# Refinando a Precisão da Detecção de Conflitos: Uma Análise do CSDiff com Abordagem Focalizada

Felipe Araujo e Paulo Borba
fbma@cin.ufpe.br
phmb@cin.ufpe.br
Centro de Informática
Universidade Federal de Pernambuco
Recife, Pernambuco, Brazil

Guilherme Cavalcanti
guilherme.cavalcanti@belojardim.ifpe.edu.br
Instituto Federal de Pernambuco
Belo Jardim, Pernambuco, Brazil

## RESUMO

Software development is increasingly complex, with developers simultaneously working on different parts of the source code to build, maintain, and enhance systems. However, this collaborative nature of development can lead to conflicts when multiple individuals attempt to simultaneously modify the same file. In this scenario, code merge tools play a crucial role in detecting and resolving these conflicts. One such tool is CSDiff, a conflict detection and resolution tool, an alternative to the traditional and widespread Diff3. CSDiff stands out by using customizable separators specific to each programming language to help in conflict resolution. In this article, we propose an improvement to the functionality of CSDiff focusing on reducing false positive and false negatives conflicts found when using the original tool. Through an analysis based on Python programs, we compare CSDiff with and without the proposed improvement; we assess the impact on reducing errors presented by the original version of the tool. The results indicate that the proposed improvement not only reduces the number of reported false positive conflicts, leading to a higher proportion of scenarios with correctly resolved conflicts, but also results in a decrease in false negatives when compared to the original tool.

## KEYWORDS

Desenvolvimento colaborativo de software, ferramentas de integração, conflitos de merge.

## 1 INTRODUÇÃO

Durante o processo de integração de código (do inglês, *merge*), é possível que as modificações realizadas em paralelo por diferentes desenvolvedores ocorram em um mesmo arquivo, ou até mesmo no mesmo trecho do código. Quando isso ocorre, poderá levar a ocorrência de conflitos de integração. Se a ferramenta de versionamento de código não é capaz de lidar e resolver automaticamente um conflito, cabe aos desenvolvedores analisar as modificações feitas e investir tempo e esforço adicional para resolvê-lo [4], impactando na produtividade geral do time, e, potencialmente, abrindo margem para a criação de novos problemas, caso a solução para o conflito não seja realizada corretamente.

Para tentar lidar com estes conflitos, técnicas de detecção e resolução automática de conflitos foram adicionados às ferramentas de versionamento de código. A mais simples e mais popularmente utilizada é a solução de análise puramente textual, não-estruturada [9]. Esta solução analisa linha a linha do código base e de cada uma das versões alteradas às quais se deseja aplicar o processo de *merge*, em caso de modificações realizadas na mesma linha ou em linhas adjacentes do código, a ferramenta identifica um novo conflito. O problema maior desta abordagem é que alterações em um mesmo trecho do código não necessariamente influenciam no resultado uma da outra. Desta forma, essa abordagem acaba por reportar conflitos em situações que seria suficiente apenas aplicar as duas modificações para resolver o conflito, ou até mesmo aplicar apenas uma das modificações [8]. A essas situações dá-se o nome de falsos conflitos [5, 6, 13]. Para tentar lidar com a geração de falsos conflitos pela abordagem linha a linha, foram desenvolvidas abordagens estruturadas e semi-estruturadas[2, 3, 5, 16]. Essas abordagens, além da localização das modificações, utilizam um conjunto de regras e símbolos da linguagem e analisam também a sintaxe das alterações envolvidas no processo de *merge*. A criação de uma ferramenta estruturada específica para cada linguagem de programação, porém, é muito custosa. Pensando em diminuir esse custo, foi desenvolvida a ferramenta *CSDiff* [7].

*CSDiff* é uma ferramenta de integração não-estruturada que utiliza separadores sintáticos das linguagens (símbolos que separam escopos e contextos) para realizar a análise durante o processo de *merge*. A ideia é usar esses separadores, além das quebras de linha, como divisores de contexto para detecção e resolução de conflitos sobre o *diff3* (uma ferramenta não-estruturada tradicional [9]). O *CSDiff* começa pré-processando os arquivos modificados em paralelo pelos desenvolvedores (*left* e *right*) e a versão base. Nesse pré-processamento, são adicionados novos marcadores para separar as linhas de acordo com os separadores sintáticos. Após isso, o *CSDiff* realiza a análise linha a linha dos arquivos pré-processados por meio do *diff3*. Conflitos detectados são então pós-processados, removendo os marcadores adicionados. No entanto, essas modificações de pré-processamento podem causar problemas de alinhamento durante o *merge*, gerando falsos conflitos e soluções incorretas.

Para reduzir esse problema, este trabalho propõe uma modificação no *CSDiff*, com um novo processo, aqui chamado de *Separators-based Merge (SepMerge)* em contraste com o tradicional *line-based merge (diff3)* trazendo uma abordagem mais focalizada para o *CSDiff*, executando-o apenas nos trechos que já foram detectados como conflitos previamente pelo *diff3*. Posteriormente, nós analisamos o impacto dessa modificação na detecção e resolução de conflitos em programas escritos em Python. Em especial, nós investigamos as seguintes perguntas de pesquisa:

(1) PP1: A abordagem focalizada do *SepMerge* reduz a quantidade de conflitos reportados em comparação ao *diff3* e ao *CSDiff*?

(2) PP2: A abordagem focalizada do *SepMerge* reduz a quantidade de cenários de integração com conflitos reportados em comparação ao *diff3* e ao *CSDiff*?
(3) PP3: A abordagem focalizada do *SepMerge* reduz a quantidade de arquivos com erros de identificação de conflitos em comparação ao *diff3* e ao *CSDiff*?
(4) PP4: A abordagem focalizada do *SepMerge* reduz a quantidade de cenários de integração com erros de identificação de conflitos em comparação ao *diff3* e ao *CSDiff*?

Nossos resultados mostram que, em nossa amostra de programas Python, o *SepMerge* não apenas diminuiu o número de conflitos relatados, mas também eliminou completamente os falsos conflitos. Além disso, o *SepMerge* aumentou o número de conflitos resolvidos corretamente e reduziu os casos de resolução automática incorreta.

# 2 MOTIVAÇÃO

## 2.1 Ferramentas de integração não-estruturadas

Uma das ferramentas mais amplamente adotadas para integração de código é o *diff3* [11]. Utilizado pelo Git, um dos sistemas de controle de versão mais populares, *diff3* é responsável por detectar e resolver conflitos que surgem durante o processo de *merge*. Ele funciona comparando três versões de um mesmo arquivo: a versão original (*base*), e as versões derivadas, modificadas pelos desenvolvedores (versões *left* e *right*), num processo chamado *three-way merge*. O *diff3* então tenta reconciliar essas alterações de forma automática, seguindo uma abordagem puramente textual baseada em linhas. No entanto, quando ocorrem interseções entre as alterações do *left* e *right*, ele os sinaliza para intervenção manual. Estas interseções recebem o nome de conflitos de integração [9]. O *diff3* demarca esses conflitos no código integrado através dos marcadores (``>>>>>>>``,``=======`` e ``<<<<<<<``) como mostrado na Figura 1.

```
def to_string(l: List[str]) -> str:
<<<<<<< /left.py
    if l is null or len(l) == 0:
        return ""
    return "__".join(l)
=======
    if len(l) == 0:
        return self.D
    return "__".join(l)
>>>>>>> /right.py
```

**Figura 1: Exemplo de conflito de integração.**

Pela natureza não-estruturada do *diff3*, ela sinaliza como conflitos modificações feitas por dois desenvolvedores que ocorreram na mesma linha, ou em linhas adjacentes. No entanto, frequentemente, essas modificações, mesmo que espacialmente próximas, não interferem uma na outra no sentido sintático e semântico do código [1, 5, 6]. Nesses casos, ao invés de reportar conflito, seria mais interessante que a ferramenta de integração realizasse a integração com sucesso, ao invés de reportar um falso conflito.

Este problema poderia ser resolvido aplicando-se o uso de ferramentas de natureza semiestruturadas e estruturadas[2, 3, 5, 14, 16]. Ao contrário das ferramenta não-estruturadas, essas levam em consideração a estrutura sintática da linguagem para identificar contextos diferentes através da criação de árvores sintáticas a partir dos arquivos do cenário de integração, e, assim, resolver automaticamente os conflitos de modificações realizadas em contextos diferentes.

## 2.2 Integração não-estruturada com separadores

As ferramentas de natureza semiestruturadas e estruturadas mencionadas na seção anterior apesar da maior capacidade de detectar e resolver conflitos, possuem um custo de implementação adicional elevado [6, 13]. Elas se baseiam em manipulação de árvores sintáticas, que, por sua vez, são dependentes da linguagem de programação em questão e demandam um esforço significativo de implementação por linguagem. Além disso, a manipulação das árvores sintáticas possui um custo computacional maior em relação a ferramentas puramente textuais.

Para reduzir essas desvantagens, Clementino et al. [7] propôs uma nova ferramenta chamada *Custom Separators Diff* (*CSDiff*), cujo funcionamento baseia-se na abordagem puramente textual, mas que também considera a estrutura sintática do código por meio de um conjunto de separadores da linguagem, escolhidos pelo usuário e passados como parâmetro para a ferramenta. A ideia trazida por Clementino et al. [7] é mesclar o uso da abordagem linha a linha usada pelo *diff3* com uma separação de contexto auxiliada pelos separadores sintáticos da linguagem de programação conforme descrito abaixo:

(1) *CSDiff* pré-processa os arquivos *base*, *left* e *right* de um cenário de integração, adicionando marcadores ($$$$$$$) e quebras de linha antes e depois de cada um dos separadores sintáticos.
(2) *CSDiff* executa o *diff3* nas versões, agora marcadas, de *base*, *left* e *right*.
(3) *CSDiff* remove, do resultado do *diff3*, os marcadores e quebras de linhas adicionadas durante o primeiro passo.

As Figuras de 2 a 9 ilustram o processo do *CSDiff* descrito acima usando os separadores sintáticos da linguagem de programação *Python*: *"( ) , :"*. No exemplo, um dos desenvolveres (*left*) adiciona uma condição a um `if` pré-existente da *base*, enquanto que o outro desenvolvedor (*right*) modifica o comando de retorno da função. Inicialmente, *CSDiff* pré-processa os arquivos *base* (Figura 2), *left* (Figura 3) e *right* (Figura 4), adicionando os marcadores para cada um dos separadores sintáticos no código fonte:

```
def to_string(l: List[str]) -> str:
    if len(l) == 0:
        return ""
    return "..".join(l)
```

**Figura 2: Arquivo *base* do cenário de integração.**

Como resultado, são criados três novos arquivos: *base_temp* (Figura 5), *left_temp* (Figura 6) e *right_temp* (Figura 7). As quebras

```
def to_string(l: List[str]) -> str:
    if l is null or len(l) == 0:
        return ""
    return "__".join(l)
```

**Figura 3: Arquivo *left* do cenário de integração.**

```
def to_string(l: List[str]) -> str:
    if len(l) == 0:
        return self.D
    return "__".join(l)
```

**Figura 4: Arquivo *right* do cenário de integração.**

de linha são adicionadas para evitar conflitos na execução do *diff3*. Sem elas, modificações em diferentes contextos (separados por separadores) podem ser interpretadas como conflitos. Isso ocorre pois as modificações não estarão mais na mesma linha, ou em linhas adjacentes, devido à linha com os marcadores e separador inserida entre elas.

```
def to_string
$$$$$$$(
$$$$$$$l
$$$$$$$:
$$$$$$$ List[str]
$$$$$$$)
$$$$$$$ -> str
$$$$$$$:
$$$$$$$
    if len
$$$$$$$(
$$$$$$$l
$$$$$$$)
$$$$$$$ == 0
$$$$$$$:
$$$$$$$
        return ""
    return "..".join
$$$$$$$(
$$$$$$$l
$$$$$$$)
$$$$$$$
```

**Figura 5: Arquivo *base* após a marcação de código por *CSDiff*.**

Depois de adicionar os marcadores, *CSDiff* executa o *diff3* nos três arquivos modificados, gerando assim um novo arquivo integrado temporário (Figura 8). Por fim, o *CSDiff* remove todos os marcadores e quebras de linha adicionados por ele do arquivo integração temporário, gerando o arquivo integrado final (Figura 9).

```
def to_string
$$$$$$$(
$$$$$$$l
$$$$$$$:
$$$$$$$ List[str]
$$$$$$$)
$$$$$$$ -> str
$$$$$$$:
$$$$$$$
    if l is null or len
$$$$$$$(
$$$$$$$l
$$$$$$$)
$$$$$$$ == 0
$$$$$$$:
$$$$$$$
        return ""
    return "__".join
$$$$$$$(
$$$$$$$l
$$$$$$$)
$$$$$$$
```

**Figura 6: Arquivo *left* após a marcação de código por *CSDiff*.**

```
def to_string
$$$$$$$(
$$$$$$$l
$$$$$$$:
$$$$$$$ List[str]
$$$$$$$)
$$$$$$$ -> str
$$$$$$$:
$$$$$$$
    if len
$$$$$$$(
$$$$$$$l
$$$$$$$)
$$$$$$$ == 0
$$$$$$$:
$$$$$$$
        return self.D
    return "__".join
$$$$$$$(
$$$$$$$l
$$$$$$$)
$$$$$$$
```

**Figura 7: Arquivo *right* após a marcação de código por *CSDiff*.**

```
def to_string
$$$$$$$(
$$$$$$$l
$$$$$$$:
$$$$$$$ List[str]
$$$$$$$)
$$$$$$$ -> str
$$$$$$$:
$$$$$$$
    if l is null or len
$$$$$$$(
$$$$$$$l
$$$$$$$)
$$$$$$$ == 0
$$$$$$$:
$$$$$$$
<<<<<<< /left_temp.py
        return ""
    return "__".join
=======
        return self.D
    return "__".join
>>>>>>> /right_temp.py
$$$$$$$(
$$$$$$$l
$$$$$$$)
$$$$$$$
```

**Figura 8: Arquivo integrado temporário por *diff3*.**

```
def to_string(l: List[str]) -> str:
    if l is null or len(l) == 0:
<<<<<<< left.py
        return ""
    return "__".join(l)
=======
        return self.D
    return "__".join(l)
>>>>>>> /right.py
```

**Figura 9: Arquivo integrado final por *CSDiff*.**

Como descrito por Khanna et al. [9], uma das etapas do funcionamento do *diff3* é o processo de pareamento das linhas entre as versões *base* e *left*, e entre as versões *base* e *right*, para identificar quais foram as modificações realizadas por cada uma dessas novas versões do arquivo. A abordagem do *CSDiff* se mostrou eficiente em reduzir ou resolver conflitos com modificações em contextos diferentes [7]. No entanto, para grandes arquivos, a alta quantidade de marcadores adicionados potencialmente aumenta o risco de uma falha nesse pareamento quando o *diff3* é executado para integrar os arquivos com marcadores. Isso aumenta o risco de um falso conflito, ou um resultado diferente do esperado.

Visando mitigar esse risco, trazemos nesse trabalho uma proposta de processo alternativo, diminuindo o escopo de atuação do *CSDiff*, e trazendo uma abordagem mais focalizada, executando a lógica trazida pelo *CSDiff* apenas sobre os conflitos reportados pela abordagem puramente textual, linha a linha, apresentada pelo *diff3*. Com esse novo processo, nomeado de *Separators-based Merge* (*SepMerge*), buscamos melhorar a precisão da detecção e resolução de conflitos do *CSDiff*, ao mesmo passo em que buscamos implementar uma ferramenta que não obtenha resultados piores do que o *diff3*, que atualmente é a ferramenta mais difundida e comumente utilizada na comunidade de desenvolvedores para detecção e resolução de conflitos.

## 3 SOLUÇÃO

A solução proposta neste trabalho —*SepMerge*— visa melhorar o processo de *merge* realizado pela ferramenta *CSDiff*, oferecendo uma abordagem mais eficiente e precisa para a detecção e resolução de conflitos. Esta solução é dividida em duas partes, a primeira delas lida com o ruído criado com a adição dos marcadores durante a etapa de pré-processamento de código do *CSDiff*, a segunda parte descreve uma otimização do processo para diminuir as falhas do *CSDiff*, relacionadas ao alinhamento do código feito pelo *diff3*.

### 3.1 Remoção do uso dos marcadores para os separadores no *CSDiff*

Atualmente, *CSDiff* pré-processa os arquivos *base*, *left* e *right* adicionando uma cadeia de *marcadores* ('$$$$$$$$') e quebras de linha antes de cada *separador sintático* da linguagem de programação. O objetivo desses marcadores é dividir os contextos antes de executar o *diff3*, e servir como referência para a reestruturação das linhas após a execução do *diff3*. No entanto, foi observado que a adição desses marcadores pode introduzir uma complexidade desnecessária e interferir na precisão da detecção e resolução de conflitos. Para reduzir o número de marcadores utilizados foram modificados os processos de pré-processamento e pós-processamento dos arquivos.

### 3.2 Modificação no pré-processamento dos arquivos

Atualmente, o *CSDiff* adiciona uma cadeia de *marcadores* ($$$$$$$) e uma quebra de linha antes e depois de cada separador presente no texto durante o pré-processamento. Aqui, nós propomos remover o uso dos marcadores, mantendo apenas as quebras de linha. Considerando o exemplo do arquivo *base* da Figura 10 abaixo, as Figuras 11 (*CSDiff*) e 12 (*SepMerge*) ilustram a diferença.

```
def to_string(l: List[str]) -> str:
    if len(l) == 0:
        return ""
    return "..".join(l)
```

**Figura 10: Arquivo *base* antes do pré-processamento do *CSDiff*.**

```
def to_string
$$$$$$$(
$$$$$$$l
$$$$$$$:
$$$$$$$ List[str]
$$$$$$$)
$$$$$$$ -> str
$$$$$$$:
$$$$$$$
    if len
$$$$$$$(
$$$$$$$l
$$$$$$$)
$$$$$$$ == 0
$$$$$$$:
$$$$$$$
        return ""
    return "..".join
$$$$$$$(
$$$$$$$l
$$$$$$$)
$$$$$$$
```

**Figura 11: Arquivo *base* após pré-processamento do *CSDiff*.**

```
def to_string
(
l
:
 List[str]
)
 -> str
:

    if len
(
l
)
 == 0
:

        return ""
    return "..".join
(
l
)
```

**Figura 12: Arquivo *base* após pré-processamento do *SepMerge*.**

## 3.3 Execução focalizada do *CSDiff*

Pensando em diminuir o risco de falhas no alinhamento e pareamento das linhas durante o uso do *diff3*, foi criado um novo processo. O objetivo desse novo processo é diminuir o escopo de análise processada pelo *CSDiff*, visando, assim, diminuir o risco de falha do *diff3* no pareamento e alinhamento causado pela quantidade de linhas iguais adicionadas. Esse processo também visa eliminar a geração de novos falsos conflitos, uma vez que o *CSDiff* será executado apenas nos conflitos que já foram gerados pelo *diff3*. Dessa forma garantimos que a nova solução não seja pior que o *diff3*, o que é essencial para justificar adoção na prática da ferramenta. Em específico, o *SepMerge*:

(1) Executa o *diff3* nos arquivos *base*, *left* e *right*, com a opção `-show-all`.
(2) Para cada conflito gerado pelo *diff3*:
    (a) Cria três arquivos: *base_tmp*, *left_tmp* e *right_tmp* contendo apenas o conteúdo do conflito.
    (b) Executa o *CSDiff* modificado, sem os marcadores, nesses três arquivos.
    (c) Substitui o conflito original pelo resultado do *CSDiff*.
(3) Retorna o arquivo final, com os conflitos substituídos, como resultado da integração.

Para ilustrar o processo descrito na acima, considere a integração dos arquivos *base*, *left* e *right* mostrados anteriormente nas Figuras 2, 3, 4 respectivamente. O primeiro passo é a execução do *diff3* nesses arquivos. Em sua configuração padrão, o *diff3* sinaliza apenas o conteúdo que causou conflito das versões *left* e *right* nos conflitos reportados. No entanto, ao utilizar a opção `-show-all`, ele adiciona também o conteúdo da versão *base* do arquivo, como ilustrado na Figura 13. Essa informação adicional da versão *base* é essencial para a execução do próximo passo, onde o *SepMerge* executa o *CSDiff* apenas nos trechos de código associados aos conflitos reportados pelo *diff3*.

```
def to_string(l: List[str]) -> str:
<<<<<<< /left.py
    if l is null or len(l) == 0:
        return ""
    return "__".join(l)
||||||| /base.py
    if len(l) == 0:
        return ""
    return "..".join(l)
=======
    if len(l) == 0:
        return self.D
    return "__".join(l)
>>>>>>> /right.py
```

**Figura 13: Resultado da execução do comando *diff3 left base right –show-all*.**

Depois de executado o *diff3* com a opção `-show-all`, separamos o conteúdo de cada conflito em três novos arquivos: *base_tmp* (Figura

14), *left_tmp* (Figura 15), *right_tmp* (Figura 16). Isso possibilita a execução do *CSDiff* **focalizada** diretamente nos conflitos, isto é, levando em conta apenas os conflitos que já foram reportados pelo *diff3*. Dessa maneira, *SepMerge* remove os casos de falsos conflitos adicionados por *CSDiff* em relação ao *diff3*, uma vez que, na pior das hipóteses, a ferramenta só reportará conflitos que já foram reportados por *diff3*.

```
    if len(l) == 0:
        return ""
    return "..".join(l)
```

**Figura 14: Arquivo *base_tmp* gerado por *SepMerge*.**

```
    if l is null or len(l) == 0:
        return ""
    return "__".join(l)
```

**Figura 15: Arquivo *left_tmp* gerado por *SepMerge*.**

```
    if len(l) == 0:
        return self.D
    return "__".join(l)
```

**Figura 16: Arquivo *right_tmp* gerado por *SepMerge*.**

Depois de executar o *CSDiff* nos arquivos *base_tmp*, *left_tmp* e *right_tmp*, substituímos o conflito original (da Figura 13) pelo resultado do *CSDiff* (da Figura 17), resultando no arquivo final do integrado pelo *SepMerge* na Figura 18. Como pode ser observado, em contraste com o resultado do *diff3* original na Figura 1, *SepMerge* foi capaz de reduzir o conflito reportado pelo *diff3*, integrando a modificação realizada pela versão *left* no condicional da segunda linha. Esse resultado só foi possível por conta do uso do "`:`" como separador na execução do *CSDiff*.

```
    if l is null or len(l) == 0:
<<<<<<< /left_tmp.py
        return ""
    return "__".join(l)
=======
(         return self.D
    return "__".join(l)
>>>>>>> /right_tmp.py
```

**Figura 17: Execução do *CSDiff* nos arquivos *base_tmp*, *left_tmp* e *right_tmp*.**

A implementação do *SepMerge*, assim como as versões do *CSDiff* utilizadas nesse estudo encontram-se disponibilizadas em um repositório público no GitHub[1].

[1]https://github.com/felipebma/sepmerge

```
def to_string(l: List[str]) -> str:
    if l is null or len(l) == 0:
<<<<<<< /left.py
        return ""
    return "__".join(l)
=======
(         return self.D
    return "__".join(l)
>>>>>>> /right.py
```

**Figura 18: Resultado final do *SepMerge* nos arquivos *base*, *left* e *right*.**

## 4 AVALIAÇÃO

Para avaliar a solução proposta neste trabalho —*SepMerge*— e abordar as perguntas de pesquisa previamente mencionadas, procedemos com a replicação da análise realizada por Clementino et al. [7] e Pereira [12]. Este estudo compara os resultados obtidos ao empregar o *CSDiff* e o *SepMerge* com os obtidos com uso do *diff3*. O objetivo é analisar a capacidade de solução de conflitos sem comprometer a integridade do processo de *merge*. Para esta análise, realizamos a execução das ferramentas *CSDiff*, *SepMerge* e *diff3* em cada arquivo de cada cenário de nossa amostra.

Os projetos escolhidos para este experimento foram projetos de código aberto desenvolvidos majoritariamente na linguagem de programação *Python*, uma vez que pretendemos avaliar também as contribuições ao *CSDiff* propostas por Pereira [12] que trata do uso da mudança de indentação de código como um possível separador sintático para códigos escritos em *Python*. Além disso, estudos anteriores focaram em *Java* e mencionaram a necessidade de investigar ferramentas de integração avançadas em linguagens de diferentes naturezas [2, 5, 7]. No entanto, é importante salientar que a solução apresentada neste trabalho é independente da linguagem de programação, requerendo apenas a especificação dos separadores sintáticos da linguagem a ser suportada.

### 4.1 Conceitos

*4.1.1 Cenário de integração.* Em um sistema de controle de versão, um *commit* é uma versão que agrupa mudanças em determinados arquivos de um projeto [10]. Considerando essa definição, um cenário de integração é definido como uma quádrupla de *commits*, que chamaremos aqui de *commits base*, *left*, *right* e *merge*. O *base* representa o *commit* de onde as modificações *left* e *right* derivaram, enquanto o *merge* representa a versão final da integração de código, armazenada no repositório original de um projeto.

*4.1.2 Falso positivo adicionado.* Seguindo as mesmas definições descritas em diversos trabalhos [5–7, 12, 15], quando uma ferramenta reporta um conflito que não deveria ser considerado como um conflito, ele é chamado de falso positivo. Aqui, ao compararmos duas ferramentas, um falso positivo adicionado (aFP) será o cenário de integração, ou arquivo integrado, em que uma das ferramentas reportou algum conflito, enquanto a outra não reportou nenhum conflito, e conseguiu resolver o conflito de forma correta. Neste estudo, consideramos uma resolução de conflito como correta quando

o resultado da ferramenta é igual ao resultado final da integração no repositório do projeto em questão, isto é, igual à versão do código do *commit de merge*.

*4.1.3 Falso negativo adicionado.* Um falso negativo ocorre quando a ferramenta de integração não reporta um conflito que deveria ter sido reportado. Ao compararmos duas ferramentas, o conceito de falso negativo adicionado (aFN) será o cenário de integração, ou arquivo integrado, em que a ferramenta de integração não reportou nenhum conflito, e não resolveu os conflitos de forma correta. Aqui, também comparamos o arquivo integrado com o resultado final da integração no repositório do projeto (versão do código do *commit de merge*).

## 4.2 Perguntas de pesquisa

Para avaliar a solução proposta, foram definidas as seguintes perguntas de pesquisa:

*4.2.1 PP1: A abordagem focalizada do* SepMerge *reduz a quantidade de conflitos reportados em comparação ao* diff3 *e ao* CSDiff. Para responder essa pergunta, é necessário realizar a análise do número de conflitos reportados por cada uma das ferramentas. Para isso, executamos as ferramentas em cada conjunto de arquivos de cada um dos *commits base*, *left*, *right* de cada cenário de integração de nossa amostra. Em seguida, contamos cada bloco de conflito reportado por cada uma das ferramentas. Um bloco de conflito reportado por essas ferramentas é um conjunto de linhas envolto pelos marcadores (`>>>>>>>` e `<<<<<<<`).

*4.2.2 PP2: A abordagem focalizada do* SepMerge *reduz a quantidade de cenários de integração com conflitos reportados em comparação ao* diff3 *e ao* CSDiff*?* Um cenário é categorizado como "cenário com conflito"por uma determinada ferramenta se pelo menos um conflito for detectado no conjunto de arquivos resultantes da integração de código realizado por essa ferramenta. Para realizar essa análise, executamos cada ferramenta em todos os cenários de nossa amostra e contabilizamos quais cenários integrados resultaram em pelo menos um conflito.

*4.2.3 PP3: A abordagem focalizada do* SepMerge *reduz a quantidade de arquivos com erros de identificação de conflitos em comparação ao* diff3 *e ao* CSDiff*?* Quando comparamos duas ferramentas de integração $X$ e $Y$, um arquivo é considerado um falso positivo adicionado (aFP) para a ferramenta $X$ quando $X$ reporta pelo menos um conflito, e a ferramenta $Y$ não reporta nenhum conflito no mesmo arquivo. Além disso, o resultado da ferramenta $Y$ precisa ser condizente com o resultado presente no repositório original no projeto (obtido via o *commit de merge*). Nesses casos, é considerado que a ferramenta $Y$ foi capaz de resolver completa e corretamente os conflitos no arquivo. Em contrapartida, um arquivo é considerado falso negativo adicionado (aFN) para a ferramenta $X$ quando ele não reporta nenhum conflito, mas o arquivo integrado por $X$ não condiz com o resultado presente no *commit de merge*. Além disso, é necessário que a ferramenta $Y$ reporte conflitos no mesmo arquivo. Nesses casos, é considerado que a ferramenta $X$ resolveu incorretamente os conflitos do arquivo, resultando em um código sintática e/ou semanticamente incorreto em relação ao resultado esperado.

*4.2.4 PP4: A abordagem focalizada do* SepMerge *reduz a quantidade de cenários de integração com erros de identificação de conflitos em comparação ao* diff3 *e ao* CSDiff*?* Quando comparamos duas ferramentas de integração $X$ e $Y$ em relação aos cenários de integração, um cenário de integração com falso positivo adicionado (aFP) é aquele no qual a ferramenta $X$ reportou pelo menos um conflito em algum arquivo de um determinado cenário de integração, enquanto a ferramenta $Y$ não reportou nenhum conflito para o mesmo cenário, e o resultado de $Y$ condiz com o resultado original do projeto. Analogamente, um cenário de integração com falso negativo adicionado (aFN) é aquele no qual uma ferramenta $X$ não reportou nenhum conflito em um determinado cenário de integração, o resultado de $X$ é diferente do resultado original do projeto, e a ferramenta $Y$ reportou pelo menos um conflito no mesmo cenário.

## 4.3 Amostra

Para realizar nosso experimento e responder nossas perguntas de pesquisa, escolhemos projetos *Python* hospedados no GitHub com chances de apresentarem conflitos em em seu histórico de cenários de integração.

Na seleção de projetos, levamos em consideração o número de contribuidores do projeto e a sua popularidade no GitHub, medida através do número de estrelas do projeto. Selecionamos, então, os primeiros 9 projetos que atendiam a esses critérios para nossa amostra[2], listados na Tabela 1.

| Projeto | Estrelas | Contribuidores |
|---|---|---|
| certbot | 30.7k | 477 |
| flask | 66k | 715 |
| ipython | 16.1k | 860 |
| matplotlib | 19k | 1407 |
| salt | 13.8k | 2452 |
| scrapy | 50.4k | 552 |
| sentry | 36.5k | 667 |
| tensorflow | 181k | 3516 |
| tornado | 21.5k | 374 |

**Tabela 1: Projetos selecionados para o experimento.**

## 4.4 Metodologia

Para comparar as ferramentas de integração, identificamos e extraímos, dos projetos listados na Tabela 1, os cenários de integração de um intervalo de dois anos (entre 01/01/2021 e 01/01/2023). Dos cenários obtidos, nós filtramos apenas os cenários em que os desenvolvedores modificaram pelo menos um arquivo em comum. Esse filtro visa retirar da nossa amostra cenários onde não existia a possibilidade da ocorrência conflitos de integração, isto é, cenários em que as versões *left* e *right* modificaram conjuntos disjuntos de arquivos. Como consequência, foram selecionados 875 cenários de integração dos 10 projetos, totalizando 1855 arquivos a serem integrados pelas diferentes ferramentas de integração.

Com o objetivo de verificar e comparar a eficácia em relação à detecção e resolução de conflitos de modelos propostos por Clementino et al. [7], Pereira [12] e a solução proposta neste artigo,

[2]Com base no mecanismo de buscas do GitHub: https://github.com/search/advanced

utilizamos 5 modelos diferentes de ferramentas de integração não-estruturada, são eles:

(1) Modelo proposto por Clementino et al. [7], que não considera a indentação como um separador, com os separadores "( ) , :". Chamemos de *CSDiff*.
(2) Modelo proposto por Pereira [12], considerando a mudança de indentação como um separador, e com os separadores "( ) , :". Chamemos de *CSDiffI*.
(3) Modelo proposto neste artigo, não considerando a mudança de indentação como um separador, e com os separadores "( ) , :". Chamemos de *SepMerge*.
(4) Modelo proposto neste artigo, considerando a mudança de indentação como um separador, e com os separadores "( ) , :". Chamemos de *SepInMerge*.
(5) O *diff3*, realizando uma análise puramente textual tradicional.

Cada uma das ferramentas foi utilizada para integrar os arquivos dos cenários de integração da nossa amostra. Para cada cenário de integração, coletamos métricas a respeito do número de arquivos em cada cenário, número de arquivos com conflitos, número de cenários com conflito, falsos positivos e falsos negativos. Nossos *scripts* leem linha a linha cada um dos resultados das ferramentas, contando o número de blocos de conflitos presentes nos arquivos integrados.

Para contabilizar os dados comparativos entre as ferramentas, como, por exemplo, se ambas obtiveram o resultado esperado (i.e. igual à versão do *commit de merge*), os *scripts* comparam o conteúdo do arquivo original nos repositórios dos projetos com o resultado de cada uma das ferramentas. Todo o experimento foi executado localmente em uma máquina operando com sistema operacional Windows, com 32GB de memória RAM e um processador Intel Core i7-10750H. Toda a infraestrutura utilizada pode ser acessada no nosso apêndice online. [3]

## 5 RESULTADOS

A seguir, apresentamos os resultados obtidos organizados conforme as questões de pesquisa abordadas.

**PP1: A abordagem focalizada do *SepMerge* reduz a quantidade de conflitos reportados em comparação ao *diff3* e ao *CSDiff*?**

Para responder esta pergunta, contabilizamos a quantidade de conflitos reportados pela execução de cada uma das ferramenta de integração nos cenários de integração de nossa amostra. Como pode ser observado na Tabela 2, a quantidade de conflitos reportados com a abordagem focalizada do *SepMerge* foi 13% menor em relação ao *diff3*. Além disso, a quantidade de conflitos reportados diminui significativamente (em 40%) em relação ao *CSDiff*. Note também que o uso da mudança de indentação como um separador sintático (em *SepInMerge* e *CSDiffI*), apesar de ainda apresentar um resultado pior do que o uso apenas dos outros separadores, não causa um impacto tão negativo quanto na versão *CSDiffI*. Isso se deve ao fato de o escopo de atuação do *SepMerge* ser menor, diminuindo o risco de falha de alinhamento e pareamento entre as linhas de código.

[3] https://github.com/felipebma/sepmerge_dataset

**PP2: A abordagem focalizada do *SepMerge* reduz a quantidade de cenários de integração com conflitos reportados em comparação ao *diff3* e ao *CSDiff*?**

Além da diminuição do número de conflitos reportados pelo *SepMerge*, a Tabela 2 mostra também uma diminuição no número de arquivos integrados e cenários de integração com conflitos reportados pelo *SepMerge*. Em particular, *SepMerge* reduziu em cerca de 13% o número de arquivos com conflitos em relação ao *diff3*, e em cerca de 9% em relação ao *CSDiff*. Além disso, *SepMerge* reduziu em cerca de 11% o número de cenários com conflitos em relação ao *diff3*, e em cerca de 3% em relação ao *CSDiff*.

Pode-se observar também, na Tabela 2, que todas as ferramentas analisadas foram capazes de apresentar uma quantidade menor de arquivos e cenários com conflitos em relação ao *diff3*. Essa diminuição está de acordo com os resultados obtidos em outros trabalhos [7, 12]. Essa redução de arquivos e cenários com conflitos pode indicar uma melhora na produtividade das equipes de desenvolvimento ao utilizar o *SepMerge* em relação às outras duas alternativas (*CSDiff* e *diff3*) já que os resultados sugerem que os desenvolvedores terão que lidar com menos situações de conflito.

**PP3: A abordagem focalizada do *SepMerge* reduz a quantidade de arquivos com erros de identificação de conflitos em comparação ao *diff3* e ao *CSDiff*?**

Para responder a essa pergunta, precisamos analisar a quantidade de arquivos *aFP* e *aFN* (veja a Seção 4.1) na comparação entre as ferramentas. Nossos resultados mostram, nas Tabela 3, que a solução proposta *SepMerge*, bem como sua variante que utiliza indentação como separador sintático (*SepInMerge*), são capazes de eliminar completamente arquivos com falsos positivos em relação ao *diff3*. Esse resultado era esperado devido a atuação focalizada de *SepMerge*, uma vez que ela atua apenas nos conflitos já reportados previamente pelo *diff3*. O mesmo não pode ser observado na ferramenta original, *CSDiff*, e sua variante *CSDiffI*: elas não foram capazes de eliminar completamente os falsos positivos, e, no caso de *CSDiffI*, que leva em consideração a indentação, houve uma piora nesse aspecto.

Já em termos de arquivos com falsos negativos, os resultados mostram, nas mesmas tabelas, que há um empate entre todas as ferramentas que utilizam os separadores sintáticos, e que a indentação (em *SepInMerge* e *CSDiffI*) não faz diferença nesse aspecto. Note também que, em todos os casos, a ferramenta não-estruturada tradicional, *diff3*, leva vantagem em resultar em menos arquivos com falsos negativos. Essa é um tendência observada também em outros estudos que exploram uso da sintaxe da linguagem em ferramentas de integração para detectar e resolver conflitos [5–7, 13, 16].

**PP4: A abordagem focalizada do *SepMerge* reduz a quantidade de cenários de integração com erros de identificação de conflitos em comparação ao *diff3* e ao *CSDiff*?**

Assim como na pergunta anterior, para responder essa pergunta precisamos analisar a quantidade de cenários de integração com falsos positivos e falsos negativos adicionados na comparação entre as ferramentas. Na Tabela 4, pode-se observar que o *SepMerge* foi capaz de resolver corretamente cerca de 12% dos cenários de integração apontados pelo *diff3*. Note também que não houve alteração em relação ao uso da indentação como separador sintático. Por outro lado, dois cenários foram reportados como aFN pelo *SepMerge*.

| **Métrica** | SepInMerge | CSDiffI | SepMerge | CSDiff | diff3 |
|---|---|---|---|---|---|
| Conflitos | 329 | 726 | 323 | 537 | 373 |
| Arquivos com conflitos | 206 | 245 | 205 | 226 | 235 |
| Cenários com conflitos | 75 | 77 | 75 | 77 | 84 |

**Tabela 2: Quantidade de conflitos reportados pelas diferentes ferramentas de integração.**

Apesar de ser uma quantidade pequena em relação ao tamanho da nossa amostra, os casos de cenários de falsos negativos adicionados podem ser considerados perigosos no âmbito de desenvolvimento de software pois eles podem, de forma automatizada, adicionar *bugs* e resultados não esperados na base de código, sem interferência do desenvolvedor.

Quando comparamos o *SepMerge* com o *CSDiff*, observamos também uma pequena vantagem para o *SepMerge*. *SepMerge* remove os casos de falsos positivos adicionados por *CSDiff*, e aumenta o número de cenários resolvidos completamente de forma correta. Há também que se notar que existe uma melhora mais significativa quando usamos a mudança de indentação como separador, o que corrobora com a ideia de que o ruído causado pela adição desse separador é mais impactante quando o *CSDiff* opera em um escopo não focalizado.

## 5.1 Ameaças à validade

Os resultados apresentados neste estudo estão sujeitos a algumas potenciais ameaças à validade.

Primeiramente, os projetos analisados foram obtidos de repositórios abertos no GitHub, os quais podem ter passado por alterações em seu histórico de *commits*, resultando em possíveis perdas de cenários de integração relevantes.

Apesar de termos utilizado uma amostra considerável de cenários, abrangendo *commits* de grandes repositórios públicos ao longo de um período significativo de dois anos, é possível que o tamanho e a diversidade dessa amostra não sejam suficientes para representar completamente o vasto espectro de situações encontradas em cenários de integração.

Ademais, é importante notar que nos limitamos a projetos predominantemente escritos em Python. Portanto, não podemos garantir que a solução proposta terá o mesmo desempenho se aplicada a projetos em outras linguagens.

# 6 DISCUSSÃO

Com base nos resultados obtidos e nas questões de pesquisa abordadas na seção anterior, evidencia-se que as melhorias propostas na implementação do processo do *SepMerge* tiveram um impacto positivo. Essas melhorias permitiram alcançar um desempenho superior em comparação com as versões atuais do *CSDiff* de Clementino et al. [7] e Pereira [12].

A estratégia de adotar uma abordagem mais focalizada, utilizando o *CSDiff* exclusivamente nos conflitos gerados previamente por uma ferramenta de integração puramente textual (*diff3*), resultou em uma redução total nos falsos conflitos relatados pela solução proposta em comparação com o *diff3*, que é a ferramenta amplamente

| | SepMerge | diff3 |
|---|---|---|
| **aFP** arquivos | 0 | 22 |
| **aFN** arquivos | 8 | 0 |

| | SepInMerge | diff3 |
|---|---|---|
| **aFP** arquivos | 0 | 21 |
| **aFN** arquivos | 8 | 0 |

| | CSDiff | diff3 |
|---|---|---|
| **aFP** arquivos | 19 | 21 |
| **aFN** arquivos | 8 | 0 |

| | CSDiffI | diff3 |
|---|---|---|
| **aFP** arquivos | 36 | 19 |
| **aFN** arquivos | 9 | 0 |

**Tabela 3: Comparação dos erros de identificação de conflitos de merge em termos de falsos positivos (aFP) e falsos negativos (aFN) em *arquivos integrados*. Os números são relativos de uma ferramenta para outra, por isso uma mesma ferramenta pode apresentar números diferentes quando comparado com outra ferramenta.**

| | SepMerge | diff3 |
|---|---|---|
| **aFP** cenários | 0 | 10 |
| **aFN** cenários | 2 | 0 |

| | SepInMerge | diff3 |
|---|---|---|
| **aFP** cenários | 0 | 10 |
| **aFN** cenários | 2 | 0 |

| | CSDiff | diff3 |
|---|---|---|
| **aFP** cenários | 1 | 9 |
| **aFN** cenários | 2 | 0 |

| | CSDiffI | diff3 |
|---|---|---|
| **aFP** cenários | 1 | 8 |
| **aFN** cenários | 3 | 0 |

**Tabela 4: Comparação dos erros de identificação de conflitos de merge em termos de falsos positivos (aFP) e falsos negativos (aFN) em *cenários integrados*. Os números são relativos de uma ferramenta para outra, por isso uma mesma ferramenta pode apresentar números diferentes quando comparado com outra ferramenta.**

reconhecida e empregada pela comunidade de desenvolvedores atualmente. No entanto, não observamos uma redução em termos de falsos negativos. Quanto mais a ferramenta compreende a sintaxe do código, mais conflitos ela deixa escapar. Isso ocorre porque essas ferramentas têm uma visão muito granular do código, falhando em capturar as dependências entre diferentes partes do código. Por outro lado, o *diff3* acidentalmente captura essas dependências ao considerar o código como um todo, tratando uma linha de texto ou linhas consecutivas de texto como as unidades a serem integradas. Essa tendência de redução de falsos positivos e *não*-redução de falsos negativos também foi observada em estudos envolvendo ferramentas semi-estruturadas e estruturadas [5, 6, 13, 16]. Trabalhos futuros poderiam empregar ferramentas dessa natureza na comparação com o *SepMerge*.

Conforme observado por Pereira [12], o uso de mudanças na indentação parece gerar mais interferências do que facilitar a resolução de conflitos. Neste estudo, embora o impacto do uso de mudanças na indentação seja consideravelmente menor em comparação com outros trabalhos, ainda é negativo. A redução desse impacto pode ser atribuída à abordagem mais focalizada do *SepMerge*. *SepMerge* restringe o escopo de atuação apenas aos conflitos já gerados previamente pelo *diff3*, em vez de atuar no arquivo completo. Isso possibilita uma redução na probabilidade de ocorrência de problemas de pareamento de linhas pelo *CSDiff*, uma vez que há uma diminuição da incidência de linhas idênticas com marcadores e separadores ao longo de todo o arquivo.

Por fim, uma abordagem para lidar com a problemática causada pelo uso de mudanças na indentação seria considerar a remoção dos marcadores associados a essas mudanças antes da execução do *diff3*. Além disso, como sugestão para pesquisas futuras, explorar a substituição do *diff3* por outras ferramentas de integração não-estruturadas pode trazer novos resultados.

## 7 TRABALHOS RELACIONADOS

Vários estudos têm sido conduzidos com o intuito de compreender e aprimorar as abordagens para lidar com conflitos em cenários de integração [2, 3, 5–7, 12, 14, 16]. Este trabalho teve como base especialmente dois desses estudos anteriores.Clementino et al. [7] propôs a abordagem utilizada pelo *CSDiff* para detecção e resolução de conflitos, combinando a lógica do *diff3* com os separadores sintáticos de Java. Pereira [12], por sua vez, estendeu a ferramenta desenvolvida por Clementino et al. [7], introduzindo o conceito de utilizar mudanças na indentação como separadores sintáticos. O objetivo de Pereira [12] ao realizar essa extensão era melhorar o desempenho do *CSDiff* em linguagens com poucos separadores sintáticos, como é o caso do *Python*. Outros pesquisadores também têm buscado ampliar a aplicabilidade do *CSDiff* para outras linguagens, como demonstrado por Souza [15], que ampliou e avaliou o funcionamento do *CSDiff* em linguagens como *Ruby* e *Typescript*.

Além dos estudos sobre ferramentas de integração não- estruturadas, há também pesquisas sobre abordagens estruturadas e semiestruturadas. Um exemplo é o trabalho apresentado por Apel et al. [3], que analisou a ferramenta semiestruturada *FSTMerge*, destacando uma vantagem da abordagem semiestruturada em relação à não-estruturada no que diz respeito à geração de falsos conflitos. Ele também instigou a busca e análise de ferramentas que combinem abordagens semiestruturadas e não-estruturadas. Outro estudo relevante é o de Cavalcanti et al. [5], no qual foi realizada uma análise comparativa entre a ferramenta semiestruturada *FSTMerge* e a ferramenta não-estruturada *Kdiff3*. O estudo concluiu que as ferramentas semiestruturadas, além de reduzirem o número de falsos conflitos relatados, também tornam os conflitos relatados mais simples de serem resolvidos.

## 8 CONCLUSÃO

Neste artigo, apresentamos uma modificação na ferramenta *CSDiff*, uma ferramenta de integração de código textual, que utiliza os separadores sintáticos da linguagem de programação para melhorar a precisão da detecção e resolução de conflitos de integração da ferramenta *diff3*. Aqui, introduzimos um novo processo, denominado *SepMerge*, com o objetivo de reduzir os erros de detecção e resolução de conflitos da ferramenta original durante o processo de integração de código. Diferentemente do *CSDiff*, nossa solução adota uma abordagem focalizada, na qual ela só entra em ação em conflitos previamente detectados por *diff3*. Realizamos um experimento comparativo entre a nossa solução, a ferramenta original, e também uma outra extensão que incorpora o uso de mudança de indentação da linguagem *Python* como um possível separador sintático.

Ao reproduzir 875 cenários de integração de 9 projetos *Python*, nossos resultados indicaram que, além de reduzir a quantidade de conflitos de integração, *SepMerge* conseguiu eliminar completamente os casos de adição de arquivos e cenários com falsos conflitos (conflitos que não deveriam ter sido reportados). Além disso, *SepMerge* aumentou o número de arquivos e cenários que tiveram seus conflitos completamente resolvidos corretamente, ao mesmo tempo que diminuiu o número de arquivos e cenários com falsos negativos (conflitos que deveriam ter sido reportados, mas não foram) em comparação com as versões do *CSDiff* apresentadas em outros trabalhos. Esses resultados evidenciam a eficácia e aprimoramento proporcionados pela abordagem proposta pelo *SepMerge* na detecção e resolução de conflitos.

## DISPONIBILIDADE DE ARTEFATOS

Para apoiar a replicação e a execução de novos estudos, disponibilizamos (i) *dataset* contendo os cenários de integração da nossa amostra[4], e os resultado das ferramentas de *merge* para cada arquivo integrado, e (ii) código-fonte da ferramenta proposta[5].

## ACKNOWLEDGMENTS

Gostaríamos de agradecer ao INES (Instituto Nacional de Engenharia de Software) e às agências de fomento à pesquisa brasileiras CNPq (bolsa 309235/2021-9), FACEPE (projetos IBPG-0546-1.03/15 e APQ/0388-1.03/14) e CAPES por apoiarem parcialmente este trabalho.

## REFERÊNCIAS

[1] Paola Accioly, Paulo Borba, and Guilherme Cavalcanti. 2018. Understanding semi-structured merge conflict characteristics in open-source Java projects. *Empirical Software Engineering* (2018).

[4] https://github.com/felipebma/sepmerge_dataset
[5] https://github.com/felipebma/sepmerge

[2] Sven Apel, Olaf Leßenich, and Christian Lengauer. 2012. *Structured merge with auto-tuning: balancing precision and performance. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering - ASE 2012. ACM Press.* https://doi.org/10.1145/2351676.2351694

[3] Sven Apel, Jörg Liebig, Benjamin Brandl, Christian Lengauer, and Christian Kästner. 2011. Semistructured Merge: Rethinking Merge in Revision Control Systems. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (Szeged, Hungary) *(ESEC/FSE '11)*. Association for Computing Machinery, New York, NY, USA, 190–200. https://doi.org/10.1145/2025113.2025141

[4] Caius Brindescu, Iftekhar Ahmed, Carlos Jensen, and Anita Sarma. 2020. An empirical investigation into merge conflicts and their effect on software quality. *Empirical Software Engineering* 25, 1 (2020), 562–590.

[5] Guilherme Cavalcanti, Paulo Borba, and Paola Accioly. 2017. Evaluating and Improving Semistructured Merge. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 59 (oct 2017), 27 pages. https://doi.org/10.1145/3133883

[6] Guilherme Cavalcanti, Paulo Borba, Georg Seibt, and Sven Apel. 2019. The Impact of Structure on Software Merging: Semistructured Versus Structured Merge. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 1002–1013. https://doi.org/10.1109/ASE.2019.00097

[7] Jônatas Clementino, Paulo Borba, and Guilherme Cavalcanti. 2021. Textual merge based on language-specific syntactic separators. In *35th Brazilian Symposium on Software Engineering (SBES 2021)*. 243–252.

[8] Gleiph Ghiotto, Leonardo Murta, Márcio Barros, and Andre Van Der Hoek. 2018. On the nature of merge conflicts: a study of 2,731 open source java projects hosted by github. *IEEE Transactions on Software Engineering* 46, 8 (2018), 892–915.

[9] Sanjeev Khanna, Keshav Kunal, and Benjamin C Pierce. 2007. A formal investigation of diff3. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 485–496.

[10] Ali Koc and Abdullah Uz Tansel. 2011. A survey of version control systems. *ICEME 2011* (2011).

[11] T. Mens. 2002. A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering* 28, 5 (2002), 449–462. https://doi.org/10.1109/TSE.2002.1000449

[12] Jose Gabriel Silva Pereira. 2023. *Analise de extensão do CSDiff para uso em linguagens com poucos separadores sintaticos.* Trabalho de Conclusão de Curso. Universidade Federal de Pernambuco.

[13] Georg Seibt, Florian Heck, Guilherme Cavalcanti, Paulo Borba, and Sven Apel. 2021. Leveraging structure in software merge: An empirical study. *IEEE Transactions on Software Engineering* (2021).

[14] Bo Shen, Wei Zhang, Haiyan Zhao, Guangtai Liang, Zhi Jin, and Qianxiang Wang. 2019. IntelliMerge: a refactoring-aware software merging technique. *Proceedings of the ACM on Programming Languages* OOPSLA (2019).

[15] Heitor Sammuel Carvalho Souza. 2021. *Extensão e análise de performance da ferramenta de merge textual CSDiff para novas linguagens.* Trabalho de Conclusão de Curso. Universidade Federal de Pernambuco.

[16] Alberto Trindade Tavares, Paulo Borba, Guilherme Cavalcanti, and Sérgio Soares. 2019. Semistructured merge in JavaScript systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1014–1025.