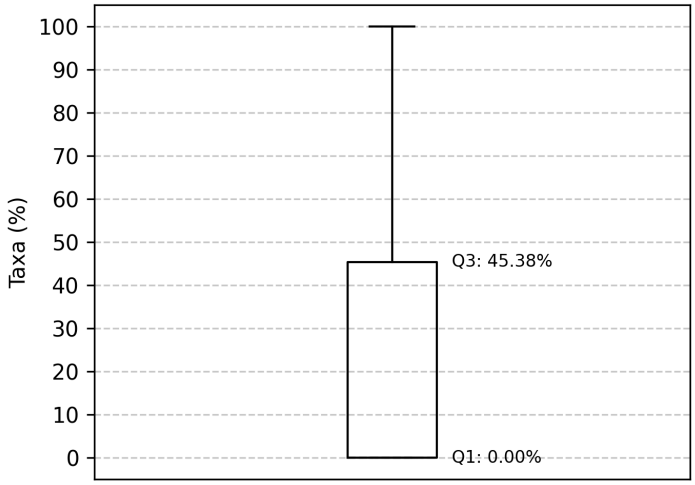

Figura 4.6: Razões de alterações externas que correspondem ao conteúdo de um *chunk* em *merges* com conflitos.

A média do RCC é 23,90% ($\sigma = 35,43\%$), mas a mediana é 0%. Isso indica que, em 50% dos nossos *merges*, não houve nenhuma alteração externa que correspondesse ao conteúdo de um *chunk*. Além disso, o terceiro quartil é 45,38%, ou seja, apenas 25% dos *merges* possuem uma taxa maior 45,38% de alterações externas que correspondem ao conteúdo de algum *chunk*.

Resultado 4: A análise revelou que, embora a média da razão de correspondência entre alterações externas e o conteúdo de *chunks* (RCC) seja de 23,90%, a mediana é 0%, indicando que, em pelo menos metade dos *merges* com alterações externas, nenhuma dessas alterações corresponde diretamente ao conteúdo de um *chunk*. Apenas 25% dos *merges* apresentam RCC superior a 45,38%.

A grande diferença entre a média e a mediana ocorre porque, como visto na Seção 4.4.2, a maioria dos *merges* com conflito possui apenas uma alteração externa. Assim, se

essa única alteração corresponder a um trecho, o RCC será de 100%.

4.5 Discussão

Os resultados revelaram que uma média de 28,05% dos conflitos de *merge* de um projeto são resolvidos com pelo menos uma alteração externa, ou seja, uma alteração realizada fora das regiões diretamente conflitantes. No entanto, na maioria desses casos, o número de alterações externas varia de 1 a 10, sugerindo que, embora tais eventos sejam relativamente frequentes, eles não exigem que o desenvolvedor altere muitas linhas de código. Esse comportamento indica que ferramentas de apoio à resolução de conflitos de *merge* não devem se restringir apenas à análise das regiões conflitantes, mas também considerar o contexto. Ainda assim, o número geralmente pequeno de alterações externas por *merge* sugere que esse código contextual tende a não ser amplamente modificado, ou seja, mesmo que seja provável que ocorram alterações externas a cada processo de resolução, o número de alterações externas não é alto.

Também observa-se que, entre os cinco projetos com a maior proporção de *merges* conflitantes resolvidos com pelo menos uma alteração externa, quatro são implementados em linguagens orientadas a objeto nativas: três em Java e um em C#. Essa constatação pode indicar a necessidade de desenvolver estratégias específicas por linguagem de programação para assistência na resolução de conflitos de *merge*.

Além disso, foi verificado que alterações externas são mais frequentes em arquivos que contêm conflitos do que naqueles sem conflitos. No entanto, a ocorrência de alterações externas em arquivos sem conflito não é muito diferente das que ocorrem em arquivos com conflito. Isso sugere um esforço adicional por parte dos desenvolvedores, que precisam inspecionar e modificar arquivos não diretamente envolvidos no conflito. Portanto, abordagens de resolução de conflitos (ZHU; HE, 2018; GONZALEZ; FRATERNALI, 2022) não devem limitar seu escopo apenas ao arquivo conflitante, mas também considerar o projeto como um todo ou, ao menos, outros arquivos relacionados ao conflito.

Ademais, os dados indicam que a probabilidade de alterações externas em arquivos sem conflito aumenta com o número total de alterações externas. Esse padrão condiz com a hipótese de que modificações nas linhas dentro dos arquivos conflitantes

podem causar efeitos colaterais em outras partes do código, exigindo ajustes adicionais em arquivos que inicialmente não estavam afetados pelo conflito. Ou seja, quanto maior a quantidade de alterações no arquivo em conflito, maior a probabilidade de ocorrência de alterações em arquivos sem conflito.

Por fim, constata-se que é raro um *merge* ser resolvido com uma alteração externa cujo conteúdo seja idêntico ao de qualquer *chunk*. Essa constatação pode indicar que simples movimentações de linha são incomuns durante a resolução de conflitos, ou que a resolução de dependências entre códigos conflitantes e não conflitantes não pode ser realizada apenas com base nas informações contidas nas regiões conflitantes.

4.6 Ameaças à Validade

Esta seção discute possíveis ameaças à validade dos nossos achados, categorizadas de acordo com os tipos de validade estabelecidos na literatura.

Validade de conclusão diz respeito à capacidade de tirar conclusões corretas com base nos dados e na análise estatística. Uma ameaça principal está na confiabilidade das medições obtidas com nossa ferramenta de análise aprimorada. Imprecisões na reprodução dos *merges*, na análise dos marcadores de conflito, no rastreamento de linhas via `diff` (especialmente as linhas de prefixo e sufixo utilizadas para classificação), ou na contagem de linhas alteradas podem afetar as frequências relatadas de ocorrências de alterações externas (OAE), alterações internas (OAI) e contagens de linhas (como na RQ2). Esse risco foi mitigado utilizando como base uma ferramenta previamente validada, aprimorando-a especificamente para capturar alterações fora das regiões conflitantes. Além disso, a ferramenta foi amplamente testada. Embora isso reduza o risco de erros graves, uma pequena possibilidade de falhas residuais permanece.

Outra ameaça potencial diz respeito às conclusões estatísticas. Na RQ2, a distribuição das contagens de linhas alteradas externamente era altamente assimétrica. Para mitigar isso, foi reportado tanto a média quanto a mediana, destacando esta última (5 linhas) como uma medida mais robusta de tendência central. Na RQ3, na qual foi comparado a distribuição de OAEs em arquivos com e sem conflito, a normalidade foi testada usando o teste de Shapiro-Wilk. Como o grupo de arquivos sem conflito violou os pres-

supostos de normalidade, utilizamos o teste não paramétrico de Mann-Whitney U para garantir inferência apropriada. Adicionalmente, foi calculado o tamanho de efeito "*Common Language Effect Size*" (CLES) (MCGRAW; WONG, 1992; VARGHA; DELANEY, 2000) para quantificar a magnitude do efeito observado, reforçando a interpretação para além da significância estatística ($p < 0.001$).

Por fim, a precisão da própria ferramenta aprimorada representa uma ameaça em potencial. Contudo, ao basear sua implementação em trabalhos previamente validados e concentrar as melhorias unicamente na detecção de alterações não conflitantes, busca-se minimizar o risco de introduzir erros significativos.

Validade interna refere-se à possibilidade de outros fatores, além dos estudados, influenciarem os padrões ou relações observadas. Como o trabalho baseia-se em mineração de repositórios e não visa estabelecer causalidade, as ameaças à validade interna dizem respeito principalmente à interpretação. Ainda assim, fatores de confusão como o tamanho do projeto, a experiência da equipe, estratégias de ramificação (por exemplo, *feature branches* vs. *branches* de longa duração), complexidade do código ou o tipo de tarefa de desenvolvimento podem influenciar a extensão e a natureza das alterações externas. Por exemplo, a variação observada entre grupos de linguagens na RQ1 pode refletir diferenças de práticas entre projetos, e não características intrínsecas das linguagens. Para mitigar isso, foi destacada a natureza observacional do estudo e evitou-se fazer afirmações causais.

Outra ameaça possível decorre da dependência dos algoritmos de *merge* e *diff* do *Git*. Caso essas ferramentas apresentem comportamentos inconsistentes, isso pode afetar a precisão na detecção de alterações. No entanto, assume-se que seu comportamento é suficientemente estável entre as versões relevantes do *Git*, o que mitiga essa ameaça de instrumentação.

Validade de construção refere-se à precisão com que os conceitos teóricos foram operacionalizados e medidos. Uma preocupação chave envolve a definição de "alteração externa", que depende da classificação de linhas fora da faixa de prefixo–sufixo de cada *chunk*. Essa definição se baseia no algoritmo de *diff* do *Git* para rastrear essas linhas entre o estado de conflito e o *commit* de *merge*. Como mencionado na Seção 4.3, se as linhas de prefixo ou sufixo forem modificadas ou removidas, a classificação pode falhar ou

resultar em erros de rotulagem.

Por fim, medir a magnitude das alterações com base na contagem de linhas pode não refletir a complexidade semântica real ou o esforço de desenvolvimento envolvido. Por exemplo, a análise não distingue entre alterações funcionais no código e modificações em comentários ou formatações. Isso foi mitigado definindo claramente o escopo e as limitações da operacionalização baseada em linhas. Similarmente, a definição de "conflito" refere-se estritamente a conflitos textuais (físicos) detectados pelo *Git*, e não considera conflitos sintáticos ou semânticos.

Validade externa trata da generalização dos achados. Busca-se mitigar ameaças à representatividade analisando uma amostra um conjunto de projetos previamente explorados na literatura com 80 projetos de código aberto, cobrindo 8 linguagens de programação (C, C#, C++, Java, JavaScript, PHP, Python e Ruby). Essa seleção multi-linguagem e multi-projeto — originalmente curada por Menezes et al. (2020) e ajustada levemente conforme descrito na Seção 4.3 — oferece maior potencial de generalização dos achados em comparação com estudos baseados em um único sistema, dado que, como visto nos resultados da RQ1, certa linguagens podem apresentar uma maior chance de possuir alterações externas no processo de resolução.

Ainda assim, a amostra pode não representar plenamente softwares proprietários, projetos que utilizam outros sistemas de controle de versão, ou o desenvolvimento em outras linguagens. Pequenos desequilíbrios na quantidade de projetos por linguagem (devido a reclassificações) foram observados, mas são improváveis de enviesar significativamente os resultados dado que o desequilíbrio não ultrapassa mais de um projeto por linguagem. Essas ameaças foram mitigadas descrevendo claramente as características da amostra e delineando os limites prováveis da generalização. Por fim, é reconhecida uma limitação relacionada à validade temporal: práticas e ferramentas evoluem com o tempo, portanto, descobertas baseadas em dados históricos de repositórios podem não refletir integralmente as tendências atuais.

O trabalho se baseia nesses estudos ao investigar diretamente o fenômeno das alterações externas. Enquanto Vale et al. (2021) identificaram a dificuldade percebida causada por dependências externas e Boll et al. (2024) destacaram a necessidade do

contexto externo em certos casos, este estudo fornece evidência empírica direta sobre a frequência e a natureza dessas alterações externas. Ao aprimorar uma ferramenta para capturar especificamente inserções e deleções de linhas em regiões não conflitantes durante o processo de resolução em 80 repositórios, foi quantificada a frequência com que os desenvolvedores modificam código fora dos trechos conflitantes e foi oferecida uma análise inicial das características dessas alterações, complementando, assim, trabalhos anteriores que focaram principalmente nas regiões de conflito ou na percepção dos desenvolvedores quanto a fatores externos.

4.7 Considerações Finais

Este capítulo apresentou o estudo prático realizado neste estudo, detalhando suas etapas, resultados e implicações. Inicialmente, foram descritas as questões de pesquisa que orientaram a investigação. Em seguida, discutiu-se o processo de seleção dos repositórios *Git* que compuseram a amostra.

Com base nos dados coletados, cada questão de pesquisa foi respondida, conforme apresentado na seção de resultados. A discussão aprofundou os principais resultados apontando possíveis interpretações e implicações práticas.

Por fim, foram analisadas as ameaças à validade do estudo, reconhecendo limitações relacionadas ao delineamento experimental, à generalização dos resultados, à definição dos constructos e à robustez estatística das conclusões.

5 Conclusão

Neste trabalho, foi investigada a frequência de alterações externas em projetos de código aberto. Para isso, foi modificada a já validada ferramenta Merge Nature para que fosse capaz de obter as alterações externas. Com ela, analisamos 80 repositórios do *GitHub* abrangendo 8 linguagens de programação diferentes.

Essa análise revelou uma mediana de 28,05% dos *merges* contendo pelo menos uma alteração externa. Além disso, entre todos os *merges* com pelo menos uma alteração externa e entre 1 a 10 dessas alterações, contata-se que 65,5% dos *merges* se enquadram nessa categoria. Observa-se também que alterações externas são mais frequentes em arquivos com conflito: a proporção mediana de *merges* com pelo menos uma alteração externa afetando arquivos conflitantes foi de 19,50%, em comparação com 10,16% para arquivos sem conflito. Por fim, foi verificado que a porcentagem mediana de alterações externas por *merge* (com pelo menos uma alteração externa) é de 0%, o que sugere que a maioria dos *merges* é dominada por alterações internas, mesmo quando há alterações externas presentes.

Dados esses resultados, é plausível argumentar que as alterações externas também deveriam ser consideradas parte da solução do conflito de *merge*. Isso porque, ao ignorar tais alterações, corre-se o risco de comprometer a validade da avaliação: alterações externas podem influenciar diretamente o resultado final do *merge*, tornando imprecisa qualquer comparação baseada exclusivamente na região do *chunk* como referência. Assim, alterações externas podem representar uma ameaça concreta à validade das avaliações de abordagens que tratam apenas o *chunks* como solução sem considerar o contexto completo das modificações envolvidas.

O escopo deste trabalho não envolve a análise dos motivos para a ocorrência de alterações externas ao *chunk*. Desta forma, como trabalho futuro, pretende-se investigar os construtos de linguagem mais comuns envolvidos nas alterações externas. Além disso, é necessário fazer uma análise qualitativa das alterações externas para verificar se a alterações externas estão estão relacionadas com os *chunks*.

Bibliografia

AMARAL, L.; OLIVEIRA, M. C.; LUZ, W.; FORTES, J.; BONIFÁCIO, R.; ALENCAR, D.; MONTEIRO, E.; PINTO, G.; LO, D. How (not) to find bugs: The interplay between merge conflicts, co-changes, and bugs. In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.]: IEEE, 2020. p. 441–452.

BOLL, A.; DOK, Y. V.; OHRNDORF, M.; SCHULTHEISS, A.; KEHRER, T. Towards semi-automated merge conflict resolution: Is it easier than we expected? In: *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. [S.l.: s.n.], 2024. p. 282–292.

BRINDESCU, C.; AHMED, I.; JENSEN, C.; SARMA, A. An empirical investigation into merge conflicts and their effect on software quality. *Empirical Software Engineering*, Springer, v. 25, p. 562–590, 2020.

BRINDESCU, C.; RAMIREZ, Y.; SARMA, A.; JENSEN, C. Lifting the curtain on merge conflict resolution: A sensemaking perspective. In: *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. [S.l.]: IEEE, 2020. p. 534–545.

BRUN, Y.; HOLMES, R.; ERNST, M. D.; NOTKIN, D. Proactive detection of collaboration conflicts. In: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE '11)*. [S.l.]: ACM, 2011. p. 168–178.

CIRIBELLI, L. R.; LIMA, J. P.; JUNIOR, H. d. S. C.; BARROS, M. de O.; HOEK, A. van der; MURTA, L. G. P.; GHIOTTO, G. Merge nature: a tool to support research about merge conflicts. In: SBC. *Workshop de Visualização, Evolução e Manutenção de Software (VEM)*. [S.l.], 2022. p. 6–10.

CONRADI, R.; WESTFECHTEL, B. Version models for software configuration management. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 30, n. 2, p. 232–282, 1998.

CUNHA, M.; ACCIOLY, P.; BORBA, P. The private life of merge conflicts. In: *XXXVI Brazilian Symposium on Software Engineering (SBES 2022)*. [S.l.]: ACM, 2022. p. 1–10.

DIAS, K.; BORBA, P.; BARRETO, M. Understanding predictive factors for merge conflicts. *Information and Software Technology*, v. 121, p. 106256, 2020.

GHIOTTO, G.; MURTA, L.; BARROS, M.; HOEK, A. V. D. On the nature of merge conflicts: a study of 2,731 open source java projects hosted by github. *IEEE Transactions on Software Engineering*, IEEE, v. 46, n. 8, p. 892–915, 2018.

GHIOTTO, G.; MURTA, L.; BARROS, M.; HOEK, A. van der. On the nature of merge conflicts: A study of 2,731 open source java projects hosted by GitHub. *IEEE Transactions on Software Engineering*, v. 46, n. 8, p. 892–915, 2020.

GONZALEZ, S. L. H.; FRATERNALI, P. Almost rerere: Learning to resolve conflicts in distributed projects. *IEEE Transactions on Software Engineering*, IEEE, v. 49, n. 4, p. 2255–2271, 2022.

JI, T.; CHEN, L.; YI, X.; MAO, X. Understanding merge conflicts and resolutions in git rebases. In: *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. [S.l.]: IEEE, 2020. p. 70–80.

KASI, B. K.; SARMA, A. Cassandra: proactive conflict minimization through optimized task scheduling. In: NOTKIN, D.; CHENG, B. H. C.; POHL, K. (Ed.). *Proceedings of the 35th ICSE*. [S.l.]: IEEE / ACM, 2013. p. 732–741. ISBN 978-1-4673-3076-3.

MCGRAW, K. O.; WONG, S. P. A common language effect size statistic. *Psychological bulletin*, American Psychological Association, v. 111, n. 2, p. 361, 1992.

MENEZES, J. W.; TRINDADE, B.; PIMENTEL, J. F.; MOURA, T.; PLASTINO, A.; MURTA, L.; COSTA, C. What causes merge conflicts? In: *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*. [S.l.: s.n.], 2020. p. 203–212.

MENS, T. A state-of-the-art survey on software merging. *IEEE transactions on software engineering*, IEEE, v. 28, n. 5, p. 449–462, 2002.

PERRY, D. E.; SIY, H. P.; VOTTA, L. G. Parallel changes in large-scale software development: an observational case study. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM New York, NY, USA, v. 10, n. 3, p. 308–337, 2001.

RIBEIRO, B. B.; COSTA, C.; SANTOS, R. P. dos. Understanding and analyzing factors that affect merge conflicts from the perspective of brazilian software developers. *Journal of Software Engineering Research and Development*, v. 10, p. 12, 2022.

SHEN, B.; GULZAR, M. A.; HE, F.; MENG, N. A characterization study of merge conflicts in java projects. *ACM Transactions on Software Engineering and Methodology*, v. 32, n. 2, 2023.

SHEN, C.; YANG, W.; PAN, M.; ZHOU, Y. Git merge conflict resolution leveraging strategy classification and llm. In: IEEE. *2023 IEEE 23rd International Conference on Software Quality, Reliability, and Security (QRS)*. [S.l.], 2023. p. 228–239.

VALE, G.; HUNSEN, C.; FIGUEIREDO, E.; APEL, S. Challenges of resolving merge conflicts: A mining and survey study. *IEEE Transactions on Software Engineering*, IEEE, v. 48, n. 12, p. 4964–4985, 2021.

VARGHA, A.; DELANEY, H. D. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, Sage Publications Sage CA: Los Angeles, CA, v. 25, n. 2, p. 101–132, 2000.

YUZUKI, R.; HATA, H.; MATSUMOTO, K. How we resolve conflict: An empirical study of method-level conflict resolution. In: *Proceedings of the 2015 IEEE 1st International Workshop on Software Analytics (SWAN '15)*. [S.l.]: IEEE, 2015. p. 21–24.

ZHU, F.; HE, F. Conflict resolution for structured merge via version space algebra. *Proceedings of the ACM on Programming Languages*, ACM New York, NY, USA, v. 2, n. OOPSLA, p. 1–25, 2018.