Universidade Federal de Pernambuco
Centro de Informática

Bacharelado em Ciência da Computação

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

Felipe Benjamin Mendonça Araújo

Trabalho de Graduação

Recife
Março de 2024

Universidade Federal de Pernambuco
Centro de Informática

Felipe Benjamin Mendonça Araújo

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

Trabalho apresentado ao Programa de Bacharelado em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: *Prof. Dr. Paulo Henrique Monteiro Borba*

Recife
Março de 2024

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Araújo, Felipe Benjamin Mendonça.
Refinando a Precisão da Detecção de Conflitos: Uma Análise do CSDiff com Abordagem Focalizada / Felipe Benjamin Mendonça Araújo. - Recife, 2024.
33 p. : il., tab.

Orientador(a): Paulo Henrique Monteiro Borba
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Ciências da Computação - Bacharelado, 2024.

1. Processo de Merge. 2. Desenvolvimento Colaborativo. 3. Merge Textual. 4. Merge Estruturado. 5. Separadores Sintáticos. I. Borba, Paulo Henrique Monteiro. (Orientação). II. Título.

000 CDD (22.ed.)

Universidade Federal de Pernambuco
Centro de Informática

Felipe Benjamin Mendonça Araújo

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

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Ciência da Computação.

Aprovado em: _21_/_03_/_2024_

BANCA EXAMINADORA

_____

Prof. Dr. Paulo Henrique Monteiro Borba (Orientador)

Universidade Federal de Pernambuco

_____

Prof. Dr. Henrique Emanuel Mostaert Rebelo (Examinador Interno)

Universidade Federal de Pernambuco

Agradecimentos

Gostaria de expressar minha mais profunda gratidão a todos que estiveram ao meu lado ao longo desta jornada acadêmica. Em primeiro lugar, quero dedicar este momento aos meus pais, cujo apoio e incentivo foram fundamentais em cada etapa do meu percurso. Obrigado por nunca desistirem de mim e por serem minha fonte constante de inspiração.

Aos meus amigos Marcos, Raíssa, Nicolle e Tácio, cuja presença nos momentos mais desafiadores foi um grande suporte emocional, meu sincero agradecimento. Vocês compartilharam comigo as alegrias e as dificuldades, tornando cada obstáculo mais fácil de superar.

Aos meus colegas de universidade, que se tornaram parte essencial da minha jornada acadêmica, quero expressar minha gratidão por cada momento compartilhado. Seja nos momentos de dúvidas ou nas celebrações das conquistas, vocês estiveram lá, prontos para oferecer suporte e encorajamento.

E, por último, mas definitivamente não menos importante, a meu orientador Paulo Borba. Sua orientação, sabedoria e dedicação foram fundamentais para o sucesso deste trabalho. Obrigado por acreditar no meu potencial, por sua paciência e por me guiar ao longo deste processo de aprendizado.

A todos vocês, a minha mais profunda gratidão. Este trabalho não teria sido possível sem o apoio e o amor de cada um de vocês. Que possamos continuar a compartilhar nossas jornadas e celebrar nossas conquistas juntos.

Resumo

Nos últimos anos, o desenvolvimento de software tem se tornado cada vez mais complexo, com equipes trabalhando simultaneamente em diferentes partes do código-fonte para construir e aprimorar sistemas. No entanto, essa natureza colaborativa do desenvolvimento pode levar à geração de conflitos quando múltiplas pessoas tentam modificar o mesmo arquivo simultaneamente. Nesse cenário, ferramentas de merge e versionamento de código desempenham um papel crucial no gerenciamento desses conflitos. Uma dessas ferramentas é o *CSDiff* [1], [2], uma ferramenta de detecção de conflitos, alternativa ao *diff3*, que utiliza-se de separadores customizáveis de cada linguagem de programação para auxiliar na resolução destes conflitos. Este trabalho tem como objetivo propor uma melhoria para o *CSDiff* com foco na redução de conflitos falsos positivos e falsos negativos encontrados com o uso da ferramenta. Por meio de uma análise comparativa entre o sistema com e sem a melhoria proposta, este estudo avalia o impacto na redução de erros apresentados.

Palavras-chave: Processo de Merge, Desenvolvimento Colaborativo, Merge Textual, Merge Estruturado, Separadores sintáticos

Abstract

In recent years, software development has become increasingly complex, with teams working simultaneously on different parts of the source code to build and enhance systems. However, this collaborative nature of development can lead to conflicts when multiple people attempt to modify the same file simultaneously. In this scenario, code merge and versioning tools play a crucial role in managing these conflicts. One such tool is *CSDiff* [1], [2], a conflict detection tool, an alternative to *diff3*, which utilizes customizable separators for each programming language to assist in resolving these conflicts. This work aims to propose an enhancement for *CSDiff* focusing on reducing false positive and false negative conflicts found when using the tool. Through a comparative analysis between the system with and without the proposed enhancement, this study evaluates the impact on reducing errors presented by it.

Keywords: Merge Process, Collaborative Development, Textual Merge, Structured Merge, Syntactic Separators.

Sumário

CAPÍTULO 1

Introdução

O estado atual de desenvolvimento de softwares em larga escala demanda que uma quantidade de desenvolvedores esteja empenhadas em implementar, modificar e manter códigos armazenados em uma mesma base. Para casos como esse, torna-se imprescindível o uso de ferramentas de versionamento de código. Estas ferramentas de versionamento de código são responsáveis por gerenciar o histórico de modificações nos códigos, bem como lidar com a organização e sincronicidade das modificações realizadas em paralelo por desenvolvedores diferentes. Quando alterações são realizadas de maneira paralela em uma mesma base de código e, posteriormente são unificadas, damos a esse processo o nome de merge.

Durante um processo de merge, é possível que as duas modificações realizadas em paralelo tenham sido feitas em um mesmo arquivo, ou até mesmo no mesmo trecho do código, e quando isso ocorre, damos a este evento o nome de conflito. Se a ferramenta de versionamento de código não é capaz de lidar e resolver automaticamente este conflito, cabe aos desenvolvedores analisar as modificações feitas e investir tempo e esforço adicional para resolvê-lo [3], impactando na produtividade geral do time e, potencialmente abrindo margem para a criação de novos bugs, caso a solução para o conflito não seja realizada corretamente.

Para tentar lidar com estes conflitos, foram adicionados às ferramentas de versionamento de código sistemas de detecção e resolução automática de conflitos. A mais simples, e mais popularmente utilizada é a solução de análise puramente textual[4]. 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. A esses cenários, dá-se o nome de falsos conflitos. Para tentar lidar com a geração de falsos conflitos pela abordagem linha a linha, foram desenvolvidas abordagens estruturadas e semi-estruturadas, abordagens essas que, além da localização das modificações, utilizando-se de um conjunto de regras e símbolos da linguagem, analisam também a sintaxe das alterações envolvidas no processo de merge. A criação de uma ferramenta estruturada específica para cada linguagem porém é muito custoso, e pensando em dimimuir esse custo, foi desenvolvida a ferramenta *CSDiff* [1], que é uma ferramenta de merge não estruturada que se utiliza de um conjunto de separadores sintáticos das linguagens (símbolos utilizados para separar escopos e contextos) para realizar esta análise sintática.

A ideia do *CSDiff* é usar separadores sintáticos, além das quebras de linha, como divisores de contexto para detecção e resolução de conflitos com o *Diff3*. Para fazer isso, a ferramenta

CSDiff [2] começa a sua análise pré-processando os arquivos modificados (*left* e *right*) e a versão base adicionando novos marcadores para fazer uma separação das linhas de acordo com os separadores sintáticos das linguagens. Após o pré-processamento, faz-se uma análise linha a linha dos arquivos pré-processados com o uso do *Diff3*. Depois de detectados os conflitos nos arquivos, é realizado o pós-processamento, removendo os marcadores adicionados durante o pré-processamento. Atualmente no entanto, essas modificações de pré-processamento ocasionalmente geram um problema de alinhamento e pareamento das linhas durante o processo de merge, gerando assim novos cenários de falsos conflitos e situações em que a resolução dos conflitos, realizada automaticamente pela ferramenta não é a solução correta, potencialmente inserindo bugs na nova versão do código, pulando a etapa de verificação manual dos conflitos realizada pelos desenvolvedores.

Para reduzir esse problema este trabalho propõe uma modificação no CSDiff, com um novo processo, aqui chamado de *Separtors-based Merge (SepMerge)* em contraste com o tradicional *line-based merge (diff3)* trazendo uma abordagem mais focalizada para o *CSDiff*. E analisa o impacto dessa modificação na detecção e resolução de conflitos. Em especial, investiga 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 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 com erros de identificação de conflitos em comparação ao *Diff3* e ao *CSDiff*?

Os resultados apontam que, além de diminuir o número de conflitos reportados pelas outras duas ferramentas, o *SepMerge* foi capaz de remover completamente os casos de adição de arquivos e cenários aFP (Falsos Positivos adicionados). Além disso o *SepMerge* foi capaz de aumentar o número de arquivos e cenários que tiveram seus conflitos completamente resolvidos corretamente, e diminuir o número de arquivos e cenários aFN (Falsos Negativos adicionados) em relação às versões do CSDiff [1], [2] apresentadas em estudos anteriores.

CAPÍTULO 2

Motivação

2.1 Merge Não Estruturado

Uma das ferramentas mais amplamente adotadas para facilitar o *merge* de texto é o *Diff3* [5]. Utilizado pelo Git, um dos sistemas de controle de versão mais populares, o Diff3 é responsável por detectar e resolver conflitos que surgem durante o processo de merge. Funciona comparando três versões de um arquivo: a versão original (*base*), a versão modificada pelo desenvolvedor local (*left*) e a versão modificada por outro desenvolvedor (*right*). O Diff3 então tenta reconciliar essas alterações de forma automática, seguindo uma abordagem puramente textual baseada em linhas, mas quando ocorrem interseções entre as alterações do *left* e *right*, ele os sinaliza para intervenção manual. A estas interseções é dado o nome de *conflitos de merge* [4], e a demarcação definida pelo *Diff3* se segue através dos marcadores (``>>>>>>>``, ``=======`` e ``<<<<<<<``) como mostrado na *Figura 2.1*.

```
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 2.1 Exemplo de conflito

Pela natureza não estruturada do Diff3, essa ferramenta sinaliza como conflitos modificações feitas por dois desenvolvedores e 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 semântico do código. Nesses casos, ao invés de reportar conflito, seria mais interessante que a ferramenta de merge realizasse a integração com sucesso, ao invés de reportar um falso conflito.

Este problema poderia ser resolvido aplicando-se o uso de uma ferramenta de merge estruturado ou semi-estruturado [6]. Ao contrário da ferramenta de merge não estruturada, 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 merge, e assim resolver automaticamente os conflitos de modificações realizadas em contextos diferentes.

2.2 Merge não Estruturado com separadores

As abordagens estruturadas e semiestruturadas mencionadas na seção anterior, apesar de benéficas em algumas situações, possuem um custo adicional elevado, dado que elas se baseiam em manipulação de árvores sintáticas, que por sua vez são dependentes da linguagem em questão e demandam um esforço significativo de implementação por linguagem, além de que a manipulação das árvores sintáticas possui um custo computacional maior em relação a abordagens puramente textuais.

Para reduzir essas desvantagens, Clementino [1] propõe uma nova ferramenta chamada **Custom Separators Diff**[1] (*CSDiff*), cujo funcionamento baseia-se na abordagem puramente textual, mas que também considera a estrutura sintática do programa por meio de um conjunto de separadores da linguagem, escolhidos pelo usuário e passados como parâmetro.

A ideia trazida por Clementino [1] é mesclar o uso da abordagem linha a linha usada pelo diff3, com uma separação de contexto auxiliada pelos separadores sintáticos da linguagem.

(1) Pré-processa os arquivos base, left e right, adicionando marcadores (`$$$$$$$`) e quebras de linha antes e depois de cada um dos separadores sintáticos.
(2) Executa o diff3 nas versões modificadas do base, left e right.
(3) Remove, do resultado do diff3, os marcadores e quebras de linhas adicionadas durante o primeiro passo.

Listagem 2.1: Processo do *CSDiff*

A ***Figura 2.2*** até a ***Figura 2.9*** ilustram o processo do CSDiff descrito na *Listagem 2.1* usando como separadores os separadores sintáticos do python: *“( ) , :“*.

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

Figura 2.2 Arquivo base

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

Figura 2.3 Arquivo left

[1] https://github.com/JonatasDeOliveira/custom-separators-merge-tool/blob/main/csdiff.sh

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

Figura 2.4 Arquivo right

Primeiro o CSDiff pré-processa os arquivos *base* (**Figura 2.2**), *left* (**Figura 2.3**) e *right* (**Figura 2.4**), adicionando os marcadores para cada um dos separadores no texto, e criando 3 novos arquivos: *base_temp* (**Figura 2.5**), *left_temp* (**Figura 2.6**) e *right_temp* (**Figura 2.7**). A adição das quebras de linha são necessárias para evitar que, durante o passo 2 da *Listagem 2.1* o *Diff3* não considere modificações em contextos diferentes (divididos pelos separadores) como interseções, pois as mudanças agora não estarão mais na mesma linha, ou em linhas adjacentes por conta da linha com apenas os marcadores e o separador.

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

Figura 2.5 Arquivo base depois do passo 1 da ***Listagem 2.1***

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

Figura 2.6 Arquivo left depois do passo 1 da ***Listagem 2.1***

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

Figura 2.7 Arquivo right depois do passo 1 da ***Listagem 2.1***

Depois de adicionar os marcadores, o *CSDiff* executa o *Diff3* nos 3 arquivos modificados, gerando assim um arquivo de merge temporário (**Figura 2.8**).

```
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 2.8 Arquivo final depois do passo 2 da ***Listagem 2.1***

Por fim, o *CSDiff* remove todos os marcadores e quebras de linha adicionados durante o primeiro passo do arquivo de merge temporário, gerando o arquivo de merge resultante do processo do *CSDiff* (**Figura 2.9**).

```
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 2.9 Arquivo final depois do passo 3 da ***Listagem 2.1***

Como descrito por *Khanna*[4] uma das etapas do funcionamento do *diff3* é o processo de pareamento das linhas entre as versões *base* e *left* e entre a *base* e a *right* para identificar quais foram as modificações realizadas por cada uma dessas novas versões do arquivo. A abordagem do *CSDiff* mencionada acima, apesar de se mostrar eficiente em reduzir ou resolver conflitos com modificações em contextos diferentes, para grandes arquivos, a alta quantidade de marcadores adicionados potencialmente aumenta o risco de uma falha nesse pareamento do *Diff3* quando executado durante o segundo passo da *Listagem 2.1*, o que 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 lógica 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.

CAPÍTULO 3

Solução

3.1 Visão Geral da Implementação

A solução proposta neste trabalho 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 primeira etapa do *CSDiff*, a segunda parte descreve uma otimização do processo para diminuir as falhas do *CSDiff* relacionadas ao alinhamento do código feito pelo diff3.

3.2 Remoção do uso dos marcadores para os separadores no *CSDiff*

Atualmente, o *CSDiff* pré-processa os arquivos *base*, *left* e *right* adicionando uma cadeia de *marcadores* ('$$$$$$$$') e quebras de linha antes de cada *separador semântico* da linguagem. O objetivo desses marcadores é dividir os contextos antes de executar o *diff3*, e como referência para a reestruturação das linhas depois da execução do *diff3*. No entanto, foi observado que a adição desses marcadores pode introduzir uma complexidade desnecessária e potencialmente 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.

3.2.1 Modificação no Processo de 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.

As *Figuras 3.1, 3.2* e *3.3* ilustram a diferença.

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

Figura 3.1 Arquivo base

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

Figura 3.2 Arquivo base depois do pré-processamento do *CSDiff*

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

    if len
(
l
)
 == 0
:

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

Figura 3.3 Arquivo base depois do pré-precessamento sem marcadores

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, descrito na Listagem *3.1*. O objetivo desse novo processo é diminuir o escopo de análise processado 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* irá ser executado apenas nos conflitos gerados pelo *Diff3*.

1 Executa o *Diff3* nos arquivos *base*, *left* e *right*, com a opção *–show-all*.
2 Para cada conflito gerado no resultado do *Diff3*:
  2.1 Cria 3 arquivos: *base_tmp*, *left_tmp* e *right_tmp* com o conteúdo do conflito.
  2.2 Executa o *CSDiff* modificado, sem os marcadores, nesses 3 arquivos.
  2.3 Substitui o conflito original pelo resultado do CSDiff.
3 Retorna o arquivo final, com os conflitos substituídos, como resultado do merge.

Listagem 3.1 Processo do SepMerge

Para ilustrar o processo descrito na ***Listagem 3.1***, tomemos os arquivos *base*, *left* e *right* apresentados nas ***Figuras 3.4***, ***3.5*** e ***3.6***.

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

Figura 3.4 Arquivo base

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

Figura 3.5 Arquivo left

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

Figura 3.6 Arquivo right

O primeiro passo é a execução do *Diff3* em cima destes arquivos. Normalmente o *Diff3* aponta apenas o conteúdo 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 demonstrado na ***Figura 3.7***. Essa informação adicionada é importante para a execução do próximo passo, onde o *SepMerge* executa o *CSDiff* apenas nos trechos de código associado 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 3.7 Execução do Diff3 com opção –show-all nos arquivos base, left e right

Depois de executado o *Diff3* com a opção –show-all, separamos o conteúdo de cada conflito em 3 novos arquivos: *base_tmp* (***Figura 3.8***), *left_tmp* (***Figura 3.9***), *right_tmp* (***Figura 3.10***). Com isso podemos ter a execução do *CSDiff* mais focalizada diretamente nos conflitos e levando em conta apenas os conflitos reportados pelo *Diff3*, removendo assim os casos de falsos conflitos adicionados pela ferramenta em relação ao *Diff3*.

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

Figura 3.8 Arquivo base_tmp

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

Figura 3.9 Arquivo left_tmp

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

Figura 3.10 Arquivo right_tmp

Depois de executar o *CSDiff* nos arquivos *base_tmp*, *left_tmp* e *right_tmp*, substituímos o conflito original (***Figura 3.7***) pelo resultado do *CSDiff* (***Figura 3.11***), resultando no arquivo final do *merge* pelo *SepMerge* (***Figura 3.12***).

```
    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 3.11 Execução do CSDiff nos arquivos base_tmp, left_tmp e right_tmp

```
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 3.12 Resultado do SepMerge nos arquivos base, left e right

Vemos que no caso exposto, o *CSDiff* 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*.

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/CSDiff

CAPÍTULO 4

Avaliação

Para examinar a modificação proposta ao CSDiff, denominada aqui como *SepMerge*. e abordar as perguntas de pesquisa previamente mencionadas, procedemos com a replicação da análise realizada por Clementino [1] e Pereira [2]. 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 previamente selecionada e filtrada. Os projetos escolhidos para este experimento foram projetos desenvolvidos majoritariamente em Python, com o interesse de avaliar as mudanças no *CSDiff*[1] propostas por *Pereira*[2] do uso da mudança de indentação como um possível separador sintático para o *Python*.

4.1 CONCEITOS

4.1.1 Cenário de Merge

Em um sistema de controle de versão, um *commit* é uma versão que agrupa mudanças em determinados arquivos de um projeto [7]. Considerando essa definição, um Cenário de Merge é definido como uma quádrupla de *commits*, que chamaremos aqui de *base*, *left*, *right* e *merge*. O *base* representa o *commit* de onde as modificações *left* e *right* partiram, enquanto o *merge* representa a versão final onde a integração das mudanças foi feita no repositório.

4.1.2 Falso Positivo Adicionado

Seguindo as mesmas definições descritas em estudos anteriores [2], [8], 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 ou arquivo 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 do merge no repositório do projeto em questão.

4.1.3 Falso Negativo Adicionado

Um falso negativo ocorre quando a ferramenta de *merge* não reporta um conflito que deveria ter sido considerado. Ao compararmos duas ferramentas, o conceito de Falso Negativo Adicionado

será dado a todos os cenários ou arquivos em que a ferramenta não reportou nenhum conflito, e não resolveu os conflitos de forma correta.

4.2 PERGUNTAS DE PESQUISA

O presente trabalho tem como objetivo investigar e analisar o comportamento e eficácia do *SepMerge*. processo proposto neste artigo, em relação ao *CSDiff* [1], [2] e ao *Diff3*, ferramenta de merge não estruturada mais comumente utilizada. Para alcançar esse propósito, foram delineadas 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 realizar a análise do número de conflitos reportados por cada uma das ferramentas executamos as ferramentas em cada conjuntos de arquivos (*base*, *left*, *right*) de cada cenário de nossa amostra, e contamos cada bloco de conflito reportado por cada uma delas. Um bloco de conflito reportado pelo *Diff3* é um conjunto de linhas envolto entre os marcadores (`>>>>>>>` e `<<<<<<<`), como mostrado na *Figura 4.1*.

```
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 4.1 Exemplo de conflito do Diff3

4.2.2 PP2: A abordagem focalizada do *SepMerge* reduz a quantidade de cenários com conflitos reportados em comparação ao *Diff3* e ao *CSDiff*?

Um cenário é reportado como cenário com conflito por determinada ferramenta caso pelo menos um conflito seja detectado no conjunto de arquivos do *merge* desse cenário realizado por essa ferramenta. Para realizar essa análise, executamos cada uma das ferramentas em cada um dos cenários de nossa amostra, e contamos a quantidade de cenários com conflitos reportados por cada uma delas.

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 estamos comparando duas ferramentas de merge *X* e *Y*, um arquivo é considerado um falso positivo adicionado (aFP) para a ferramenta *X* quando ela apresenta pelo menos um conflito reportado, enquanto a ferramenta *Y* não apresenta nenhum conflito para ele, e o resultado da ferramenta *Y* é condizente com o resultado esperado (versão *merge* do arquivo) para o merge do arquivo, nesse caso 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 apresenta nenhum conflito quando processado pela ferramenta *X*, mas não condiz com o resultado esperado (versão *merge* do arquivo) para o merge do arquivo, enquanto que a ferramenta *Y* reporta corretamente os conflitos relacionados a esse arquivo, nesse caso, é considerado que a ferramenta *Y* resolveu incorretamente os conflitos do arquivo, resultando em um código sintatica ou semanticamente incorreto em relação ao resultado esperado.

4.2.4 PP4: A abordagem focalizada do *SepMerge* reduz a quantidade de cenários com erros de identificação de conflitos em comparação ao *Diff3* e ao *CSDiff*?

Quando comparamos duas ferramentas de merge *X* e *Y* em relação aos cenários, um cenário falso positivo adicionado (aFP) para a ferramenta *X* é um cenário em que nenhum arquivo reportou nenhum conflito, quando processado pela ferramenta *X*, e o resultado desse processamento foi igual ao esperado (versão *merge* do arquivo) para o merge do arquivo, enquanto que pelo menos um desses arquivo gerou um conflito quando processado pela ferramenta *Y*. Por outro lado, quando, sem reportar nenhum conflito nos arquivos, a ferramenta *X* produz um resultado diferente do esperado para pelo menos um dos arquivos do cenário, enquanto a ferramenta *Y* apresenta ao menos um arquivo com conflito, esse cenário é considerado um falso negativo adicionado (aFN) para a ferramenta *X*.

4.3 AMOSTRA

Para realizar nosso experimento e responder as perguntas de pesquisa discutidas na seção anterior, escolhemos projetos com boas chances de possuírem conflitos em cenários de merge, levando em consideração o número de contribuidores e estrelas no github. Para esse experimento foram selecionados 10 projetos, listados na Tabela 4.1. Outro critério para a escolha dos projetos, foi a utilização da linguagem Python, pois havia também o interesse de comparar o resultado obtido com as mudanças propostas nesse artigo com o modelo do *CSDiff* desenvolvido por *Pereira* [2], que avalia a mudança de indentação como um possível *separador sintático* para a utilização do *CSDiff* desenvolvido por Clementino[1] em projetos que utilizam Python. A mudança de indentação possui um papel sintático em Python de separação de contextos. Cada um dos projetos escolhidos contam com mais de 13 mil estrelas e mais de 370 contribuidores no *github*.

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 4.1 Projetos selecionados para o experimento

4.4 METODOLOGIA

Para comparar as ferramentas de merge, fizemos a mineração dos commits dos projetos listados na Tabela 4.1 utilizando o *MiningFramework*[1] e capturando os *commits* realizados em um intervalo de dois anos (entre 01/01/2021 e 01/01/2023), e foram filtrados apenas os cenários de merge em que houveram modificações em pelo menos um arquivo em comum. Essa filtragem visa retirar da nossa amostra cenários onde não existia a possibilidade de conflitos (cenários em que as versões *left* e *right* modificaram conjuntos disjuntos de arquivos). com o objetivo de verificar e comparar a eficácia dos modelos propostos por Clementino [1], Pereira [2] e o processo com as modificações propostas por este artigo, em relação à detecção e resolução de conflitos. Utilizamos então 5 modelos diferentes de ferramentas de merge não estruturado, são eles:

(1) Modelo proposto por Clementino [1], que não considera a indentação como um separador, com os separadores "( ) , :". Chamemos de *CSDiff*.
(2) Modelo proposto por Pereira [2], 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. Chamemos de *Diff3*.

Após o processo de mineração e filtragem dos cenários, cada um dos cenários foi processado por cada uma das ferramentas supracitadas, e os dados relevantes referentes ao resultado de cada uma delas foi armazenado em uma tabela diferente para cada um dos projetos. Entre os dados armazenados de cada resultado temos: Número de arquivos em cada cenário, número de arquivos com conflitos reportados em cada cenários por cada um dos modelos, número de conflitos reportados em cada cenários por cada um dos modelos, etc.

Para a obtenção dos dados relevantes foi implementado um programa em *Java* para ler os

[1] https://github.com/spgroup/miningframework

arquivos resultantes de cada ferramenta e calcular os dados, esta implementação, junto com a implementação dos modelos das ferramentas utilizadas estão disponibilizados em um repositório público no github.[2]. Para realizar a contagem de conflitos, o programa lê linha a linha de cada um dos resultados, contando o número de blocos de conflitos presentes no arquivo. Para contabilizar os dados comparativos entre as ferramentas, como por exemplo, se ambas obtiveram o resultado esperado (igual à versão *merge* do arquivo), o programa compara o conteúdo do arquivo *merge* do cenário com o resultado de cada uma das ferramentas.

Todos esses passos foram executados localmente em uma máquina operando com sistema operacional Windows, com 32GB de memória RAM e um processador Intel Core i7-10750H.

4.5 RESULTADOS

Para a amostra, depois do processo de mineração e filtragem dos cenários de merge, foram selecionados 875 cenários (Totalizando 1855 arquivos). Dos commits minerados pelo *MiningFramework*, foram filtrados apenas aqueles que possuiam pelo menos um arquivo que foi modificado pelas duas versões (*left* e *right*) a partir da versão *base*. Essa filtragem foi realizada no próprio *MiningFramework*.

4.5.1 PP1: A abordagem focalizada do *SepMerge* reduz a quantidade de conflitos reportados em comparação ao *Diff3* e ao *CSDiff*?

Para realizar essa análise, contabilizamos a quantidade de conflitos reportados pela execução de cada um dos modelos nos cenários de nossa amosta. Os resultados obtidos podem ser encontrados na Tabela 4.2.

Tipo de conflito	***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 4.2 Quantidade de conflitos encontrados após execução do experimentos

De acordo com a Tabela 4.2, podemos ver que a quantidade de conflitos reportados com a abordagem focalizada do *SepMerge* tem uma diminuição de 13% em relação ao Diff3, e diminui drasticamente (40%) em relação ao *CSDiff* [1], [2]. O estudo anterior [2] apresentou um aumento similar na relação entre o *CSDiff* e o Diff3, apresentando um aumento de 77% para a versão sem utilizar a mudança de indentação. Também é interessante observar que o uso da mudança de indentação como um marcador, apesar de ainda apresentar um resultado pior do que o uso apenas dos separadores, não causa um impacto tão grande quanto nas versões anteriores do *CSDiff* [1], [2], isso se deve ao fato do escopo de atuação do *CSDiff* quando utilizado pelo *SepMerge* ser menor, diminuindo o risco de falha de alinhamento e pareamento entre as linhas pelo Diff3.

[2]https://github.com/felipebma/CSDiff

4.5.2 PP2: A abordagem focalizada do *SepMerge* reduz a quantidade de cenários com conflitos reportados em comparação ao *Diff3* e ao *CSDiff*?

Além da diminuição do número de conflitos reportados pelo *SepMerge*, como podemos ver na Tabela 4.2, tivemos também uma diminuição dos arquivos e cenários com conflitos reportados pelo *SepMerge*. com uma redução de arquivos com conflitos de 12.7% em relação ao *Diff3* e 9.2% em relação ao *CSDiff* [1], e uma redução de cenários com conflitos de 10.7% em relação ao *Diff3* e 2.6% em relação ao CSDiff [1].

Podemos ver na Tabela 4.2 que todas as ferramentas analisadas foram capazes de apresentar uma quantidade menor de cenários e arquivos com conflitos em relação ao Diff3. Essa diminuição está de acordo com os resultados obtidos pelos estudos anteriores. Essa redução de arquivos e cenários com conflitos pode indicar uma melhora na produtividade dos times ao utilizar o *SepMerge* em relação às outras duas alternativas (*CSDiff* e Diff3).

4.5.3 PP3: A abordagem focalizada do *SepMerge* reduz a quantidade de arquivos com erros de identificação de conflitos em comparação ao *Diff3* e ao *CSDiff*?

Para responder a essa pergunta precisamos analisar a quantidade de arquivos *aFP* e *aFN* na comparação entre as ferramentas.

	SepInMerge	***Diff3***
aFP arquivos	0	21
aFN arquivos	8	0

Tabela 4.3 Resultados do *SepInMerge* em comparação ao *Diff3*

	CSDiffI	***Diff3***
aFP arquivos	36	19
aFN arquivos	9	0

Tabela 4.4 Resultados do *CSDiffI* em comparação ao *Diff3*

	SepMerge	***Diff3***
aFP arquivos	0	22
aFN arquivos	8	0

Tabela 4.5 Resultados do *SepMerge* em comparação ao *Diff3*

	CSDiff	***Diff3***
aFP arquivos	19	21
aFN arquivos	8	0

Tabela 4.6 Resultados do *CSDiff* em comparação ao *Diff3*

Quando comparamos a Tabela 4.3 com a Tabela 4.4, que representam o uso das ferramentas com a mudança de indentação sendo usada como um separador, podemos notar que o *SepInMerge* consegue resolver completamente a geração de novos arquivos aFP em relação ao *CSDiffI* , também podemos observar uma pequena melhora em relação aos aFP's gerados pelo *Diff3* em relação ao *SepInMerge* se comparado com a relação do *Diff3* ao *CSDiffI* , e uma diminuição do número de aFN's gerados pelo SepInMerge. O mesmo pode ser visto ao analisarmos a Tabela 4.5 e a Tabela 4.6, quando não utilizamos a mudança de indentação.

Quando comparamos os resultados do *SepMerge* com o *Diff3* em relação aos dados observados na Tabela 4.2, é interessante notar que que o *SepMerge* consegue resolver completamente e corretamente todos os conflitos de 9% dos arquivos reportados pelo *Diff3* e relata como aFN apenas 8 dos 1855 arquivos utilizados como amostra.

Era de se esperar também a redução de 100% dos arquivos aFP reportados pelo *SepMerge*. uma vez que ele utiliza uma abordagem focalizada, executando, e processando conflitos apenas nos conflitos apontados a priori pelo Diff3.

4.5.4 PP4: A abordagem focalizada do *SepMerge* reduz a quantidade de cenários com erros de identificação de conflitos em comparação ao *Diff3* e ao *CSDiff* ?

Assim como na *PP3*, para responder essa pergunta precisamos analisar a quantidade de cenários aFP e aFN na comparação entre as ferramentas.

Na ***Tabela* 4.9** vemos que o *SepMerge* foi capaz de resolver corretamente 11.9% dos cenários apontados pelo Diff3, não houve alteração em relação ao uso da mudança de indentação, por outro lado, dois cenários foram reportados como aFN pelo SepMerge. 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* vemos também uma pequena vantagem para o *SepMerge*. ao remover os casos de falsos positivos adicionados por ela, e aumentar 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.

	SepInMerge	***Diff3***
aFP cenários	0	10
aFN cenários	2	0

Tabela 4.7 Resultados do *SepInMerge* em comparação ao *Diff3*

	CSDiffI	***Diff3***
aFP cenários	1	8
aFN cenários	3	0

Tabela 4.8 Resultados do *CSDiffI* em comparação ao *Diff3*

	SepMerge	***Diff3***
aFP cenários	0	10
aFN cenários	2	0

Tabela 4.9 Resultados do *SepMerge* em comparação ao *Diff3*

	CSDiff	***Diff3***
aFP cenários	1	9
aFN cenários	2	0

Tabela 4.10 Resultados do *CSDiff* em comparação ao *Diff3*

4.6 DISCUSSÃO

De acordo com os resultados obtidos, e pelas perguntas de pesquisa respondidas anteriormente, podemos perceber que as melhorias propostas para a implementação do processo do *SepMerge* surtiram efeitos muito positivos, conseguindo um resultado melhor do que as versões atuais do *CSDiff* propostas por Clementino [1] e Pereira [2].

A ideia de trazer uma abordagem mais focalizada, executando o *CSDiff* apenas nos conflitos gerados previamente por uma ferramenta de merge puramente textual (*Diff3*) conseguiu reduzir em 100% dos falsos conflitos reportados pela ferramenta em relação ao *diff3*, ferramenta mais aceita e comumente utilizada atualmente pela comunidade de desenvolvedores.

Como reportado também por Pereira [2], o uso da mudança de indentação aparenta causar mais ruído do que ajudar na resolução dos conflitos. Neste estudo o impacto do uso da mudança de indentação ser muito menor, se comparado ao encontrado nos artigos anteriores, ainda é negativo. E a redução no impacto provavelmente é causada pela natureza focalizada do *SepMerge*. reduzindo o escopo de ação do *CSDiff* que antes atuava em cima do arquivo completo, o que potencializava a probabilidade da ocorrência de problemas de pareamento das linhas pelo CSDiff, uma vez que havia uma grande incidência de linhas iguais com os marcadores e separadores por todo o arquivo.

Uma possibilidade para tentar-se resolver a problemática causada pelo uso da mudança de indentação seria remover o uso dos marcadores para essa mudança antes da execução do *Diff3*. Também seria interessante, para trabalhos futuros tentar substituir o uso do *Diff3* por outras ferramentas de merge não estruturado, como indicado por *Pereira* [2] com o uso do *KDiff3*,[3] ou *wdiff*[4].

4.7 AMEAÇAS A VALIDADE

Os resultados apresentados aqui, conforme previsto, apresentam potenciais ameaças à validade. Em primeiro lugar, os projetos utilizados para este estudo foram obtidos a partir de repositórios abertos no GitHub, os quais podem ter sofrido modificações em seu histórico de commits, resultando em possíveis perdas de cenários de fusão.

Apesar de termos utilizado uma amostra considerável de cenários, capturando commits de grandes repositórios públicos e em um intervalo considerável de 2 anos, pode ser que o tamanho e diversidade dessa amostra pode não ser o suficiente para represntar com o universo de possibilidades de situações em cenários de merge.

Adiciona-se a isso o fato de termos utilizado apenas projetos que utilizam majoritariamente o python como linguagem, e com isso, não temos garantia que, se utilizada com outras linguagens, a ferramenta replicará a mesma performance.

Além disso, há a possibilidade de os programas desenvolvidos e utilizados pelo autor para auxiliar na análise e capturar os dados relevantes dos resultados das ferramentas da pesquisa conterem erros.

[3] https://github.com/KDE/kdiff3

[4] https://www.gnu.org/software/wdiff/

CAPÍTULO 5

Trabalhos Relacionados

Diversos estudos foram conduzidos com o objetivo de entender e aprimorar as abordagens para lidar com conflitos em cenários de merge. Este projeto foi baseado especialmente em 2 destes estudos prévios. Clementino[1] que propôs a abordagem utilizada pelo CSDiff para detecção e resolução de conflitos mesclando a lógica do Diff3 com os separadores sintáticos do Java, e Pereira[2] que extendeu a ferramenta desenvolvida por Clementino, adicionando o conceito de se utilizar a mudança de indentação como separador sintático. Pereira buscava com essa extensão melhorar a performance do CSDiff para linguagens com poucos separadores sintáticos, como é o caso do Python. Outros pesquisadores buscam ampliar a performance do CSDiff para outras linguagens, como foi o caso de Souza[8], ampliando e avaliando o funcionamento do CSDiff em outras linguagens, como Ruby e Typescript.

Além de estudos sobre ferramentas não estruturadas de merge, existem também estudos acerca de abordagens estruturadas e semi-estruturadas, como é o caso apresentado por Apel[9], analisando a ferramenta semi-estruturada *FSTMerge*, onde ele aponta uma vantagem da abordagem semi-estruturada em relação à não estruturada com relação à geração de falsos conflitos, e instiga a busca e análise de ferramentas que misturem uma abordagem semi-estruturada com uma não estruturada. Outro trabalho a destacar é o proposto por *Cavalcanti*[10], no qual ele faz uma análise entre a ferramenta semi-estruturada *FSTMerge*[9] em contraste com uma ferramenta não estruturada *Kdiff3*, concluindo que as ferramentas semi-estruturadas, além de diminuir o número de falsos conflitos reportados, também torna os conflitos reportados mais simples de serem resolvidos.

CAPÍTULO 6

Conclusão

Propomos nesse artigo uma modificação sobre a ferramenta CSDiff [1], com um novo processo, nomeado aqui de SepMerge, visando reduzir o número de conflitos aFP (Falsos Positivos adiconados) e aFN (Falsos Negativos adicionados) com o uso da ferramenta durante o processo de merge, para detecção e resolução de conflitos. Fizemos testes com a nova proposta de ferramenta, com a ferramenta desenvolvida por *Clementino* [1] e com a extensão dela desenvolvida por Pereira[2], que adiciona o uso da mudança de indentação do Python como um possível separador para a ferramenta. Seguimos a mesma metodologia e avaliação utilizada pelos dois trabalhos.

Os resultados mostram que, além de reduzir a quantidade de conflitos reportados, o *SepMerge* foi capaz de remover 100% dos casos de adição de arquivos e cenários aFP (Falsos Positivos adicionados). Além disso o *SepMerge* foi capaz de aumentar o número de arquivos e cenários que tiveram seus conflitos completamente resolvidos corretamente, e diminuir o número de arquivos e cenários aFN (Falsos Negativos adicionados) em relação às versões do CSDiff [1], [2] apresentadas nos estudos anteriores.

Bibliografia

[1] J. Clementino, P. Borba e G. Cavalcanti, "Textual merge based on language-specific syntactic separators," em *35th Brazilian Symposium on Software Engineering (SBES 2021)*, 2021, pp. 243–252.

[2] J. G. S. Pereira, *Análise de extensão do CSDiff para uso em linguagens com poucos separadores sintáticos*, Bacharelado, Recife, 2023.

[3] C. Brindescu, I. Ahmed, C. Jensen e A. Sarma, "An empirical investigation into merge conflicts and their effect on software quality," *Empirical Software Engineering*, v. 25, n. 1, pp. 562–590, 2020.

[4] S. Khanna, K. Kunal e B. C. Pierce, "A formal investigation of diff3," em *International Conference on Foundations of Software Technology and Theoretical Computer Science*, Springer, 2007, pp. 485–496.

[5] T. Mens, "A state-of-the-art survey on software merging," *IEEE Transactions on Software Engineering*, v. 28, n. 5, pp. 449–462, 2002. DOI: 10.1109/TSE.2002.1000449.

[6] G. Cavalcanti, P. Borba, G. Seibt e S. Apel, "The Impact of Structure on Software Merging: Semistructured Versus Structured Merge," em *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019, pp. 1002–1013. DOI: 10.1109/ASE.2019.00097.

[7] A. Koc e A. U. Tansel, "A survey of version control systems," *ICEME 2011*, 2011.

[8] H. S. C. Souza, *Extensão e análise de performance da ferramenta de merge textual CSDiff para novas linguagens*, Graduation Thesis, 2021.

[9] S. Apel, J. Liebig, B. Brandl, C. Lengauer e C. Kästner, "Semistructured Merge: Rethinking Merge in Revision Control Systems," em *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, sér. ESEC/FSE '11, Szeged, Hungary: Association for Computing Machinery, 2011, pp. 190–200, ISBN: 9781450304436. DOI: 10.1145/2025113.2025141. endereço: https://doi.org/10.1145/2025113.2025141.

[10] G. Cavalcanti, P. Borba e P. Accioly, "Evaluating and Improving Semistructured Merge," *Proc. ACM Program. Lang.*, v. 1, n. OOPSLA, out. de 2017. DOI: 10.1145/3133883. endereço: https://doi.org/10.1145/3133883.