
Universidade Federal de Pernambuco
Centro de Informática
Graduação em Ciência da Computação

# Extensão do *CSDiff* para uso em linguagens com poucos separadores sintáticos

Proposta de Trabalho de Graduação

**Aluno:** José Gabriel Silva Pereira (jgsp2@cin.ufpe.br)

**Orientador:** Paulo Henrique Monteiro Borba (phmb@cin.ufpe.br)

19 de Dezembro de 2022.

# 1. Resumo

A prática de desenvolvimento de software, há muito tempo deixou de ser uma tarefa que era realizada por somente uma pessoa, pois, com o avanço da tecnologia, sistemas cada vez mais complexos foram sendo criados fazendo com que muitas pessoas trabalhassem no mesmo projeto. Por conta disso, ferramentas de controle de versionamento de código foram criadas, permitindo que múltiplos desenvolvedores trabalhassem modificando o mesmo trecho de código simultaneamente. Porém, essas modificações simultâneas podem gerar conflitos quando feitas em um mesmo pedaço de código, o que impacta negativamente na produtividade de um time. Ao decorrer do tempo, diversas formas de como detectar conflitos na junção de versão de códigos foram criadas, dentre elas: linha a linha, estruturada e semiestruturada. Neste trabalho, é proposto uma extensão para uma ferramenta semiestruturada de detecção de conflitos ja existente, o *CSDiff* [1], de forma que ela utilize indentação como separador da linguagem, permitindo assim que, durante a detecção de conflitos em linguagens com poucos separadores sintáticos, ainda seja possivel permitir uma redução de falsos conflitos, consequentemente melhorando a produtividade de um time.

# 2. Introdução

Com o crescimento da tecnologia e desenvolvimento de softwares de larga escala, o número de pessoas trabalhando simultaneamente no mesmo projeto também cresce, fazendo com que ferramentas de controle de versão de código sejam extremamente necessárias. Uma ferramenta no controle de versão de código auxilia na produtividade de uma equipe quando se trata de múltiplos edições paralelas no código. A partir disso, surge a necessidade de que, de tempos em tempos, pedaços de código que foram alterados separadamente sejam unidos, essa união dos códigos é chamada de merge.

No processo de unir partes do projeto utilizando o processo de merge, conflitos podem surgir. Um conflito de merge é definido como duas alterações feitas num mesmo trecho de código, que se afetam mutualmente. Quando conflitos de merge acontecem, é gerado um esforço extra para os desenvolvedores [2], pois necessitam investir tempo estudando melhor as mudanças para resolvê-los. Por causa disso, esses conflitos impactam negativamente na produtividade do time. Além disso, uma má resolução de conflitos pode levar a introdução de bugs dentro do código, o que influencia na qualidade do produto final.

Com o objetivo de auxiliar os desenvolvedores na resolução dos conflitos, várias ferramentas de merge foram propostas, sendo mais amplamente utilizada a solução puramente textual que analisa conflitos linha a linha [3]. Como estas ferramentas utilizam abordagens textuais, elas tendem a gerar conflitos falso positivos, ou seja, conflitos que são produzidos indevidamente. Para contornar isto, formas estruturadas e semiestruturadas de merge, que exploram tanto a parte sintática e semântica do código, foram sugeridas na academia, visando reduzir estes falsos positivos e consequentemente melhorar a experiência dos desenvolvedores. Sintáxe de código é um conjunto de regras que define combinações de símbolos que estruturam uma determinada linguagem de programação. Muitas linguagens de programação utilizam desses símbolos para separar escopos, statements (comandos de códigos bem formados para linguagem), entre outras coisas. Para a linguagem Python, que não possui esse tipo específico de símbolo, utilizando-se de indentação (espaços em branco) como separador sintático, ocorrem casos em que diferentes statements ou escopos apareçam em linhas consecutivas, sem nenhum separador sintático entre elas. Num evento de merge, considerando a abordagem meramente textual linha a linha, se dois desenvolvedores modificam statements diferentes, ou statements que estão em escopos diferentes, porém em linhas consecutivas, o resultado do merge geraria conflito, mesmo que as versões modifiquem partes independentes do código.

A partir da ideia desses separadores, uma nova abordagem semiestruturada de merge pode ser aplicada para linguagens que não possuem esses separadores (a princípio a linguagem Python). Para isso, é necessário que essa nova ferramenta de merge considere também a identação do código como separador. Este trabalho busca implementar essa nova abordagem melhorando uma ferramenta já existente, o *CSDiff* [1], e avaliar se a utilização de indentação, contribui para redução de conflitos que na solução padrão seriam falsos positivos.

# 3. Objetivos

Este trabalho tem como objetivo melhorar uma solução semiestruturada de merge já existente [1], fazendo uso de indentação de código para separar blocos de código que estão em linhas consecutivas, que num evento de merge poderia ser fonte de conflitos, e também avaliar se esta modificação tem efeito na redução de falsos conflitos em comparação a solução já existente.

## 5. Cronograma

| Atividade | Nov | Dez | Jan | Fev | Mar |
|---|---|---|---|---|---|
| Revisão Bibliográfica | X | | | | |
| Implementação da Solução | X | | | | |
| Teste Básico da Solução | X | X | | | |
| Reprodução dos Experimentos | | X | X | | |
| Análise dos Resultados | | | X | | |
| Escrita do Texto | | | X | X | |
| Preparação da Defesa | | | | X | X |

# Referências


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

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

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

# 6. Possíveis Avaliadores

São possíveis avaliadores do trabalho os professores:

- Leopoldo Teixeira (lmt@cin.ufpe.br)
- Guilherme Cavalcanti (gjcc@cin.ufpe.br)

## 7. Assinaturas

Paulo Henrique Monteiro Borba
**Orientador**

José Gabriel Silva Pereira
**Orientando**