Universidade Federal de Pernambuco
Centro de Informática

Graduação em Ciência da Computação

Extensão e análise de performance da ferramenta de merge textual CSDiff para novas linguagens

Heitor Sammuel Carvalho Souza

Trabalho de Graduação

Recife
16/12/2021

Universidade Federal de Pernambuco
Centro de Informática

Heitor Sammuel Carvalho Souza

Extensão e análise de performance da ferramenta de merge textual CSDiff para novas linguagens

Trabalho apresentado ao Programa de Graduação 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
16/12/2021

Para Devonice Carvalho, que sempre enxergou muito além
de seu tempo e de seu neto.

Agradecimentos

Agradeço a força misteriosa que rege esse universo, seja ela real ou apenas uma abstração de nosso consciente limitado. O que importa na realidade é que um dia eu, assim como todos os cientistas da computação, retornarei a esta terra e eventualmente me tornarei insumo para que outros produzam seus trabalhos de conclusão de curso.

Agradeço a minha avó, Devonice, uma figura essencial que possibilitou que eu chegasse onde jamais sonharia.

A minha mãe, Deise, que aprendeu com a mãe dela e sempre me apoiou em minhas empreitadas, sempre me incentivando a uma evolução constante, principalmente no âmbito espiritual. *Salve salve essa nega, que axé ela tem!*

Ao meu padrasto, Odair, que me deu a segurança e os consumíveis necessários para que eu pudesse focar nos meus objetivos.

A minha belíssima namorada, amiga, parceira e companheira dos altos e baixos dessa vida, Ayla, que nadou comigo nesta jornada acadêmica. Embora ambos estejam perdidos neste oceano, é melhor estar perdido com alguém.

Ao meu grande amigo e irmão Bernardo Moraes, que caso eu não tivesse aleatoriamente encontrado em uma conversa em grupo de MSN anos atrás, eu jamais teria vindo para Recife, e na remota possibilidade que isso ocorresse, eu provavelmente teria desistido no primeiro período do curso, ou perdido completamente a saúde e a sanidade. O que viesse primeiro.

Aos meus amigos de república e de curso que me impediram de ficar pobre e prematuramente insano: Diego Costa, Douglas Valério, Gabriela Holanda, Letícia Souza, Mateus Valgueiro, Ricardo Fagundes, Victor Siqueira.

A meu orientador, Paulo, que esteve sempre presente em um semestre apertado, me apoiando para que eu conseguisse entregar este trabalho em tempo.

A tantos outros que eu poderia encher todas as páginas deste trabalho somente com bajulações costuradas sob medida para cada um, mas que me limitarei a citar os nomes pelo bem da brevidade: Leonardo, Consolata, Rosa, Ailton, Belinha, Silvia, João, Juliano, Jacqueline, Pedro, Gabriel, Ícaro, Rafael, Bruno, Enzo Thiago, Arthur Daniel, Paulo Fonseca, Márcio Cornélio, Alexandre Mota, Edna Barros, Guilherme, Lailson, Renata e Fanny.

Antes de encerrar este documento, eu gostaria de fazer um último agradecimento a mim mesmo por ter aguentado firme até aqui.

Caso eu tenha esquecido de alguém, peço desculpas. Se você estiver lendo meu trabalho de graduação (o que por si só já merece um agradecimento) e esperava encontrar uma menção a seu nome, não hesite em me contactar.

Aproveitem. Axé.

Cada um cumpre o destino que lhe cumpre.
E deseja o destino que deseja;
Nem cumpre o que deseja,
Nem deseja o que cumpre.

Como as pedras na orla dos canteiros
O Fado nos dispõe, e ali ficamos;
Que a Sorte nos fez postos
Onde houvemos de sê-lo.

Não tenhamos melhor conhecimento
Do que nos coube que de que nos coube.
Cumpramos o que somos.
Nada mais nos é dado.

—FERNANDO PESSOA (Odes de Ricardo Reis)

Resumo

Conflitos de *merge* são presentes no dia a dia do desenvolvimento de software e podem causar impacto negativo na performance dos desenvolvedores. Uma das linhas de pesquisa para tentar reduzir esse problema é a evolução das ferramentas utilizadas para auxiliar no processo de *merge* automatizado. Hoje, a abordagem mais comum é o *merge* textual, onde a comparação entre os arquivos de código é feita considerando apenas as linhas do texto. No outro extremo, existe o *merge* estruturado, que compara os arquivos levando em consideração sua estrutura sintática e semântica. Neste trabalho, damos continuidade no desenvolvimento de uma ferramenta (CSDiff) que utiliza o *merge* textual mas visa simular o estruturado, utilizando os marcadores sintáticos de cada linguagem para guiar o processo de *merge*. Implementamos e mostramos uma extensão da ferramenta capaz de analisar conflitos de código nas linguagens *Ruby* e *TypeScript*, bem como fazemos uma análise de sua acurácia em *commits* de *merge* em repositórios de código aberto. Os resultados obtidos se mostram consistentes entre Java e TypeScript para o número de conflitos reportados (que aumenta), porém observamos uma variação oposta para a linguagem Ruby (o número de conflitos reportados diminui). Para ambas as linguagens, os resultados mostram uma redução nos cenários de *merge* com conflitos — isto é, a ferramenta resolve automaticamente a integração entre os arquivos utilizados para realizar o *merge* naquele mesmo *commit*, enquanto o diff3 reporta conflitos para os arquivos e requer uma resolução manual por porte do desenvolvedor. Apesar dessa redução ser muito menor em relação ao estudo similar para a linguagem Java [CBC21], os resultados mostram que o CSDiff tem um efeito positivo na redução de conflitos quando comparado com o diff3 [Fou21, KKP07], ferramenta popular para *merge* textual. A extensão do CSDiff proposta contribui para que seja possível entender melhor e proporcionar uma evolução da ferramenta, abrindo portas para novos estudos em diferentes linguagens e contextos.

Palavras-chave: merge, conflitos de merge, ferramentas de merge, estratégias de merge, merge automatizado, merge textual, merge estruturado, separatores sintáticos, análise de performance, ruby, typescript

Abstract

Merge conflicts are present on a daily basis in software development, and can cause a negative impact on the performance of the developers. One of the research tracks that try to reduce this problem is the evolution of the tools used to help on the automated merge process. Today, the most common approach is the textual merge, where the comparison between the code files is done considering only the text lines. At the other end, we have the structured merge which compares the archives taking into account their syntactic and semantic structure. In this work we continue the development of a tool (CSDiff) that uses textual merge but simulates the structured, by using the syntactic markers of each language to guide the merge process. We implement and show an extension of the tool capable of analyzing code conflicts on the Ruby and TypeScript languages, and we do also an analysis of its accuracy at merge commits on open source repositories. The results achieved are consistent between Java and TypeScript for the number of reported conflicts (which increases), but we observed an opposite variation for the Ruby language (the number of reported conflicts decreases). For both languages, the results also show a reduction of scenarios with merge conflicts — that is, the tool automatically solves the integration between the files involved in the merge process of a single commit, while diff3 reports conflicts to the files and requires a manual resolution to be made by the developer. Besides being a lot less in comparison to the similar study for the Java language [CBC21], the results show that CSDiff has a positive effect on conflict reduction when compared to diff3 [Fou21, KKP07], a popular tool for textual merge. The proposed CSDiff extension contributes to make it possible to better understand and provide an evolution of the tool, opening doors to new studies in different languages and contexts.

Keywords: merge, merge conflicts, merge tools, merge strategies, automated merge, textual merge, structured merge, syntactic separators, performance analysis, ruby, typescript

Sumário

1 Introdução **1**

2 Motivação **5**

3 Extensão Proposta **11**
3.1 Melhorias 11
3.1.1 Documentação 11
3.1.2 Parametrização 11
3.1.3 Correção da chamada interna da ferramenta diff3 12
3.2 Visão Geral da Implementação 12
3.3 Limitações 15

4 Avaliação **21**
4.1 Definições Importantes 21
4.2 Avaliação 23
4.3 Metodologia 24
4.4 Resultados 26
4.4.1 PP1 - A nova solução de merge não estruturado, utilizando separadores, reduz a quantidade de conflitos reportados em comparação ao merge puramente textual? 26
4.4.2 PP2 - A nova solução de merge não estruturado, utilizando separadores, reduz a quantidade de cenários com conflitos reportados em comparação ao merge puramente textual? 27
4.4.3 PP3 - A nova solução de merge não estruturado, utilizando separadores, reduz a quantidade de falsos conflitos e cenários com falsos conflitos reportados (falsos positivos) em comparação ao merge puramente textual? 28
4.4.4 PP4 - A nova solução de merge não estruturado, utilizando separadores, amplia a possibilidade de comprometer a corretude do código, por aumentar o número de integrações de mudanças que interferem uma na outra, sem reportar conflitos (falsos negativos), além de aumentar cenários com falsos negativos? 31
4.5 Discussão 33
4.6 Ameaças à Validade 35

5 Trabalhos Relacionados **39**

6 Conclusão **41**

Lista de Figuras

2.1 Exemplo motivante: Arquivos *base*, *left* e *right* a serem utilizados na execução do CSDiff (Ruby) 6
2.2 Exemplo motivante: Arquivo parcial gerado pelo CSDiff ao integrar os arquivos de exemplo (Ruby) 7
2.3 Exemplo Motivante: Resultado final da execução do CSDiff (Ruby) 7
2.4 Exemplo Motivante: Resultado final da execução do diff3 (Ruby) 8
2.5 Exemplo Motivante: Demonstração de execução do CSDiff (TypeScript) 9
2.6 Exemplo Motivante: Comparação da execução do CSDiff e diff3 (TypeScript) 10

3.1 Mensagem de ajuda do CSDiff 11
3.2 Exemplo de execução do CSDiff com o parâmetro -s 12
3.3 Arquivos *base*, *left* e *right* a serem utilizados na execução do CSDiff 13
3.4 Passo 1 da execução do CSDiff nos arquivos de exemplo - Arquivo *base.rb* 14
3.5 Passo 1 da execução do CSDiff nos arquivos de exemplo - Arquivo *left.rb* 15
3.6 Passo 1 da execução do CSDiff nos arquivos de exemplo - Arquivo *right.rb* 16
3.7 Passo 2 da execução do CSDiff nos arquivos de exemplo 17
3.8 Passo 3 da execução do CSDiff nos arquivos de exemplo 18
3.9 Passo 4 da execução do CSDiff nos arquivos de exemplo 19
3.10 Resultado final da execução do CSDiff nos arquivos de exemplo 20
3.11 Resultado final da execução do diff3 nos arquivos de exemplo 20

4.1 Exemplo de falso positivo adicionado pelo diff3 (arquivos *base*, *left*, *right*) 24
4.2 Exemplo de falso positivo adicionado pelo diff3 (resultado da integração) 25
4.3 Exemplo de falso negativo adicionado pelo CSDiff (*base.ts*) 26
4.4 Exemplo de falso negativo adicionado pelo CSDiff (*left.ts*) 27
4.5 Exemplo de falso negativo adicionado pelo CSDiff (*right.ts*) 28
4.6 Exemplo de falso negativo adicionado pelo CSDiff (execução diff3) 29
4.7 Exemplo de falso negativo adicionado pelo CSDiff (execução CSDiff) 30
4.8 Comando utilizado para executar o experimento de Ruby no *miningframework* 30
4.9 Comando utilizado para executar o experimento de TypeScript no *miningframework* 31
4.10 Exemplo do aumento de conflitos pelo CSDiff (arquivo diff3) 37
4.11 Exemplo do aumento de conflitos pelo CSDiff (arquivo CSDiff) 38

Lista de Tabelas

4.1 Repositórios analisados para medir a acurácia do CSDiff 31
4.2 Quantidade de conflitos encontrados (Ruby) 32
4.3 Quantidade de conflitos encontrados (TypeScript) 32
4.4 Quantidade de falsos positivos e falsos negativos (Ruby) 33
4.5 Quantidade de falsos positivos e falsos negativos (TypeScript) 33
4.6 Acurácia da Ferramenta CSDiff em comparação ao diff3 (Ruby) 34
4.7 Acurácia da Ferramenta CSDiff em comparação ao diff3 (TypeScript) 34

CAPÍTULO 1

Introdução

O desenvolvimento de software paralelo onde dois ou mais desenvolvedores realizam alterações em um mesmo arquivo não é incomum, sendo na verdade muitas vezes necessário para agilizar o desenvolvimento. Essa prática pode causar conflitos de *merge*, geralmente devido a uma alteração simultânea feita por dois desenvolvedores diferentes em uma mesma linha ou em linhas consecutivas. Ao tentar integrar as duas versões do código com o intuito de gerar uma terceira versão com ambas as modificações, é possível que a ferramenta de *merge* automático utilizada pelos sistemas de controle de versão não seja capaz de dizer qual modificação deva ser mantida (visto que estão na mesma linha), tampouco unir ambas as mudanças, mesmo que elas não interfiram uma com a outra. Isso se deve a limitação do *merge* textual, onde a comparação entre as versões diferentes de um mesmo arquivo de código é feita linha a linha apenas.

Uma outra abordagem possível que pode mitigar conflitos de *merge* quando não há interferência nas modificações feitas por ambos os desenvolvedores (isto é, a ordem em que as alterações conflitantes são integradas não faz diferença semântica, ou quando a alteração é feita em uma mesma linha porém em regiões diferentes e não conflitantes entre si. É importante ressaltar que nem todos os conflitos devem ser evitados — em especial os que realmente dizem respeito a mudanças conflitantes e necessitam de correção manual) é a de *merge* estruturado, que analisa o código de forma mais profunda: tal técnica analisa a árvore sintática do código para determinar se as mudanças feitas pelos desenvolvedores conflitam entre si. Apesar de proporcionar uma análise mais refinada [CBSA19], essa abordagem se torna custosa por ser sensível a linguagem utilizada e necessita de uma implementação específica para cada linguagem sendo analisada [CBSA19, ALB+11].

Considerando isto, a proposta do presente trabalho é dar continuidade aos estudos iniciais e ao desenvolvimento da ferramenta CSDiff, uma ferramenta de *merge* mais robusta que o textual, que leva em consideração os separadores sintáticos específicos da linguagem, de forma a simular o *merge* estruturado. Em resumo, o CSDiff avalia não apenas as linhas modificadas consecutivamente no arquivo como o *merge* textual tradicional, mas também leva em consideração contextos distintos dentro do código que podem ser isolados antes da comparação textual, utilizando os separadores sintáticos específicos daquela linguagem. O estudo inicial, feito para a linguagem *Java*, mostrou resultados positivos - ainda que pouco expressivos [CBC21]. Mesmo assim, por mostrar uma melhoria na redução de falsos conflitos em comparação ao *merge* textual tradicional, sem impacto significativo na performance de execução, tais resultados abriram portas para melhorias na solução proposta, bem como sua extensão para suporte de novas linguagens, a fim de obter mais dados e métricas para investigar mais profundamente sobre seu desempenho e eficácia como ferramenta de *merge*.

Assim, este trabalho estende o CSDiff feito para a linguagem Java de forma que possa

ser executado para também outras linguagens (no caso, Ruby e TypeScript, mas a extensão feita possibilita que qualquer outra linguagem seja utilizada facilmente), e analisa a acurácia da ferramenta CSDiff em comparação com a ferramenta *diff3* — que é uma ferramenta de merge puramente textual - na redução de conflitos de *merge* em repositórios de código aberto das linguagens Ruby e TypeScript. Para a comparação consistente dos resultados obtidos no estudo original [CBC21], iremos responder as mesmas perguntas de pesquisa:

- **PP1** A nova solução de *merge* não estruturado, utilizando separadores (CSDiff), reduz a quantidade de conflitos reportados em comparação ao *merge* puramente textual?
- **PP2** A nova solução de *merge* não estruturado, utilizando separadores(CSDiff), reduz a quantidade de cenários com conflitos reportados em comparação ao *merge* puramente textual?
- **PP3** A nova solução de *merge* não estruturado, utilizando separadores (CSDiff), reduz a quantidade de falsos conflitos e cenários com falsos conflitos reportados (falsos positivos) em comparação ao *merge* puramente textual?
- **PP4** A nova solução de *merge* não estruturado, utilizando separadores (CSDiff), amplia a possibilidade de comprometer a corretude do código, por aumentar o número de integrações de mudanças que interferem uma na outra, sem reportar conflitos (falsos negativos), além de aumentar cenários com falsos negativos?

Os resultados dos experimentos executados no *miningframework*, bem como as respostas para essas perguntas de pesquisa são apresentadas no Capítulo 4. Os arquivos contendo uma análise mais detalhada dos falsos positivos e falsos negativos, bem como os dados brutos resultantes da execução dos experimentos foram adicionados a um repositório[1] público no GitHub do autor.

Ao fim dos experimentos e análise dos dados, observamos que, para **Ruby**, a ferramenta CSDiff consegue reduzir a quantidade de conflitos identificados em 9.63%; e de cenários com conflitos em 16.32%, embora consiga atingir um resultado final de *merge* idêntico ao arquivo de *merge* do repositório em apenas 0.05% mais casos em comparação com o diff3. Observou-se também a redução de falsos positivos adicionados[2] ao utilizar o CSDiff (80% menos arquivos e cenários), e não identificamos a presença de nenhum falso negativo adicionado.

Para **TypeScript**, CSDiff apresenta um resultado idêntico ao *merge* do repositório em 0.46% mais casos, além de mostrar um aumento no número de conflitos reportados em 41.97% e uma redução no número de cenários com conflitos de 13.78%, em comparação ao uso da ferramenta diff3. Notamos também uma diminuição significativa na quantidade de falsos positivos adicionados (94.54% menos arquivos e 91.83% menos cenários), mas, diferentemente de Ruby, houveram dois falsos negativos adicionados pelo CSDiff.

[1] https://github.com/heitorado/tg-heitor-2021

[2] Um falso positivo adicionado diz que a ferramenta em questão reportou um conflito quando não deveria reportar, visto que a outra ferramenta (no caso, diff3) não reportou e acertou o resultado do merge final do repositório. Mais detalhes sobre estas e outras definições podem ser encontradas no Capítulo 4.

Foi possível visualizar que existe uma variação em relação ao aumento/redução de conflitos. Para Ruby, observamos uma redução na quantidade de conflitos identificados, e para TypeScript, observamos que houve um aumento. Comparando com o estudo anterior do CSDiff para a linguagem Java, observamos que os resultados obtidos para TypeScript seguem na mesma direção: Para a linguagem Java, houve um aumento de 105.8% no número de conflitos reportados e uma redução de 5.21% para cenários com conflitos (no estudo anterior, este foi o experimento executado com o maior conjunto de separadores, porém o experimento com menor conjunto também segue a mesma direção: Um aumento de 34.79% no número de conflitos e uma redução de 2.01%). A causa para o CSDiff ter reportado menos conflitos para a linguagem Ruby ainda não está clara.

Os resultados obtidos confirmam pontos fortes (como a capacidade de reduzir falsos positivos e resolver conflitos em uma mesma região de código de forma automática) e fracos (como o problema de alinhamento quando *left* ou *right* adicionam ou removem grandes porções de código, alterando a "forma" do arquivo original) do CSDiff em linguagens diferentes das utilizadas no estudo original. Ainda existe espaço para melhorias e pesquisa futura, em especial para tratar do problema de alinhamento que é o principal responsável pelo surgimento de falsos positivos e falsos negativos por parte do CSDiff.

CAPÍTULO 2

Motivação

Em trabalho anterior, Clementino [CBC21] propõe e analisa o CSDiff, uma ferramenta de *merge* desenvolvida com o intuito de remover falsos conflitos (como quando um desenvolvedor A modifica o nome do método em determinada linha e o desenvolvedor B modifica o nome de um dos parâmetros deste mesmo método em uma mesma linha. Ainda que as mudanças não sejam conflitantes aos olhos humanos, ferramentas de *merge* textual acusam conflito devido as mudanças terem sido feitas na mesma linha - independente da semântica da mudança. Dessa forma, é possível acelerar o processo de integração de código. A ferramenta chama atenção por sua simplicidade (implementada em seu protótipo usando apenas comandos UNIX) e eficácia em resolver automaticamente *falsos conflitos*. O estudo, feito para a linguagem Java, mostra resultados positivos, ainda que pouco expressivos - o que abre portas para o questionamento de se os resultados obtidos no estudo realizado para a linguagem Java são generalizáveis para outras linguagens. Com as implementações adicionais feitas neste estudo para a ferramenta CSDiff, temos recursos suficientes para replicar o estudo, agora com uma versão do CSDiff que tem a capacidade de processar diferentes linguagens. O potencial da ferramenta para resolver conflitos em que o diff3 não conseguiria é mostrado da Figura 2.1 até a Figura 2.4 na linguagem Ruby, e um exemplo mais curto é mostrado para a linguagem TypeScript na Figura 2.5 e na Figura 2.6. Ambos os motivos abstraem partes do funcionamento interno do CSDiff, que será explicado em mais detalhes no Capítulo 3. Podemos observar nestes pequenos exemplos que o CSDiff adiciona mais capacidades ao *merge* textual por ser capaz de comparar blocos de código separados por contextos delimitados pelos separadores sintáticos.

```ruby
# ------------------------------------------------------------
# base.rb
# ------------------------------------------------------------
def even_or_odd(x)
  x&1 == 0 ? "#{x} is even" : "#{x} is odd"
end

# ------------------------------------------------------------
# left.rb
# ------------------------------------------------------------
def even_or_odd(num)
  num&1 == 0 ? "#{num} is even" : "#{num} is odd"
end

# ------------------------------------------------------------
# right.rb
# ------------------------------------------------------------
def even_or_odd(x)
  x&1 == 0 ? "#{x} is Even" : "#{x} is Odd"
end
```

Figura 2.1 Arquivos *base*, *left* e *right* a serem utilizados na execução do CSDiff. Note que o desenvolvedor que alterou o arquivo *left.rb* modificou o nome e as respectivas chamadas da variável utilizada como parâmetro da função. O desenvolvedor que alterou o arquivo *right.rb* por sua vez alterou dois caracteres, nas duas possíveis strings retornadas pela função

```
# [Using separators: 'do begin rescue end ( ) { } [ ] , |']

# ------------------------------------------------------------
# mid_merged.rb
# ------------------------------------------------------------
def even_or_odd
$$$$$$$(
$$$$$$$num
$$$$$$$)
$$$$$$$
  num&1 == 0 ? "#
$$$$$$${
$$$$$$$num
$$$$$$$}
$$$$$$$ is Even" : "#
$$$$$$${
$$$$$$$num
$$$$$$$}
$$$$$$$ is Odd"

$$$$$$$end
$$$$$$$
```

Figura 2.2 Arquivo parcial gerado pelo CSDiff ao integrar os arquivos de exemplo. Após seccionar o código nos separadores sintáticos, executamos a ferramenta diff3, que gera este arquivo.

```
# [Using separators: 'do begin rescue end ( ) { } [ ] , |']

# ------------------------------------------------------------
# csdiff.rb
# ------------------------------------------------------------
def even_or_odd(num)
  num&1 == 0 ? "#{num} is Even" : "#{num} is Odd"
end
```

Figura 2.3 Resultado final da execução do CSDiff nos arquivos de exemplo. Note que o conflito que seria reportado pelo *diff3* no corpo do método foi resolvido automaticamente.

```
# ------------------------------------------------------------
# diff3.rb
# ------------------------------------------------------------
<<<<<<< rb_test/left.rb
def even_or_odd(num)
  num&1 == 0 ? "#{num} is even" : "#{num} is odd"
=======
def even_or_odd(x)
  x&1 == 0 ? "#{x} is Even" : "#{x} is Odd"
>>>>>>> rb_test/right.rb
end
```

Figura 2.4 Resultado final da execução do diff3 nos arquivos de exemplo. Note que a declaração e o corpo da função são marcados como possuindo conflito, visto que a ferramenta é incapaz de unir ambas as mudanças, embora elas sejam independentes e não conflitantes entre si, apesar de residirem na mesma linha de código.

```
// ------------------------------------------------------------
// base.ts
// ------------------------------------------------------------
type MyType = {
  myAttribute1: string
  myAttribute2: number
}

function attributeOrDefaultString(attr: Pick<MyType, 'myAttribute1'> | null | undefined) {
  return attr ?? 'DEFAULT_STRING'
}
// ------------------------------------------------------------
// left.ts
// ------------------------------------------------------------
type MyType = {
  myAttribute1: string
  myAttribute2: number
}

function attributeOrDefaultString({myAttribute1}: MyType | null | undefined) {
  return myAttribute1 ?? 'DEFAULT_STRING'
}
// ------------------------------------------------------------
// right.ts
// ------------------------------------------------------------
type MyType = {
  myAttribute1: string
  myAttribute2: number
}

function attributeOrDefaultString(attr: Pick<MyType, 'myAttribute1'> | undefined) {
  return attr ?? 'ATTR_DEFAULT'
}
```

Figura 2.5 Arquivos *base*, *left* e *right* a serem utilizados na execução do CSDiff. Note que o desenvolvedor que alterou o arquivo *left.ts* modificou a variável utilizada primeiro parâmetro da função e atualizou o nome da variável alterada no retorno do método. O desenvolvedor que alterou o arquivo *right.ts* por sua vez alterou a `string` padrão que seria um dos possíveis valores de retorno da função, além de remover o tipo `null` do primeiro parâmetro da função

```
// [Using separators: '( ) [ ] , ; { } < > | ??']

// ------------------------------------------------------------
// csdiff.ts
// ------------------------------------------------------------
type MyType = {
  myAttribute1: string
  myAttribute2: number
}

function attributeOrDefaultString({myAttribute1}: MyType | undefined) {
  return myAttribute1 ?? 'ATTR_DEFAULT'
}

// ------------------------------------------------------------
// diff3.ts
// ------------------------------------------------------------
type MyType = {
  myAttribute1: string
  myAttribute2: number
}

<<<<<<< ts_test/left.ts
function attributeOrDefaultString({myAttribute1}: MyType | null | undefined
    ) {
  return myAttribute1 ?? 'DEFAULT_STRING'
=======
function attributeOrDefaultString(attr: Pick<MyType, 'myAttribute1'> |
    undefined) {
  return attr ?? 'ATTR_DEFAULT'
>>>>>>> ts_test/right.ts
}
```

Figura 2.6 Comparação dos arquivos de saída obtidos após a execução do CSDiff e diff3 nos arquivos *base.ts*, *left.ts* e *right.ts*

CAPÍTULO 3

Extensão Proposta

3.1 Melhorias

Para realizar o estudo da performance do CSDiff nas linguagens Ruby e TypeScript, propomos a seguinte extensão[1] do programa original.[2] As modificações e melhorias realizadas são descritas a seguir.

3.1.1 Documentação

Para melhorar a clareza do programa para realização deste e de futuros estudos, foi adicionada documentação no código da ferramenta CSDiff, em forma de comentários ao longo da implementação para esclarecer o comportamento do código, bem como introduzindo um texto de ajuda acessível quando executado o programa via linha de comando fazendo uso do parâmetro `-h`, como pode ser observado na Figura 3.1.

```
------------------------------------------------------------
Syntax: csdiff [-options] MYFILE OLDFILE YOURFILE
options:
-h                    Print this Help.
-s "<separators>"     Specify the list of separators,
                      e.g. "{ } ( ) ; ,"
------------------------------------------------------------
```

Figura 3.1 Mensagem de ajuda do CSDiff ao executar o programa na linha de comando com o parâmetro `-h`

3.1.2 Parametrização

O protótipo original do CSDiff possuía os separadores utilizados no estudo *hardcoded* em sua implementação, isto é, mudar o conjunto de separadores (seja para um conjunto menor dentro de uma mesma linguagem ou para outra linguagem) somente era possível alterando diretamente o código fonte. Isso causava uma complexidade adicional para realizar novos estudos com a ferramenta. Nesta nova extensão, adicionamos o parâmetro `-s` que permite que um conjunto

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

arbitrário de separadores seja informado ao executar o CSDiff. Internamente, o código se ajusta dinamicamente para realizar quantas substituições e quebras forem necessárias, de acordo com o número de separadores. Um exemplo de chamada utilizando esse parâmetro, com alguns separadores da linguagem TypeScript, seria como observado na Figura 3.2.

```
1 csdiff -s '( ) { } < > | : ??' mine.ts older.ts yours.ts
```

Figura 3.2 Exemplo de execução do CSDiff com o parâmetro `-s`. Veja que, para o processamento correto, os separadores devem ser informados entre aspas e separados entre si por um espaço em branco

3.1.3 Correção da chamada interna da ferramenta diff3

No código original, a ferramenta diff3 é chamada apenas com o parâmetro `-m`, que, segundo a documentação[3] gera um quarto arquivo, sendo este o arquivo com o merge resultante. Entretanto, a ferramenta diff3, em seu comportamento padrão, não realiza o merge como os sistemas de controle de versão o fazem automaticamente. Por exemplo, vamos assumir um cenário de merge que tem os arquivos *base*, *left* e *right*. O primeiro (*base*) diz respeito ao ancestral comum de *left* e *right*, enquanto *left* e *right* são arquivos modificados a partir da versão existente em seu ancestral comum (*base*) por desenvolvedores diferentes. Caso esses desenvolvedores realizem a mesma mudança na mesma região do código, o *merge* automático dos sistemas de controle de versão considera que não existe um conflito, pois se a mesma mudança foi feita por ambos os desenvolvedores, muito provavelmente existe um consenso de que aquela modificação naquela região deve estar no resultado final do arquivo de *merge*.

O diff3, entretanto, não se comporta assim por padrão: caso apenas o arquivo base seja diferente enquanto *left* e *right* são iguais entre si, um conflito é reportado[4]. Como esse comportamento não é desejado em nossa análise, visto que estamos analisando commits de um sistema de controle de versão que possui o comportamento descrito acima que destoa do diff3, é preciso fazer uma pequena alteração: adicionando o parâmetro `-E`[5], conseguimos fazer com que o diff3 se comporte como o merge feito automaticamente pelos sistemas de controle de versão, isto é, incorporando no arquivo final as mudanças idênticas nos arquivos *left* e *right* ao invés de reportar um conflito, visto que, no cenário que aqui analisamos, a resolução de tal conflito é tida como óbvia.

3.2 Visão Geral da Implementação

Em linhas gerais, o CSDiff comporta-se, a menos das melhorias supracitadas, da mesma forma como no estudo original. Para a descrição do funcionamento que se segue, foi usado o conjunto de separadores da linguagem Ruby "`( ) do`". O passo a passo descrito a seguir é também

[3] https://man7.org/linux/man-pages/man1/diff3.1.html
[4] https://www.gnu.org/software/diffutils/manual/html_node/diff3-Merging.html
[5] https://www.gnu.org/software/diffutils/manual/html_node/Which-Changes.html

ilustrado da Figura 3.3 até a Figura 3.10. Através do uso de comandos `sed`[6] para manipulação de texto, recursivamente quebra-se o texto dos três arquivos de código antes e depois de cada separador utilizado, inserindo uma nova linha e os símbolos `$$$$$$$`. Estes símbolos serão usados posteriormente para restaurar o código para sua forma original.

```ruby
# ------------------------------------------------------------
# base.rb
# ------------------------------------------------------------
def all_upcase(arr)
  arr.map do |el|
    el.upcase
  end
end

# ------------------------------------------------------------
# left.rb
# ------------------------------------------------------------
def upcase_array(arr)
  arr.map do |str|
    str.upcase
  end
end

# ------------------------------------------------------------
# right.rb
# ------------------------------------------------------------
def all_upcase(string_array)
  string_array.map do |s|
    s.upcase
  end
end
```

Figura 3.3 Arquivos *base*, *left* e *right* a serem utilizados na execução do CSDiff.

Com a quebra do texto ao redor dos separadores, aumentamos a quantidade de linhas do código. Dessa forma, na etapa seguinte, quando o `diff3 -E -m` é executado sobre os arquivos, temos uma comparação textual que simula uma comparação estruturada [CBC21], pois embora o diff3 ainda compare linha a linha, devido ao uso do comando `sed` anteriormente, inserimos quebras de linha a fim de separar contextos e escopos diferentes dentro do código (que geralmente são delimitados por separadores sintáticos). A partir deste ponto, duas etapas ainda são necessárias: a primeira, mais direta, é remover os símbolos `$$$$$$$` inseridos pelo comando `sed`.

Ainda que isso possa ser resolvido com apenas mais um comando `sed`, é possível que o código não seja completamente restaurado para sua sintaxe original, devido as anotações de conflito inseridas pelo diff3 quando o arquivo ainda continha os símbolos adicionados `$$$$$$$`.

[6]https://linux.die.net/man/1/sed

```
# # [Using separators: '( ) do']

# base.rb
# -------------------------------------------------------------
def all_upcase
$$$$$$$(
$$$$$$$arr
$$$$$$$)
$$$$$$$
  arr.map
$$$$$$$do
$$$$$$$ |el|
    el.upcase
  end
end
# -------------------------------------------------------------
```

Figura 3.4 Passo 1 da execução do CSDiff nos arquivos de exemplo: quebra das linhas do arquivo *base.rb* ao redor dos separadores sintáticos fornecidos para o CSDiff

Além disso, é necessário corrigir os nomes dos arquivos utilizados na anotação dos conflitos, uma vez que o diff3 está trabalhando com arquivos temporários. Para melhor entendimento de qual arquivo contém cada mudança, precisamos corrigir para os nomes originais. Um exemplo disso pode ser observado na Figura 3.9. Dessa forma, mais uma sequência de comandos `sed` se faz necessária para estes últimos ajustes, cujo resultado final será escrito em um arquivo de nome `csdiff` com a mesma extensão dos arquivos fornecidos e na mesma pasta onde encontram-se os três arquivos originais. Um exemplo de resultado final pode ser observado na Figura 3.10, e o resultado da execução do diff3 no mesmo conjunto de arquivos, para comparação, pode ser observado na Figura 3.11.

Entretanto, é importante ressaltar que existe um *bug*, aparentemente inerente à forma como o CSDiff é implementado, que impacta no resultado final do código, deixando-o em uma forma que prejudica sua sintaxe original, visto que devido as quebras de linha inseridas ao redor dos separadores, um erro que acontece em uma parte específica de uma linha de código será transformado em uma nova linha própria para marcação do conflito, visto que o *diff3* reporta conflitos linha a linha. Isto acontece a depender do conjunto de separadores utilizados. Identificar e remover o separador mitiga o problema, mas não é a solução ideal. Uma possível solução seria adicionar mais capacidades ao programa original, como mapear o conteúdo textual com o número de cada linha em que o mesmo se encontra, ajude a resolver este problema. Outra sugestão seria pensar em formas alternativas de anotação de conflitos de *merge*. Isto foge, entretanto, ao escopo e ao tempo disponível para realização deste estudo. Um exemplo do *bug* acima citado pode ser observado no arquivo *csdiff.rb* na Figura 3.10.

Sumarizando o que foi apresentado nesta Seção o CSDiff possuía alguns pontos de melhora que foram tratados neste trabalho, os principais sendo:

- Generalização do programa para que aceitasse qualquer conjunto de separadores sintáti-

```
# [Using separators: '( ) do']

# ------------------------------------------------------------
# left.rb
# ------------------------------------------------------------
def upcase_array
$$$$$$$(
$$$$$$$arr
$$$$$$$)
$$$$$$$
  arr.map
$$$$$$$do
$$$$$$$ |str|
    str.upcase
  end
end
```

Figura 3.5 Passo 1 da execução do CSDiff nos arquivos de exemplo: quebra das linhas do arquivo *left.rb* ao redor dos separadores sintáticos fornecidos para o CSDiff

cos (a versão anterior utilizava os separadores os '`{ } ( ) ; ,`' de maneira *hardcoded* para a linguagem Java) e qualquer extensão de arquivo;

- Correção do uso do comando *diff3* feito internamente pela ferramenta. Sem a *flag* -E,[7] seu comportamento difere do merge feito comumente pelos sistemas de controle de versão e impacta na performance do CSDiff;

- Generalização e melhoria do ambiente de experimento, feito utilizando a ferramenta *miningframework*,[8] de forma a aceitar novos parâmetros relevantes para este estudo e futuros, bem como adição de documentação.[9]

3.3 Limitações

É importante ressaltar que o CSDiff ainda se encontra em uma fase inicial de desenvolvimento e estudos sobre sua performance. Algumas de suas principais limitações estão listadas a seguir.

- Assim como no estudo inicial, o *script* do CSDiff está limitado a ser executado somente em sistemas operacionais baseados no UNIX devido ao seu uso de comandos *Bash*. Adicionalmente, devido a recursos de linguagem recentes utilizados neste trabalho, a versão do *Bash* deve ser igual ou superior à 4.2.

[7] A opção -E seleciona apenas alterações em que *left* e *right* diferem; ignorando as alterações de *base* para *right* onde *left* e *right* são idênticos. *Fonte: https://www.gnu.org/software/diffutils/manual/html_node/Which-Changes.html*

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

[9] https://github.com/spgroup/miningframework/pull/142

```
1  # [Using separators: '( ) do']
2
3  # ------------------------------------------------------------
4  # right.rb
5  # ------------------------------------------------------------
6  def all_upcase
7  $$$$$$$(
8  $$$$$$$string_array
9  $$$$$$$)
10 $$$$$$$
11   string_array.map
12 $$$$$$$do
13 $$$$$$$ |s|
14     s.upcase
15   end
16 end
```

Figura 3.6 Passo 1 da execução do CSDiff nos arquivos de exemplo: quebra das linhas do arquivo *right.rb* ao redor dos separadores sintáticos fornecidos para o CSDiff

- Assim como no estudo inicial, o caractere '`\n`' sempre fará parte do conjunto de separadores.
- Assim como no estudo inicial, a ferramenta não distingue o uso de separadores utilizados efetivamente como separadores sintáticos do uso dos mesmos símbolos (caracteres) dentro de *strings*, comentários, etc.

```
# [Using separators: '( ) do']

# ------------------------------------------------------------
# mid_merged.rb
# ------------------------------------------------------------
def upcase_array
$$$$$$$(
$$$$$$$string_array
$$$$$$$)
$$$$$$$
  string_array.map
$$$$$$$do
<<<<<<< ./folder/left.rb_temp.rb
$$$$$$$ |str|
    str.upcase
=======
$$$$$$$ |s|
    s.upcase
>>>>>>> ./folder/right.rb_temp.rb
  end
end
```

Figura 3.7 Passo 2 da execução do CSDiff nos arquivos de exemplo: resultado após a execução do `diff3 -E -m` nos códigos que foram seccionados através da sequência de símbolos '$$$$$$$'

```
# [Using separators: '( ) do']

# ------------------------------------------------------------
# mid_merged.rb
# ------------------------------------------------------------
def upcase_array
$$$$$$$(
$$$$$$$string_array
$$$$$$$)
$$$$$$$
  string_array.map
$$$$$$$do
<<<<<<< ./folder/left.rb_temp.rb
$$$$$$$ |str|
    str.upcase
=======
$$$$$$$ |s|
    s.upcase
>>>>>>> ./folder/right.rb_temp.rb
  end
end

# ------------------------------------------------------------
# merged.rb
# ------------------------------------------------------------
def upcase_array(string_array)
  string_array.map do
<<<<<<< ./folder/left.rb_temp.rb |str|
    str.upcase
======= |s|
    s.upcase
>>>>>>> ./folder/right.rb_temp.rb
  end
end
```

Figura 3.8 Passo 3 da execução do CSDiff nos arquivos de exemplo: remoção dos símbolos '$$$$$$$' do arquivo resultante da execução do comando `diff3 -E -m`. O arquivo *merged.rb* mostra o resultado da remoção. Note que há alguns problemas com a anotação adicionada pelo *diff3*, que serão corrigidas no próximo passo.

```
# [Using separators: '( ) do']

# ----------------------------------------------------------------
# merged.rb
# ----------------------------------------------------------------
def upcase_array(string_array)
  string_array.map do
<<<<<<< ./folder/left.rb_temp.rb |str|
    str.upcase
======= |s|
    s.upcase
>>>>>>> ./folder/right.rb_temp.rb
  end
end

# ----------------------------------------------------------------
# csdiff.rb
# ----------------------------------------------------------------
def upcase_array(string_array)
  string_array.map do
<<<<<<< ./folder/left.rb
 |str|
    str.upcase
=======
 |s|
    s.upcase
>>>>>>> ./folder/right.rb
  end
end
```

Figura 3.9 Passo 4 da execução do CSDiff nos arquivos de exemplo: correção das quebras de linha incorretas devido as anotações introduzidas pelo *diff3*. O arquivo *merged.rb* mostra a versão anterior à correção.

```
# [Using separators: '( ) do']

# ------------------------------------------------------------
# csdiff.rb
# ------------------------------------------------------------
def upcase_array(string_array)
  string_array.map do
<<<<<<< ./folder/right.rb
 |s|
    s.upcase
=======
 |str|
    str.upcase
>>>>>>> ./folder/left.rb
  end
end
```

Figura 3.10 Resultado final da execução do CSDiff nos arquivos de exemplo. Note que o conflito que seria reportado pelo *diff3* na primeira linha foi resolvido automaticamente, entretanto o mesmo não foi possível dentro do bloco de loop (entre `do...end`) devido a mudanças simultâneas exatamente na mesma região em ambos os arquivos *left.rb* e *right.rb*. Note também o *bug* introduzido pelo CSDiff: Como até antes do comando `do` não há conflito, houve uma quebra de linha para que fosse possível reportar apenas a região com conflito. Isso requer um trabalho manual extra para restaurar a sintaxe do código.

```
# ------------------------------------------------------------
# diff3.rb
# ------------------------------------------------------------
<<<<<<< ./folder/left.rb
def upcase_array(arr)
  arr.map do |str|
    str.upcase
=======
def all_upcase(string_array)
  string_array.map do |s|
    s.upcase
>>>>>>> ./folder/right.rb
  end
end
```

Figura 3.11 Resultado final da execução do diff3 nos arquivos de exemplo. Note que diferentemente da execução do CSDiff, as três primeiras linhas do método estão marcadas com conflito, porém o conflito detectado pelo diff3 nas duas primeiras linhas poderia ser resolvido automaticamente pelo CSDiff.

CAPÍTULO 4

Avaliação

Para avaliar a eficácia do CSDiff para Ruby e TypeScript, utilizamos o *miningframework* para extrair *commits de merge* de dez repositórios, cinco para cada uma das linguagens avaliadas. Feito isso, extraímos métricas relevantes para mensurar a acurácia do CSDiff em relação ao diff3, calculando o número de falsos positivos adicionados, falsos negativos adicionados, e número de conflitos para cada linguagem. Neste capítulo, descrevemos este processo. Iniciamos apresentando algumas definições importantes na Seção 4.1. Em seguida explicamos o processo de execução do experimento na Seção 4.2 e a escolha dos parâmetros para o estudo na Seção 4.3. Finalizamos apresentando os resultados e respondendo as perguntas de pesquisa na Seção 4.4, discutimos sobre eles na Seção 4.5, e apresentamos possíveis ameaças à validade deste trabalho na 4.6.

Os arquivos contendo a análise dos falsos positivos adicionados e falsos negativos adicionados, bem como os dados brutos resultantes da execução dos experimentos foram adicionados a um repositório público[1] no GitHub.

4.1 Definições Importantes

Aqui definimos alguns termos chave que serão frequentemente usados ao longo deste trabalho. São eles:

- **Cenário de Merge:** Sendo um *commit* em sistemas de controle de versão um agrupamento de mudanças em um conjunto de arquivos dentro daquele projeto, um cenário de *merge* refere-se a uma *quádrupla de commits*, sendo eles:
  - ***base***: Ancestral comum de *left* e *right*. Representa o estado da versão do conjunto de arquivos que compõem o repositório antes das modificações implementadas por *left* e *right*.
  - ***left***: Contém mudanças feitas em um conjunto de arquivos, sendo que ao menos um desses arquivos também foi modificado em *right*. Tem *base* como ancestral em algum ponto da árvore de *commits*.
  - ***right***: Contém mudanças feitas em um conjunto de arquivos, sendo que ao menos um desses arquivos também foi modificado em *left*. Tem *base* como ancestral em algum ponto da árvore de *commits*.

[1] https://github.com/heitorado/tg-heitor-2021

  - ***merge commit***: Filho direto dos *commits left* e *right*. Resultado da integração de *base*, *left* e *right*.

  Desta forma, um cenário pode ser visto como uma *quádrupla de commits*, que agrupam uma ou mais *quádruplas de arquivos* (ver item a seguir), cada uma dessas referente a um arquivo proveniente de *base*, modificado por *left* e *right*, e finalmente integrado no arquivo final de *merge*. Por exemplo, em um cenário onde três arquivos foram modificados em um ramo de desenvolvimento, ao realizar o *merge* deste ramo com outro, serão executados três processos de *merge*: um para cada arquivo, todos dentro do mesmo cenário.

- **Quádruplas de Arquivos:** Uma quádrupla de arquivos refere-se ao conjunto de quatro arquivos que compõem o processo de merge: *base*, o ancestral comum; *left* e *right*, os arquivos modificados paralelamente e que partiram de *base*, e por fim, o arquivo de *merge* propriamente dito, que é o arquivo resultante do processo de *merge* entre os três arquivos anteriores.

- **Conflito de merge:** Um conflito de merge surge quando existem modificações divergentes entre os arquivos *left* e *right*. Por exemplo, no caso de desenvolvimento paralelo de funcionalidades por dois desenvolvedores, eles podem modificar a mesma linha do código de maneiras diferentes, de forma que a ferramenta de merge automatizado é incapaz de decidir qual o resultado final esperado da junção das modificações feitas por ambos os desenvolvedores. Quando detectado tal cenário, a ferramenta reporta um conflito, inserindo uma notação especial no código que indica a região onde houve o conflito e as mudanças feitas por ambos os desenvolvedores.

- **Merge final do repositório:** O merge final do repositório refere-se ao arquivo resultante do processo de merge dentro de uma quádrupla de arquivos, e que está no repositório do GitHub. Este arquivo final pode ter sido integrado com sucesso de forma automática pela ferramenta, isto é, sem reportar conflitos, ou pode ser que tenha sido necessária uma modificação ou ajuste manual por parte do desenvolvedor para resolver o conflito. Vale ressaltar que o desenvolvedor pode não só apenas resolver o conflito como também adicionar novas linhas de código não relacionadas ao merge (e.g. ele pode ter percebido um *bug* enquanto corrigia o merge e aproveitou a mudança para inserir uma correção)

- **Falso Positivo Adicionado (ou AFP - *Added False Positive*):** Um falso positivo adicionado (em relação a uma ferramenta de merge, quando comparando-a com outra) diz que a ferramenta em questão reportou um conflito quando não deveria reportar, visto que a outra ferramenta (com a que estamos comparando-a) não reportou e o resultado da integração é idêntico ao merge final do repositório. Por exemplo: Um falso positivo adicionado do CSDiff em relação ao diff3 ocorre quando:

  1. O CSDiff reporta conflito
  2. O resultado do diff3 é igual ao merge final do repositório
  3. O diff3 não reporta conflito

Um exemplo de AFP do diff3 (onde o diff3 reporta conflito, mas o CSDiff não reporta e obtém resultado semanticamente correto) pode ser visto na Figura 4.1 e na Figura 4.2

- **Falso Negativo Adicionado (ou AFN - *Added False Negative*):** Um falso negativo adicionado (em relação a uma ferramenta de *merge*, quando comparando-a com outra) diz que a ferramenta em questão não reportou um conflito (quando deveria tê-lo reportado), produziu um resultado de integração que diverge do merge final do repositório, e a outra ferramenta reportou conflito. Falsos negativos são mais difíceis de encontrar, geralmente exigindo uma análise manual. Para entender o porquê, primeiro apresentamos as características necessárias (porém não determinantes) para um falso negativo. Vamos usar como exemplo um falso negativo do CSDiff. Ele *poderá* ocorrer quando:

  1. O CSDiff não reporta conflito
  2. O resultado do CSDiff é diferente do merge final do repositório
  3. O diff3 reporta conflito

  Note que o item 2 não implica o erro do CSDiff. Como mencionado anteriormente, pode ser que o resultado não seja igual ao merge final do repositório devido a alterações manuais feitas pelo desenvolvedor, e não necessariamente por um falso negativo. Caso isso ocorra, o que observamos na verdade é um *falso positivo adicionado* pela outra ferramenta, no caso do exemplo, o diff3, já que o CSDiff teria realizado o *merge* corretamente, e apenas divergiu do *merge* final do repositório devido a alterações manuais feitas no arquivo. Por isso, podemos obter pela análise inicial dos dados apenas *possíveis falsos negativos*, sendo necessária uma análise manual caso a caso para confirmar. Um exemplo de falso positivo adicionado pelo CSDiff em comparação com o diff3 pode ser visto da Figura 4.3 até a Figura 4.7

4.2 Avaliação

Nesta seção, explicamos o processo de execução do experimento e apresentamos os resultados obtidos. Essa avaliação tem como objetivo determinar o impacto que a ferramenta CSDiff, com as melhorias implementadas neste trabalho, em resolver conflitos de merge para duas novas linguagens ainda não exploradas para a ferramenta: *Ruby* e *TypeScript*. Analisamos cinco projetos *open-source* hospedados na plataforma *GitHub* para cada linguagem. A lista de repositórios e os motivos da escolha de cada um serão detalhados na Seção 4.3.

Através do uso do *miningframework*, foram coletados *commits de merge* em um intervalo de tempo pré definido para cada um dos cinco repositórios, que explicamos com mais detalhes na Seção 4.3. A ferramenta também é responsável por executar, para cada *cenário de merge*, ambas as ferramentas *diff3* (com as opções `-e -M`) e CSDiff, comparando com o resultado final presente no repositório. No fim da execução, temos os *cenários de merge* armazenados localmente com as quádruplas de arquivos que compõem merge, além de arquivos gerados pelo diff3 e CSDiff, a fim de ajudar na avaliação da acurácia: *left*, *base*, *right*, *merge*, *diff3*, *csdiff*.

```ruby
# ------------------------------------------------------------
# base.rb
# ------------------------------------------------------------
def even_or_odd(x)
  x&1 == 0 ? "#{x} is even" : "#{x} is odd"
end

# ------------------------------------------------------------
# left.rb
# ------------------------------------------------------------
def even_or_odd(num)
  num&1 == 0 ? "#{num} is even" : "#{num} is odd"
end

# ------------------------------------------------------------
# right.rb
# ------------------------------------------------------------
def even_or_odd(x)
  x&1 == 0 ? "#{x} IS EVEN" : "#{x} IS ODD"
end
```

Figura 4.1 Exemplo de falso positivo adicionado pelo diff3. Método simples que tem como objetivo retornar uma *string* dependendo se o número é par ou ímpar. O desenvolvedor em *left* modificou o nome da variável utilizada como parâmetro na definição do método e também no corpo. O desenvolvedor em *right* alterou as *strings* de retorno do método de forma que todos os caracteres agora sejam maiúsculos.

O *miningframework* também gera uma planilha com métricas iniciais ao estudo: contagem de commits, quantidade de cenários, quantos conflitos foram detectados por cada ferramenta, que arquivos são iguais entre si, etc.

Através da aplicação de filtros nesta tabela, podemos obter a contagem de *falsos positivos adicionados* e possíveis *falsos negativos adicionados* para cada uma das ferramentas utilizadas para fazer o merge. Para confirmar a real quantidade de falsos negativos, e também para explicar as causas dos falsos positivos, é necessária uma análise manual, passando por cada *cenário de merge*.

4.3 Metodologia

Inicialmente, definimos certas restrições para a escolha de repositórios que serão utilizados para mensurar a acurácia do CSDiff nas duas linguagens escolhidas. São elas:

- **Contagem de estrelas (*stars*):** O repositório deve possuir mais de dez mil estrelas no *GitHub*. Esta é apenas uma métrica inicial para indicar a popularidade e atividade do repositório.
- **Porcentagem de uso da linguagem:** O repositório deve utilizar no mínimo 60% da linguagem que se deseja estudar com ele. Por exemplo, em um projeto composto de

```
# [Using separators:  'do end ( ) { } [ ] ,']
# ------------------------------------------------------------
# merged.rb (csdiff)
# ------------------------------------------------------------
def even_or_odd(num)
  num&1 == 0 ? "#{num} IS EVEN" : "#{num} IS ODD"
end

# ------------------------------------------------------------
# merged.rb (diff3)
# ------------------------------------------------------------
<<<<<<< rb_test/left.rb
def even_or_odd(num)
  num&1 == 0 ? "#{num} is even" : "#{num} is odd"
=======
def even_or_odd(x)
  x&1 == 0 ? "#{x} IS EVEN" : "#{x} IS ODD"
>>>>>>> rb_test/right.rb
end
```

Figura 4.2 Exemplo de falso positivo adicionado pelo diff3, comparando a integração feita pelo CSDiff e pelo diff3. Note que o CSDiff não reporta conflito e obtém um resultado semanticamente correto, enquanto o diff3 reporta conflito. Isso contabiliza um falso positivo adicionado para o diff3 em comparação com o CSDiff.

Python e TypeScript, a linguagem TypeScript deve compor no mínimo 60% do código

- **Relevância e uso dos projetos:** Nesta última métrica qualitativa, o projeto deve se mostrar relevante para a comunidade de software em geral, servindo a algum propósito prático quando se desenvolve software na linguagem em questão.

A Tabela 4.1 mostra os repositórios analisados e as restrições quantitativas cumpridas por cada um.

Para iniciar o experimento, executamos o *miningframework* na lista de repositórios proposta. Para os repositórios de TypeScript, filtramos todos os commits de merge entre 01/01/2020 e 31/10/2021. Para os repositórios de Ruby, o intervalo escolhido foi de 01/01/2018 a 31/10/2021. O período para os repositórios de Ruby foi maior devido a um resultado inicial pouco expressivo quando utilizando o mesmo intervalo do experimento de TypeScript, isso porque foram obtidos menos *commits de merge* para a linguagem Ruby neste mesmo intervalo. Assim, para aumentar a quantidade de dados disponíveis e possibilitar a construção de afirmações mais sólidas, o período utilizado é maior. Os comandos executados para iniciar o experimento de cada linguagem podem ser vistos na Figura 4.8, para o experimento de Ruby, e na Figura 4.9, para o experimento de TypeScript. A documentação dos comandos utilizados e suas funções podem ser encontradas no repositório da ferramenta.

Em posse das tabelas geradas pelo *miningframework*, analisamos e agrupamos os dados em tabelas que discriminam os resultados obtidos em relação ao número de conflitos, falsos

```
// ------------------------------------------------------------
// base.ts
// ------------------------------------------------------------
const CTRL_LETTER_OFFSET = 64;

if (BrowserFeatures.clipboard.readText) {
  actionRegistry.registerWorkbenchAction(SyncActionDescriptor.from(TerminalPasteAction, {
    primary: KeyMod.CtrlCmd | KeyCode.KEY_V,
    win: { primary: KeyMod.CtrlCmd | KeyCode.KEY_V, secondary: [KeyMod.CtrlCmd | KeyMod.Shift | KeyCode.KEY_V] },
    linux: { primary: KeyMod.CtrlCmd | KeyMod.Shift | KeyCode.KEY_V }
  }, KEYBINDING_CONTEXT_TERMINAL_FOCUS), 'Terminal: Paste into Active Terminal', category,
      KEYBINDING_CONTEXT_TERMINAL_PROCESS_SUPPORTED);
  // An extra Windows-only ctrl+v keybinding is used for pwsh that sends ctrl+v directly to the
  // shell, this gets handled by PSReadLine which properly handles multi-line pastes. This is
  // disabled in accessibility mode as PowerShell does not run PSReadLine when it detects a screen
  // reader.
  if (platform.isWindows) {
    registerSendSequenceKeybinding(String.fromCharCode('V'.charCodeAt(0) - CTRL_LETTER_OFFSET), { // ctrl+v
      when: ContextKeyExpr.and(KEYBINDING_CONTEXT_TERMINAL_FOCUS, ContextKeyExpr.equals(
      KEYBINDING_CONTEXT_TERMINAL_SHELL_TYPE_KEY, WindowsShellType.PowerShell), CONTEXT_ACCESSIBILITY_MODE_ENABLED.negate()),
      primary: KeyMod.CtrlCmd | KeyCode.KEY_V
    });
  }
}
```

Figura 4.3 Exemplo de falso negativo adicionado pelo CSDiff (arquivo *base.ts*)

positivos e falsos negativos de cada ferramenta. Os resultados são elaborados na seção a seguir.

4.4 Resultados

Esta seção apresenta os resultados obtidos com a execução dos experimentos para as duas linguagens, Ruby e TypeScript, respondendo as perguntas de pesquisa apresentadas anteriormente sob a luz dos dados obtidos.

4.4.1 PP1 - A nova solução de merge não estruturado, utilizando separadores, reduz a quantidade de conflitos reportados em comparação ao merge puramente textual?

Para responder esta pergunta, analisamos a quantidade de conflitos por cenário obtidos da execução dos experimentos. Os resultados dos conflitos identificados para ambas as ferramentas, *diff3* e *CSDiff*, podem ser vistos na Tabela 4.2 para Ruby e na Tabela 4.3 para TypeScript.

Para **TypeScript**, observamos um resultado que vai na mesma direção do estudo feito para a linguagem Java, para todas as cinco métricas apresentadas na tabela: devido a forma com que processa os arquivos, inserindo diversas quebras de linha para separar as diferente seções de código entre os separadores sintáticos, há um aumento na quantidade de comparações que serão feitas posteriormente pelo diff3, e, consequentemente, do número de conflitos (nesse caso, observamos um aumento de 41.97%). Em outras palavras, um mesmo conflito do diff3 pode representar dois ou mais conflitos do CSDiff - há um aumento na granularidade dos conflitos [CBC21]. Um exemplo disso pode ser visto na Figura 4.10 e na Figura 4.11.

Já para **Ruby**, observamos que quatro das cinco métricas vão na mesma direção do estudo anterior feito para a linguagem Java e também das métricas obtidas para TypeScript. Em uma métrica específica observamos o comportamento oposto: houve uma redução de 9.63% em relação a quantidade total de conflitos apontada pelo CSDiff em comparação com o diff3.

```
// ------------------------------------------------------------
// left.ts
// ------------------------------------------------------------
const CTRL_LETTER_OFFSET = 64;

if (BrowserFeatures.clipboard.readText) {
  actionRegistry.registerWorkbenchAction(SyncActionDescriptor.from(TerminalPasteAction, {
    primary: KeyMod.CtrlCmd | KeyCode.KEY_V,
    win: { primary: KeyMod.CtrlCmd | KeyCode.KEY_V, secondary: [KeyMod.CtrlCmd | KeyMod.Shift | KeyCode.KEY_V] },
    linux: { primary: KeyMod.CtrlCmd | KeyMod.Shift | KeyCode.KEY_V }
  }, KEYBINDING_CONTEXT_TERMINAL_FOCUS), 'Terminal: Paste into Active Terminal', category,
     KEYBINDING_CONTEXT_TERMINAL_PROCESS_SUPPORTED);
  // An extra Windows-only ctrl+v keybinding is used for pwsh that sends ctrl+v directly to the
  // shell, this gets handled by PSReadLine which properly handles multi-line pastes. This is
  // disabled in accessibility mode as PowerShell does not run PSReadLine when it detects a screen
  // reader.
  if (platform.isWindows) {
    registerSendSequenceKeybinding(String.fromCharCode('V'.charCodeAt(0) - CTRL_LETTER_OFFSET), { // ctrl+v
      when: ContextKeyExpr.and(KEYBINDING_CONTEXT_TERMINAL_FOCUS, ContextKeyExpr.equals(
      KEYBINDING_CONTEXT_TERMINAL_SHELL_TYPE_KEY, WindowsShellType.PowerShell), CONTEXT_ACCESSIBILITY_MODE_ENABLED.negate()),
      primary: KeyMod.CtrlCmd | KeyCode.KEY_V
    });
  }

  if (platform.isLinux) {
    actionRegistry.registerWorkbenchAction(SyncActionDescriptor.from(TerminalPasteSelectionAction, {
      linux: { primary: KeyMod.Shift | KeyCode.Insert }
    }, KEYBINDING_CONTEXT_TERMINAL_FOCUS), 'Terminal: Paste selection into Active Terminal', category,
     KEYBINDING_CONTEXT_TERMINAL_PROCESS_SUPPORTED);
  }
}
```

Figura 4.4 Exemplo de falso negativo adicionado pelo CSDiff (arquivo *left.ts*). Note que a intenção do desenvolvedor foi adicionar uma segunda condicional abaixo da condicional mais interna que estava presente em *base.ts* (Figura 4.3).

Foi feita uma análise manual dos cenários onde a quantidade de conflitos reportada pelo diff3 eram maiores que a quantidade de conflitos reportados pelo CSDiff (exemplo: um cenário onde existam duas quádruplas de arquivos: O CSDiff consegue resolver os conflitos de uma dessas quádruplas, enquanto o diff3 reporta conflito, e na outra quádrupla ambas as ferramentas reportam conflito. Isto fará com que para o cenário, diff3 tenha reportado dois conflitos, enquanto CSDiff apenas um). Ainda que as causas para esse comportamento de redução de conflitos reportados não estejam muito claras, foi constatado que a maioria dos casos ocorreram devido a falsos positivos do diff3 em certos arquivos do cenário, causados por mudanças feitas nas mesmas regiões do código. Na maioria desses casos, o resultado do CSDiff não convergiu com o *merge* do repositório, devido a modificações manuais feitas no resultado final. Além disso, mesmo que o resultado convergisse esses eventos ocorreram em apenas um arquivo dentro de cenários com em média 3 arquivos, assim, esses cenários não são contabilizados na resolução bem sucedida de conflitos pelo CSDiff, por terem ocorrido em um único arquivo e não no cenário como um todo.

4.4.2 PP2 - A nova solução de merge não estruturado, utilizando separadores, reduz a quantidade de cenários com conflitos reportados em comparação ao merge puramente textual?

Podemos aproveitar os dados da Tabela 4.2 e da Tabela 4.3 para responder esta pergunta. Olhando para a terceira linha de ambas as tabelas, notamos que há uma redução na quantidade de cenários com conflitos. Para Ruby, essa redução foi de 16.32%, enquanto que para

```
// ---------------------------------------------------------
// right.ts
// ---------------------------------------------------------
// The text representation of `^<letter>` is `'A'.charCodeAt(0) + 1`.
const CTRL_LETTER_OFFSET = 64;

if (BrowserFeatures.clipboard.readText) {
  actionRegistry.registerWorkbenchAction(SyncActionDescriptor.from(TerminalPasteAction, platform.isMacintosh && platform.
      isWeb ? undefined : {
    primary: KeyMod.CtrlCmd | KeyCode.KEY_V,
    win: { primary: KeyMod.CtrlCmd | KeyCode.KEY_V, secondary: [KeyMod.CtrlCmd | KeyMod.Shift | KeyCode.KEY_V] },
    linux: { primary: KeyMod.CtrlCmd | KeyMod.Shift | KeyCode.KEY_V }
  }, KEYBINDING_CONTEXT_TERMINAL_FOCUS), 'Terminal: Paste into Active Terminal', category,
      KEYBINDING_CONTEXT_TERMINAL_PROCESS_SUPPORTED);
}

// An extra Windows-only ctrl+v keybinding is used for pwsh that sends ctrl+v directly to the
// shell, this gets handled by PSReadLine which properly handles multi-line pastes. This is
// disabled in accessibility mode as PowerShell does not run PSReadLine when it detects a screen
// reader. This works even when clipboard.readText is not supported.
if (platform.isWindows) {
  registerSendSequenceKeybinding(String.fromCharCode('V'.charCodeAt(0) - CTRL_LETTER_OFFSET), { // ctrl+v
    when: ContextKeyExpr.and(KEYBINDING_CONTEXT_TERMINAL_FOCUS, ContextKeyExpr.equals(
      KEYBINDING_CONTEXT_TERMINAL_SHELL_TYPE_KEY, WindowsShellType.PowerShell), CONTEXT_ACCESSIBILITY_MODE_ENABLED.negate()),
    primary: KeyMod.CtrlCmd | KeyCode.KEY_V
  });
}
```

Figura 4.5 Exemplo de falso negativo adicionado pelo CSDiff (arquivo *right.ts*). Note que a intenção do desenvolvedor foi extrair a condicional "`if (platform.isWindows)`" que se encontrava aninhada (como pode ser visto no arquivo *base.ts* na Figura 4.3) na condicional mais acima um nível de escopo acima, deixando ambas as condicionais no mesmo escopo. O desenvolvedor também adicionou um comentário na primeira linha, e acrescentou mais caracteres à linha de comentário diretamente acima da última condicional.

TypeScript esta redução foi de 13.78%. No estudo anterior feito para a linguagem Java, foi obtida uma redução de 5.21% para o experimento com o conjunto maior de separadores, e uma redução de 2.01% para o conjunto menor de separadores.

Também há uma redução de arquivos com conflitos, como se pode observar na segunda linha da tabela - 18.46% menos arquivos com conflitos para Ruby e 19.86% menos arquivos com conflitos para TypeScript. Esses valores indicam que o conjunto de separadores escolhido performa bem para a resolução automática de conflitos, reduzindo os conflitos em arquivos e também em cenários de *merge* como um todo. No estudo anterior feito para a linguagem Java, essa redução foi de 0.83% para o conjunto maior de separadores e 0.2% para o conjunto menor.

4.4.3 PP3 - A nova solução de merge não estruturado, utilizando separadores, reduz a quantidade de falsos conflitos e cenários com falsos conflitos reportados (falsos positivos) em comparação ao merge puramente textual?

Para responder a essa pergunta, realizamos a contagem de aFP's e aFN's, que podem ser visualizadas na Tabela 4.4 para Ruby, e na Tabela 4.5 para TypeScript.

Analisando os resultados obtidos para o experimento realizado na linguagem **Ruby**, notamos que o CSDiff apresenta uma redução de 80% de falsos positivos em relação ao diff3, sendo capaz de integrar código corretamente sem conflitos em 8 dos 10 cenários/arquivos analisados (como a quantidade de quádruplas de arquivos é igual a quantidade de cenários, podemos falar de ambos intercambiavelmente, já que cada cenário deve ter pelo menos uma quádrupla

```
// ------------------------------------------------------------
// diff3.ts
// ------------------------------------------------------------
// The text representation of `^<letter>` is `'A'.charCodeAt(0) + 1`.
const CTRL_LETTER_OFFSET = 64;

if (BrowserFeatures.clipboard.readText) {
  actionRegistry.registerWorkbenchAction(SyncActionDescriptor.from(TerminalPasteAction, platform.isMacintosh && platform.
      isWeb ? undefined : {
    primary: KeyMod.CtrlCmd | KeyCode.KEY_V,
    win: { primary: KeyMod.CtrlCmd | KeyCode.KEY_V, secondary: [KeyMod.CtrlCmd | KeyMod.Shift | KeyCode.KEY_V] },
    linux: { primary: KeyMod.CtrlCmd | KeyMod.Shift | KeyCode.KEY_V }
  }, KEYBINDING_CONTEXT_TERMINAL_FOCUS), 'Terminal: Paste into Active Terminal', category,
      KEYBINDING_CONTEXT_TERMINAL_PROCESS_SUPPORTED);
<<<<<<< example_afn_csdiff_typescript/left.ts
  // An extra Windows-only ctrl+v keybinding is used for pwsh that sends ctrl+v directly to the
  // shell, this gets handled by PSReadLine which properly handles multi-line pastes. This is
  // disabled in accessibility mode as PowerShell does not run PSReadLine when it detects a screen
  // reader.
  if (platform.isWindows) {
    registerSendSequenceKeybinding(String.fromCharCode('V'.charCodeAt(0) - CTRL_LETTER_OFFSET), { // ctrl+v
      when: ContextKeyExpr.and(KEYBINDING_CONTEXT_TERMINAL_FOCUS, ContextKeyExpr.equals(
      KEYBINDING_CONTEXT_TERMINAL_SHELL_TYPE_KEY, WindowsShellType.PowerShell), CONTEXT_ACCESSIBILITY_MODE_ENABLED.negate()),
      primary: KeyMod.CtrlCmd | KeyCode.KEY_V
    });
  }

  if (platform.isLinux) {
    actionRegistry.registerWorkbenchAction(SyncActionDescriptor.from(TerminalPasteSelectionAction, {
      linux: { primary: KeyMod.Shift | KeyCode.Insert }
    }, KEYBINDING_CONTEXT_TERMINAL_FOCUS), 'Terminal: Paste selection into Active Terminal', category,
      KEYBINDING_CONTEXT_TERMINAL_PROCESS_SUPPORTED);
  }
=======
}

// An extra Windows-only ctrl+v keybinding is used for pwsh that sends ctrl+v directly to the
// shell, this gets handled by PSReadLine which properly handles multi-line pastes. This is
// disabled in accessibility mode as PowerShell does not run PSReadLine when it detects a screen
// reader. This works even when clipboard.readText is not supported.
if (platform.isWindows) {
  registerSendSequenceKeybinding(String.fromCharCode('V'.charCodeAt(0) - CTRL_LETTER_OFFSET), { // ctrl+v
    when: ContextKeyExpr.and(KEYBINDING_CONTEXT_TERMINAL_FOCUS, ContextKeyExpr.equals(
      KEYBINDING_CONTEXT_TERMINAL_SHELL_TYPE_KEY, WindowsShellType.PowerShell), CONTEXT_ACCESSIBILITY_MODE_ENABLED.negate()),
    primary: KeyMod.CtrlCmd | KeyCode.KEY_V
  });
>>>>>>> example_afn_csdiff_typescript/right.ts
}
```

Figura 4.6 Exemplo de falso negativo adicionado pelo CSDiff. Ao executar o diff3 nos arquivos *base.ts*(Figura 4.3), *left.ts*(Figura 4.4) e *right.ts*(Figura 4.5), este é o resultado obtido, com o diff3 reportando conflito para a integração.

de arquivos). Isso é um ponto positivo para a ferramenta CSDiff, mostrando que sua capacidade de resolver conflitos quando ocorrem em uma mesma região do código, com a ajuda de separadores, reduz a quantidade de conflitos que seriam inevitavelmente reportados pelo diff3.

Os arquivos/cenários para qual o CSDiff apresentou a adição de falsos positivos ocorreram devido a grandes reorganizações feitas no código com movimentações de blocos de código para diferentes seções. Devido ao CSDiff quebrar o código em áreas que são analisados de forma mais granular, ocorrem problemas nas comparações entre linhas que podem estar em regiões completamente diferentes do código. Isso pode ser observado nos repositórios *brew*[2] e *rails*[3].

Para o caso do experimento realizado nos repositórios **TypeScript**, percebemos uma redução de 91.83% de falsos positivos adicionados para cenários, e de 94.54% para arquivos. Novamente vemos uma amostra da capacidade do CSDiff em reduzir a quantidade de esforço

[2]Commit `918f1b775b4705cf042a004bd0df0fb23ca7a7ad`
[3]Commit `91cd8ccdb5353427605aba61c3108363179426a8`

```
// ------------------------------------------------------------
// csdiff.ts
// ------------------------------------------------------------
// The text representation of `^<letter>` is `'A'.charCodeAt(0) + 1`.
const CTRL_LETTER_OFFSET = 64;

if (BrowserFeatures.clipboard.readText) {
  actionRegistry.registerWorkbenchAction(SyncActionDescriptor.from(TerminalPasteAction, platform.isMacintosh && platform.
      isWeb ? undefined : {
    primary: KeyMod.CtrlCmd | KeyCode.KEY_V,
    win: { primary: KeyMod.CtrlCmd | KeyCode.KEY_V, secondary: [KeyMod.CtrlCmd | KeyMod.Shift | KeyCode.KEY_V] },
    linux: { primary: KeyMod.CtrlCmd | KeyMod.Shift | KeyCode.KEY_V }
  }, KEYBINDING_CONTEXT_TERMINAL_FOCUS), 'Terminal: Paste into Active Terminal', category,
      KEYBINDING_CONTEXT_TERMINAL_PROCESS_SUPPORTED);
}

// An extra Windows-only ctrl+v keybinding is used for pwsh that sends ctrl+v directly to the
// shell, this gets handled by PSReadLine which properly handles multi-line pastes. This is
// disabled in accessibility mode as PowerShell does not run PSReadLine when it detects a screen
// reader. This works even when clipboard.readText is not supported.
if (platform.isWindows) {
  registerSendSequenceKeybinding(String.fromCharCode('V'.charCodeAt(0) - CTRL_LETTER_OFFSET), { // ctrl+v
    when: ContextKeyExpr.and(KEYBINDING_CONTEXT_TERMINAL_FOCUS, ContextKeyExpr.equals(
      KEYBINDING_CONTEXT_TERMINAL_SHELL_TYPE_KEY, WindowsShellType.PowerShell), CONTEXT_ACCESSIBILITY_MODE_ENABLED.negate()),
    primary: KeyMod.CtrlCmd | KeyCode.KEY_V
  });

  if (platform.isLinux) {
    actionRegistry.registerWorkbenchAction(SyncActionDescriptor.from(TerminalPasteSelectionAction, {
      linux: { primary: KeyMod.Shift | KeyCode.Insert }
    }, KEYBINDING_CONTEXT_TERMINAL_FOCUS), 'Terminal: Paste selection into Active Terminal', category,
      KEYBINDING_CONTEXT_TERMINAL_PROCESS_SUPPORTED);
  }
}
```

Figura 4.7 Exemplo de falso negativo adicionado pelo CSDiff. Ao executar o CSDiff nos arquivos *base.ts*(Figura 4.3), *left.ts*(Figura 4.4) e *right.ts*(Figura 4.5), este é o resultado obtido. Note que o CSDiff não reporta conflito e gera um código semanticamente incorreto, com o bloco de condicional "`if (platform.isLinux)`" sendo aninhado dentro da condicional "`if (platform.isWindows)`"

```
./gradlew run --args="-e .rb -l 'do begin rescue end { } [ ] ( ) , |' -i
    injectors.CSDiffModule -t 5 -s 01/01/2018 -u 31/10/2021 ./ruby_projects.
    csv ./RubyResultsModifiedDiff3"
```

Figura 4.8 Comando utilizado para executar o experimento de Ruby no *miningframework*.

para realizar o *merge*, visto que 55 arquivos foram marcados com conflitos pelo diff3 enquanto o CSDiff não reportou nenhum, enquanto que apenas 3 arquivos foram marcados com conflitos enquanto o diff3 integrou automaticamente.

Nestes três arquivos, o CSDiff encontrou conflito devido a problemas de alinhamento: devido a grandes adições/remoções de código seja em *left* ou *right*, a estrutura do código foi modificada, fazendo com que a comparação após as quebras de linha comparasse linhas em blocos de código diferentes, "embaralhando" a comparação, causando um falso positivo por parte do CSDiff, ainda que a mudança em si fosse extremamente simples. Isso nos mostra um ponto negativo do CSDiff: a sensibilidade a mudanças estruturais na "forma" do arquivo de texto. Da Figura 4.3 até a a Figura 4.7, observamos o problema do alinhamento que acarreta no falso positivo adicionado para o CSDiff já exposto anteriormente. Os arquivos intermediários da execução do CSDiff que mostram as quebras de linha feitas ao redor dos separadores para os arquivos exemplificados nesses trechos de código são grandes demais para serem inseridos

Repositório	Linguagem	Estrelas	% da Linguagem
rails	Ruby	49383	92,8%
jekyll	Ruby	43608	70,8%
discourse	Ruby	34369	60,3%
brew	Ruby	29698	93,5%
devise	Ruby	22004	97,2%
DefinitelyTyped	TypeScript	36547	100,0%
vscode	TypeScript	123754	93,6%
TypeScript	TypeScript	75739	100,0%
storybook	TypeScript	65855	89,6%
n8n	TypeScript	18691	92,3%

Tabela 4.1 Repositórios analisados neste estudo para medir a acurácia do CSDiff. Dados obtidos em 08/11/2021 às 12:44. Fonte: api.github.com

```
./gradlew run --args="-e .ts -l '( ) [ ] , ; { } < > | : ??' -i injectors.
    CSDiffModule -t 5 -s 01/01/2020 -u 31/10/2021 ./typescript_projects.csv
    ./TypeScriptResultsModifiedDiff3"
```

Figura 4.9 Comando utilizado para executar o experimento de TypeScript no *miningframework*

neste trabalho. O leitor pode explorar mais sobre os detalhes do problema de alinhamento vendo os arquivos na íntegra no repositório[4] do estudo no GitHub.

Em linhas gerais, as duas primeiras métricas (referentes à redução de falsos positivos adicionados) seguem na mesma direção, mas com uma redução mais intensa para TypeScript — 94.54% menos em arquivos e 91.83% menos em cenários para TypeScript contra 80% menos cenários e arquivos para Ruby. Ainda assim, ambos são resultados similares aos obtidos na pesquisa anterior feita para a linguagem Java, que apresentou uma redução de 90% para aFPs em cenários e 90.47% para aFPs em arquivos, no caso do conjunto maior de separadores.

4.4.4 PP4 - A nova solução de merge não estruturado, utilizando separadores, amplia a possibilidade de comprometer a corretude do código, por aumentar o número de integrações de mudanças que interferem uma na outra, sem reportar conflitos (falsos negativos), além de aumentar cenários com falsos negativos?

Para responder a essa pergunta, analisamos os possíveis falsos negativos para os repositórios de Ruby e TypeScript, para ambas as ferramentas, diff3 e CSDiff. Como explicado anteriormente, um possível falso negativo para uma determinada ferramenta ocorre quando esta não reporta conflito e erra o resultado final do *merge* quando comparado com o arquivo do repositório, enquanto a outra ferramenta reporta conflito. Após essa filtragem inicial, é necessário uma

[4]https://github.com/heitorado/tg-heitor-2021/tree/main/csdiff_afn_example

#	diff3	CSDiff
Conflitos	83	75
Arquivos com Conflitos	65	53
Cenários com Conflitos	49	41
Arquivos com conflito apenas para a ferramenta	10	2
Cenários com conflito apenas para a ferramenta	10	2

Tabela 4.2 Quantidade de conflitos encontrados após execução do experimento nos repositórios da linguagem Ruby - os separadores utilizados pelo CSDiff foram: `"do begin rescue end { } [ ] ( ) , |"`

#	diff3	CSDiff
Conflitos	1022	1451
Arquivos com conflitos	589	472
Cenários com conflitos	341	294
Arquivos com conflito apenas para a ferramenta	61	4
Cenários com conflito apenas para a ferramenta	51	4

Tabela 4.3 Quantidade de conflitos encontrados após execução do experimento nos repositórios da linguagem TypeScript - os separadores utilizados pelo CSDiff foram: `"( ) [ ] , ; { } < > | : ??"`

análise manual para confirmar se os casos são realmente falsos negativos - isso porque apenas errar o resultado do *merge* não significa que existe um falso negativo, visto que o desenvolvedor pode ter feito mudanças no arquivo final. Caso o falso negativo não seja confirmado (isto é, a ferramenta teria acertado o resultado se não fossem feitas modificações manuais), ele é contabilizado como um falso positivo da outra ferramenta (da que reportou conflito).

Para o estudo de **Ruby**, foram encontrados três possíveis falsos negativos, porém após análise manual identificamos que o CSDiff apenas errou devido a modificações manuais feitas no arquivo final. Estes três casos foram contabilizados como falsos positivos do diff3.

Em relação ao diff3, não foi encontrado nenhum possível falso negativo.

Para o estudo de **TypeScript** houve apenas dois casos de falsos negativos, ambos no repositório 'vscode'[5][6]. Em ambos, o CSDiff deixou de reportar erro, isto é, apresentou um falso negativo, devido ao problema de alinhamento mencionado anteriormente para os falsos positivos. Isso reforça o problema da sensibilidade do CSDiff a mudanças na "forma" geral do arquivo de código, mostrando que isto pode causar também o surgimento de falsos negativos. Todos os outros possíveis falsos negativos analisados se tratavam de cenários onde uma modificação manual foi feita no arquivo de *merge* final no repositório, e que o CSDiff teria realizado o *merge* corretamente se não houvesse essa modificação. Esses casos foram adicionados na contagem de aFPs do diff3.

[5]Commit: `db2809fe44335e484a502a5a42b8e4d59a5772bf`
[6]Commit: `631f1bdb8fdfe4491610e54bca68afeb85fb729c`

#	diff3	CSDiff
Cenários com aFP's	10	2
Arquivos com aFP's	10	2
Cenários com aFN's	0	0
Arquivos com aFN's	0	0

Tabela 4.4 Quantidade de falsos positivos e falsos negativos encontrados após análise do experimento realizado nos repositórios da linguagem Ruby (para confirmar os falsos negativos, foi necessária análise manual) - os separadores utilizados pelo CSDiff foram: "`do begin rescue end { } [ ] ( ) , |`"

#	diff3	CSDiff
Cenários com aFP's	49	4
Arquivos com aFP's	55	3
Cenários com aFN's	0	2
Arquivos com aFN's	0	2

Tabela 4.5 Quantidade de falsos positivos e falsos negativos encontrados após análise do experimento realizado nos repositórios da linguagem TypeScript (para confirmar os falsos negativos, foi necessária análise manual) - os separadores utilizados pelo CSDiff foram: "`( ) [ ] , ; { } < > | : ??`"

Houve apenas um possível falso negativo para o diff3 no repositório n8n[7], que, após análise manual, verificamos que seria necessária uma modificação manual em ambos os casos, visto que ambos (diff3 e CSDiff) cometeram erros no *merge* do código - entretanto, o CSDiff adicionou mais conflitos no arquivo, e devido a isso foi contabilizado como um falso positivo adicionado do CSDiff.

4.5 Discussão

Com as perguntas de pesquisa respondidas na seção anterior, conseguimos observar mais detalhes sobre o comportamento do CSDiff. Foi possível verificar a capacidade do CSDiff em integrar código para mais linguagens além da que foi projetado originalmente, feitas as melhorias e alterações apresentadas no Capítulo 3.

Entretanto, podemos elicitar alguns problemas que ainda permanecem como pontos de melhoria para a ferramenta. Um deles, seria a redução de conflitos em cenários de *merge* resolvidos corretamente (a ferramenta produz um resultado igual ao que se encontra no repositório do projeto) — 0.46% para Ruby e 0.05% para TypeScript. Ainda que a redução de falsos positivos seja otimista (94.54% menos em arquivos e 91.83% menos em cenários para TypeScript como mostrado na Tabela 4.5, e 80% menos cenários e arquivos para Ruby, como mostrado na Ta-

[7]Commit: `e24fa909cbb2f36435f53fd67a173ec8c04eb6fc`

bela 4.4), vemos que há espaço para melhora. Os valores para comparação de cenários onde as ferramentas produzem resultado igual ao do repositório, quantidade de arquivos com conflitos, quantidade de cenários com conflitos e total de conflitos para cada ferramenta estão detalhados na Tabela 4.6 para Ruby e na Tabela 4.7 para TypeScript.

#	diff3	CSDiff	Variação
Cenários com resultado igual ao repositório	9039	9044	+0.055%
Arquivos com conflitos	65	53	-18.46%
Cenários com conflitos	49	41	-16.32%
Total de conflitos	83	75	-9.63%

Tabela 4.6 Acurácia da Ferramenta CSDiff em comparação ao diff3 - relação percentual do número de cenários resolvidos e conflitos encontrados para a linguagem Ruby.

#	diff3	CSDiff	Variação
Cenários com resultado igual ao repositório	5135	5159	+0.46%
Arquivos com conflitos	589	472	-19.86%
Cenários com conflitos	341	294	-13.78%
Total de conflitos	1022	1451	+41.97%

Tabela 4.7 Acurácia da Ferramenta CSDiff em comparação ao diff3 - relação percentual do número de cenários resolvidos e conflitos encontrados para a linguagem TypeScript.

A quantidade de conflitos detectados pelo CSDiff permaneceram maiores em comparação ao diff3 para a linguagem TypeScript, porém, os resultados obtidos em Ruby mostraram um comportamento divergente. As causas para isso, como explicadas na seção anterior, ocorreram devido a falsos positivos do diff3. Como Clementino propõe em seu estudo original, poderia-se resolver o problema da quantidade de conflitos executando o diff3 caso o CSDiff apontasse conflitos para aquele arquivo [CBC21].

Adicionalmente, uma abordagem sugerida para trabalhos futuros, porém não testada, seria executar o diff3 também antes do CSDiff, de forma a resolver conflitos onde a ferramenta obtém bons resultados, isto é, em alterações de grandes blocos de código, que, para o CSDiff, pode causar falsos positivos ou falsos negativos, devido ao problema de alinhamento, também já discutido. Nessa abordagem, caso o diff3 aponte conflitos no arquivo, poderíamos "resolvê-los" por não escolher nem *left* nem *right*, mas sim *base*. O arquivo resultante então será usado como um novo arquivo *base*, que deverá ser usado pelo CSDiff para tentar resolver os conflitos onde o diff3 não conseguiu anteriormente. A ideia seria utilizar o CSDiff como um auxiliar ao diff3, resolvendo conflitos remanescentes da primeira execução do diff3.

Uma outra sugestão para futuras pesquisas nesta ferramenta, com o intuito de resolver o problema de alinhamento, seria de adicionar marcadores dinâmicos, isto é, ao invés de apenas `$$$$$$$`, construir os separadores incrementalmente: `[$$$1$$$, $$$2$$$, $$$3$$$, ...]`. Eles podem depois ser removidos facilmente com uma expressão regular simples como

`"/\$\$\$[0-9]+\$\$\$/"`, por exemplo. Desta forma, estamos virtualmente adicionando "rótulos" aos trechos de código que são isolados entre os marcadores sintáticos de cada linguagem, fazendo com que a comparação feita pelo diff3 dentro do algoritmo CSDiff ocorra de forma mais controlada, respeitando as diferentes seções de código, agora enumeradas.

Complementarmente, sugere-se uma investigação mais profunda das razões pela qual o CSDiff reportou menos conflitos para Ruby e inclusive, porque as métricas obtidas para a linguagem têm uma magnitude tão diferente das métricas obtidas para TypeScript. Isso possibilita estudos que possam investigar, por exemplo, se isso ocorre devido a algum fator comunidade de desenvolvimento Ruby, ou se ocorre devido a um fator presente na estrutura da linguagem em si. Desenvolver melhor tais questões pode possibilitar o mapeamento de linguagens para qual o CSDiff performa melhor e que poderia ser utilizado como ferramenta dedicada para reduzir a quantidade de conflitos.

4.6 Ameaças à Validade

Nesta seção discutimos algumas ameaças à validade identificadas no decorrer da elaboração deste trabalho.

Por termos seguido a mesma metodologia, novamente apontamos como ameaça o fato das análises de aFNs terem sido realizadas manualmente, o que abre margem para erros humanos [CBC21].

Os separadores utilizados para obter os dados dos estudos feitos nas linguagens Ruby e TypeScript foram arbitrariamente escolhidos pelo autor, de acordo com a familiaridade que possuía com as linguagens. O autor fez uma listagem simples dos separadores que estava mais acostumado a encontrar nos códigos de Ruby e TypeScript com que teve contato. Como os conjuntos de separadores não foram escolhidos seguindo alguma regra ou parâmetro claro, não se sabe que variações nos resultados poderia-se encontrar por usar um conjunto diferente (seja ele maior, menor ou igual em tamanho) de separadores.

Ademais, os experimentos foram executados com poucos repositórios para cada linguagem (cinco para cada) e isto pode impactar na obtenção de dados suficientes para o resultado final. Segundo os estudos de Brun et al. [BHEN11], 16% de todos os *merges* encontrados no estudo realizado requeriam esforço humano para serem resolvidos, isto é, o conflito não pôde ser resolvido automaticamente pelo sistema de controle de versão. Como o que nos interessa para avaliar a acurácia de ferramentas de *merge* automatizado são justamente esses 16% que resultam em conflitos que requerem intervenção humana, um volume maior de repositórios, ou ainda, um espaço de tempo mais amplo na busca por commits de *merge* pode ajudar a obter um volume maior de dados e, consequentemente, resultados mais significativos, além de contribuir para identificar pontos fortes ou de melhoria da ferramenta.

Outro ponto a se considerar é a variação na quantidade de separadores utilizados para cada experimento. Diferentemente do estudo original [CBC21], não foram analisados diferentes conjuntos de separadores, devido ao tempo disponível para realização da pesquisa e análise dos dados. Este tipo de variação na análise contribuiria para obter novos dados sobre o comportamento do CSDiff, e se existe uma melhora ou piora em sua acurácia a depender dos separadores utilizados para cada linguagem. Dito isso, encontrar o conjunto ótimo de separadores para uma

dada linguagem pode ser objeto de estudo de trabalhos futuros.

```
// ------------------------------------------------------------
// diff3.ts
// ------------------------------------------------------------
private getOrCreateResourceGroupMenusItem(group: ISCMResourceGroup): SCMMenusItem {
  let result = this.resourceGroupMenusItems.get(group);

  if (!result) {
    const contextKeyService = this.contextKeyService.createScoped();
      contextKeyService.createKey('scmProvider', group.provider.contextValue);
    contextKeyService.createKey('scmResourceGroup', getSCMResourceContextKey(group));

<<<<<<< /vscode/c466365768cdbe46592c75b8d7d324cc1756ce44/src/vs/workbench/contrib/scm/browser/menus.ts/left.ts
    const resourceGroupMenu = this.menuService.createMenu(MenuId.SCMResourceGroupContext, contextKeyService);
    const resourceFolderMenu = this.menuService.createMenu(MenuId.SCMResourceFolderContext, contextKeyService);
=======
    result = new SCMMenusItem(contextKeyService, this.menuService);
    this.resourceGroupMenusItems.set(group, result);
  }
>>>>>>> /vscode/c466365768cdbe46592c75b8d7d324cc1756ce44/src/vs/workbench/contrib/scm/browser/menus.ts/right.ts

<<<<<<< /vscode/c466365768cdbe46592c75b8d7d324cc1756ce44/src/vs/workbench/contrib/scm/browser/menus.ts/left.ts
    const resourceMenusByContext = new Map<string, ISCMResourceMenu>();
    const resourceMenusDisposable = { dispose: () => resourceMenusByContext.forEach(menu => menu.dispose()) };

    const disposable = combinedDisposable(contextKeyService, resourceGroupMenu, resourceFolderMenu, resourceMenusDisposable);

    this.resourceGroupMenus.set(group, { resourceGroupMenu, resourceFolderMenu, resourceMenusByContext });
    return { group, disposable };
  });
=======
  return result;
}
>>>>>>> /vscode/c466365768cdbe46592c75b8d7d324cc1756ce44/src/vs/workbench/contrib/scm/browser/menus.ts/right.ts
```

Figura 4.10 Exemplo do aumento de conflitos pelo CSDiff. Podemos observar que um mesmo conflito no diff3 corresponde a múltiplos conflitos no CSDiff, pois a divisão do código em seções delimitadas pelos separadores sintáticos aumenta a granularidade dos conflitos reportados pelo CSDiff. Neste arquivo do diff3, notamos dois conflitos reportados contra três do CSDiff (Figura 4.11)

```
// ------------------------------------------------------------
// csdiff.ts
// ------------------------------------------------------------
private getOrCreateResourceGroupMenusItem(group: ISCMResourceGroup): SCMMenusItem {
  let result = this.resourceGroupMenusItems.get(group);

  if (!result) {
    const contextKeyService = this.contextKeyService.createScoped();
    contextKeyService.createKey('scmProvider', group.provider.contextValue);
    contextKeyService.createKey('scmResourceGroup', getSCMResourceContextKey(group));

    result = new SCMMenusItem(contextKeyService, this.menuService);
<<<<<<< /vscode/c466365768cdbe46592c75b8d7d324cc1756ce44/src/vs/workbench/contrib/scm/browser/menus.ts/left.ts
    const resourceFolderMenu = this.menuService.createMenu(MenuId.SCMResourceFolderContext
=======
    this.resourceGroupMenusItems.set(group
>>>>>>> /vscode/c466365768cdbe46592c75b8d7d324cc1756ce44/src/vs/workbench/contrib/scm/browser/menus.ts/right.ts
, result);
<<<<<<< /vscode/c466365768cdbe46592c75b8d7d324cc1756ce44/src/vs/workbench/contrib/scm/browser/menus.ts/left.ts

    const resourceMenusByContext = new Map<string, ISCMResourceMenu>();
    const resourceMenusDisposable = { dispose: () => resourceMenusByContext.forEach(menu => menu.dispose()) }
=======
  }
>>>>>>> /vscode/c466365768cdbe46592c75b8d7d324cc1756ce44/src/vs/workbench/contrib/scm/browser/menus.ts/right.ts

  return result;
<<<<<<< /vscode/c466365768cdbe46592c75b8d7d324cc1756ce44/src/vs/workbench/contrib/scm/browser/menus.ts/left.ts

  const disposable = combinedDisposable(contextKeyService, resourceGroupMenu, resourceFolderMenu, resourceMenusDisposable);
=======
}
>>>>>>> /vscode/c466365768cdbe46592c75b8d7d324cc1756ce44/src/vs/workbench/contrib/scm/browser/menus.ts/right.ts
```

Figura 4.11 Exemplo do aumento de conflitos pelo CSDiff. Podemos observar que um mesmo conflito no diff3 corresponde a múltiplos conflitos no CSDiff, pois a divisão do código em seções delimitadas pelos separadores sintáticos aumenta a granularidade dos conflitos reportados pelo CSDiff. Neste arquivo do CSDiff, notamos três conflitos reportados contra dois do diff3 (Figura 4.10)

CAPÍTULO 5

Trabalhos Relacionados

Diversos outros trabalhos existem na academia dedicados a investigar e aprofundar os conhecimentos em diferentes tipos de abordagens e ferramentas *merge*. Entre eles, o trabalho que serviu como base para este estudo, onde Clementino [CBC21], propõe uma nova ferramenta de *merge* textual que visa simular a abordagem estruturada fazendo uso dos separadores sintáticos de uma dada linguagem. Neste estudo, ele analisa a performance da ferramenta proposta com dois diferentes conjuntos de separadores para a linguagem Java. Embora o impacto na resolução de conflitos não tenha sido muito expressivo, foi o esforço inicial que preparou terreno para trabalhos como este.

Em um estudo dedicado ao diff3, Khanna et al. [KKP07] desenvolve uma investigação formal da ferramenta, descrevendo através de formulações e provas teóricas propriedades do diff3 que auxiliam a entender seu funcionamento. Em especial, os autores mostram que ainda que as mudanças feitas nos arquivos que serão integrados pela ferramenta sejam feitas localmente em regiões distintas do código, não há garantia de que o diff3 será capaz de integrar sem conflitos, ainda que, intuitivamente, pareça o contrário.

Cavalcanti et al. [CBA17b] discorre sobre *merge* semiestruturado e seus impactos na redução de conflitos em relação ao merge não estruturado, levantando pontos de análise para avaliar quando uma ferramenta seria preterível em relação a outra: antes de trocarmos nossas ferramentas de *merge*, devemos ter certeza de que isto reduzirá o esforço na integração de código e não trará problemas que comprometam a corretude do mesmo. Desta forma, até mesmo um número baixo de falsos negativos pode ser um grande empecilho para que se utilize o CSDiff em relação ao diff3, se esse número representar uma piora.

Ainda na área de merge semiestruturado, temos o trabalho de Apel et al. [ALB+11], onde propõe e analisa a ferramenta *FSTMerge*. Esta mesma ferramenta foi posteriormente estudada [CAB15, CBA17a] para mensurar a redução de falsos conflitos em comparação a uma abordagem não estruturada, comprovando sua capacidade em reduzi-los em contraste com a abordagem textual (não estruturada).

Cavalcanti et al. [CBSA19] disserta sobre uma comparação feita entre *merge* semiestruturado e estruturado, mostrando que o primeiro reporta mais falsos positivos, enquanto o segundo reporta mais falsos negativos, o que abre portas para investigar o balanceamento entre as diferentes ferramentas de diferentes abordagens, isto é, um "meio termo" entre uma abordagem textual e as abordagens semiestruturadas e estruturadas.

Accioly et al. [ABC18] realiza um estudo para avaliar e catalogar padrões de conflitos em projetos *open source* da linguagem Java, a fim de identificar que tipo de padrões na estrutura do código levam a que conflitos. Ainda que o estudo tenha sido feito para Java, os resultados podem ser aproveitados principalmente para linguagens que usam separadores similares (como

TypeScript), visto que uma de suas descobertas foi que a maioria dos conflitos obtidos após execução da ferramenta de *merge* semiestruturado ocorre quando desenvolvedores modificam a mesma região de um mesmo bloco de código (um método, por exemplo).

CAPÍTULO 6

Conclusão

Conflitos de *merge* são parte do dia a dia em desenvolvimento de software. A depender da ferramenta utilizada para integrar o mesmo arquivo de código modificado simultaneamente por dois ou mais desenvolvedores, podem ocorrer conflitos que tal ferramenta é incapaz de solucionar automaticamente, necessitando de intervenção humana. Em pesquisa de Brun et al. [BHEN11], foi detectado que 16% dos conflitos de *merge* encontrados necessitaram de intervenção humana para serem solucionados.

A solução de conflitos por meios manuais pode consumir tempo e demandar muito esforço por parte dos desenvolvedores [BHEN11, BZ12], levando a um impacto negativo na produtividade.

Visando uma solução que contemple a eficiência do *merge* textual, porém, que analise o código em mais profundidade como o *merge* estruturado, Clementino[CBC21] desenvolveu a ferramenta CSDiff, que realiza o *merge* textual, porém, através do uso de marcadores que dividem as regiões do código que são delimitadas pelos separadores sintáticos da linguagem, simula o *merge* estruturado. O estudo inicial forneceu uma análise da acurácia da ferramenta para a linguagem Java com dois diferentes conjuntos de separadores, comparando-a com outra ferramenta, chamada diff3.

Neste trabalho, demos continuidade ao projeto, resolvendo alguns problemas que identificamos e implementando melhorias sugeridas no primeiro estudo. Adaptamos a infraestrutura de experimentação e repetimos as perguntas de pesquisa e a metodologia, a fim de analisar a acurácia do CSDiff para duas novas linguagens - Ruby e TypeScript.

Encontramos resultados similares para a linguagem TypeScript no que diz respeito à redução de falsos positivos e introdução de falsos negativos em relação ao diff3, porém, observou-se um aumento na resolução automática de conflitos (que necessitariam de análise manual com o uso do diff3) menor em comparação ao estudo original: apenas 0.46%. Resultados similares também foram encontrados para Ruby, com uma pequena divergência na quantidade de conflitos reportados - era esperado que a quantidade aumentasse com o uso do CSDiff, porém o oposto aconteceu. Já aumento da resolução automática de casos para Ruby foi de 0.055%.

Isto não descarta o potencial do CSDiff na resolução de conflitos, e ressalta a necessidade de prosseguir com a implementação de melhorias que tratem seus pontos fracos, bem como uma maior variedade de experimentos, antes que possamos aprová-lo ou descartá-lo como uma ferramenta de *merge* efetiva. Sugerimos melhorias e extensões a este trabalho na Seção 4.5 e na Seção 4.6.

Referências Bibliográficas

[ABC18] Paola Accioly, Paulo Borba, and Guilherme Cavalcanti. Understanding semistructured merge conflict characteristics in open-source java projects (journal-first abstract). In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, page 955, New York, NY, USA, 2018. Association for Computing Machinery.

[ALB+11] Sven Apel, Jörg Liebig, Benjamin Brandl, Christian Lengauer, and Christian Kästner. 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*, ESEC/FSE '11, page 190–200, New York, NY, USA, 2011. Association for Computing Machinery.

[BHEN11] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. Proactive detection of collaboration conflicts. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, page 168–178, New York, NY, USA, 2011. Association for Computing Machinery.

[BZ12] Christian Bird and Thomas Zimmermann. Assessing the value of branches with what-if analysis. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, New York, NY, USA, 2012. Association for Computing Machinery.

[CAB15] Guilherme Cavalcanti, Paola Accioly, and Paulo Borba. Assessing semistructured merge in version control systems: A replicated experiment. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–10, 2015.

[CBA17a] Guilherme Cavalcanti, Paulo Borba, and Paola Accioly. Evaluating and improving semistructured merge. *Proc. ACM Program. Lang.*, 1(OOPSLA), oct 2017.

[CBA17b] Guilherme Cavalcanti, Paulo Borba, and Paola Accioly. Should we replace our merge tools? In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 325–327, 2017.

[CBC21] Jônatas Clementino, Paulo Borba, and Guilherme Cavalcanti. Textual merge based on language-specific syntactic separators. In *Brazilian Symposium on Software*

Engineering, SBES '21, page 243–252, New York, NY, USA, 2021. Association for Computing Machinery.

[CBSA19] Guilherme Cavalcanti, Paulo Borba, Georg Seibt, and Sven Apel. The impact of structure on software merging: Semistructured versus structured merge. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1002–1013, 2019.

[Fou21] Free Software Foundation. diff3, jan 2021. Packaged with Diffutils version 3.8.

[KKP07] Sanjeev Khanna, Keshav Kunal, and Benjamin C. Pierce. A formal investigation of diff3. In V. Arvind and Sanjiva Prasad, editors, *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, pages 485–496, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

Este volume foi tipografado em LaTeX na classe UFPEThesis (www.cin.ufpe.br/~paguso/ufpethesis).