Semantic Conflict Analysis in Continuous Software Integration with Semantic Merging Tool

Matheus Barbosa
mbo2@cin.ufpe.br
Centro de Informática
Universidade Federal de Pernambuco
Recife, Pernambuco, Brazil

ABSTRACT

As funcionalidades do sistema de controle de versão possibilitam que os desenvolvedores realizem suas tarefas de desenvolvimento de maneira autônoma. Além disso, essas ferramentas simplificam a incorporação de alterações por meio de operações de merge e identificam conflitos de texto. Conflitos textuais são adversidades significativas, no entanto, existem outras formas de conflitos que podem ser ainda mais prejudiciais e não são identificados pelas ferramentas de merge atualmente disponíveis. Conflitos semânticos dinâmicos referem-se a situações em que não há conflitos textuais aparentes durante o relatórios de merge, porém, resultam em interferências indesejadas que podem ocasionar comportamentos inesperados do programa durante sua execução. Atualmente, há uma variedade de ferramentas disponíveis que adotam abordagens diversas para enfrentar esse desafio. Estas incluem desde análises estáticas até a implementação de testes automatizados. Contudo, é observado que a abordagem centrada em testes geralmente resulta em um maior número de falsos negativos, enquanto as análises estáticas tendem a apresentar uma incidência maior de falsos positivos. Adicionalmente, essas ferramentas estão limitadas a identificar interferências que ocorrem dentro do mesmo método ou em métodos subsequentes, o que pode não refletir a complexidade real dos sistemas. Além disso, foram avaliadas principalmente em cenários de código aberto, sem a inserção direta de intervenção humana em ambientes corporativos. Neste estudo, propomos o aprimoramento nas análises estáticas para possibilitar sua execução com diferentes pontos de entrada, visando identificar interferências em diferentes áreas de um sistema. Além disso, propomos a criação de uma infraestrutura que integra o processo de merge do Git, a ser implementada nas máquinas dos desenvolvedores em um contexto corporativo. Isso viabilizará a coleta de relatórios de merge e a condução de avaliações qualitativas com a participação ativa dos profissionais envolvidos.

KEYWORDS

conflitos de integração de código, ferramenta de merge semântica

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM
https://doi.org/XXXXXXX.XXXXXXX

ACM Reference Format:
Matheus Barbosa. 2024. Semantic Conflict Analysis in Continuous Software Integration with Semantic Merging Tool. In *Companion Proceedings of the 32nd ACM Symposium on the Foundations of Software Engineering (FSE '24), July 15–19, 2024, Porto de Galinhas, Brazil.* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

As ferramentas de merge textual, que operam com base nas linhas de código, identificam conflitos ao mesclar duas versões que aplicam alterações às mesmas linhas ou a linhas consecutivas [1]. Embora esses *conflitos textuais* ou conflitos de *merge* sejam comuns e extensivamente estudados, não representam o único problema ao integrar o código de dois desenvolvedores.

Devido à sua limitação em considerar apenas a combinação de linhas, as ferramentas de merge textual não conseguem detectar alterações incompatíveis que ocorrem em áreas do código separadas por pelo menos uma única linha. Por exemplo, se um desenvolvedor altera a assinatura de um método e outro adiciona uma chamada ao método com a assinatura original, mas em linhas diferentes, as ferramentas de merge textual integram o código sem conflitos ou alertas. No entanto, o código resultante pode não compilar, o que configura um *conflito semântico estático* [6, 15, 17, 19, 20, 22]. Também chamado de conflito de *Build* .

Além disso, essas ferramentas não conseguem identificar *conflitos semânticos dinâmicos* [3, 6, 7, 9, 13, 16, 18, 21, 22], que ocorrem quando as alterações de um desenvolvedor afetam um elemento de estado acessado pelo código alterado por outro desenvolvedor, resultando em um comportamento inesperado durante a execução do programa. Visto que não gera conflito de *merge*, pode aumentar as chances de ocorrência de outros conflitos [11]. Conflitos de *teste* e de *produção* (e não detectados) correspondem a conflitos semânticos dinâmicos.

Conflitos semânticos têm o potencial de prejudicar tanto a eficiência do desenvolvimento quanto a qualidade dos produtos de software. Este impacto é particularmente notável em projetos com bifurcações divergentes, conforme discutido em estudos anteriores [5, 6, 12, 19, 22, 23]. Contudo, mesmo em projetos que possuem apenas um repositório remoto, esses conflitos podem representar desafios significativos.

Diversas abordagens têm sido exploradas para lidar com a detecção de conflitos semânticos. Por exemplo, a geração de testes [17] tem sido utilizada, mas enfrenta o desafio de altas taxas de falsos negativos, uma vez que verifica a interferência com base em valores de entrada específicos durante a execução do código testado.

Por outro lado, algoritmos de análise estática foram propostos para identificar conflitos semânticos [3, 4, 9], utilizando grafos complexos, como os System Dependence Graphs. Entretanto, à medida que o tamanho das bases de código aumenta, essas abordagens sofrem uma degradação significativa de desempenho, tornando-as impraticáveis para aplicações em bases de código reais com mais de 50 mil linhas de código. Diante desse cenário, surge a necessidade de explorar simplificações nos algoritmos existentes, visando a detecção de conflitos semânticos com menor custo computacional.

Recentemente, em nossa pesquisa, introduzimos a utilização de análises estáticas simplificadas [14], focadas exclusivamente na versão mesclada do código. Nossa abordagem envolve a marcação dessa versão com metadados que indicam quais instruções foram modificadas ou adicionadas por cada desenvolvedor. Contudo, esta abordagem enfrenta dois desafios significativos. Em primeiro lugar, essas ferramentas estão restritas a identificar interferências que se manifestam dentro do mesmo método ou em métodos subsequentes, o que pode não representar adequadamente a complexidade inerente aos sistemas reais. Em segundo lugar, sua avaliação foi predominantemente conduzida em ambientes de código aberto, onde a intervenção humana direta em contextos corporativos não foi considerada.

Nesse contexto, este estudo propõe aprimoramentos nas análises estáticas, visando à sua adaptação para operar com diversos pontos de entrada, a fim de identificar interferências em diferentes áreas de um sistema. Além disso, o artigo apresenta uma investigação sobre o uso de análises estáticas simplificadas para detectar interferências, com a participação de desenvolvedores reais em seus ambientes de trabalho.

2 BACKGROUND AND RELATED WORK

Nesta seção, apresentamos uma revisão dos estudos anteriores que serviram como base de evidência para nossa pesquisa, bem como trabalhos relacionados no campo.

Para identificar efetivamente conflitos semânticos, Da Silva *et al* [17] introduziram uma metodologia fundamentada na geração automatizada de testes unitários, os quais servem como especificações parciais para a detecção de interferências em situações de integração. Os autores empregam critérios de aprovação de teste que sugerem interferência: se um teste gerado falhar com a versão base, passar em um dos desenvolvedores e, em seguida, falhar novamente na versão de merge, isso indica que a mudança realizada por um dos desenvolvedores não está preservada na versão mergeada. Em contraste, nossa abordagem se baseia em análise estática. Além disso, esperamos que nosso desempenho seja superior ao deles, embora as informações fornecidas sejam limitadas. A combinação dessas duas técnicas pode ser uma área promissora para futuras investigações.

No campo das análises estáticas, Horwitz et al. foram pioneiros no estudo de conflitos semânticos dinâmicos [9, 10, 21] e formalizaram a definição de interferência, um conceito central adotado neste estudo. Sua abordagem envolve a construção de Program Dependence Graphs (PDGs)[9] e System Dependence Graphs (SDGs)[10] para as quatro versões do programa em um cenário de merge, analisando as diferenças entre eles para detectar e resolver interferências. Em um trabalho mais recente, Barros Filho [3] usando o framework JOANA[8] propôs a utilização de análises com o objetivo de investigar se o Information Flow Control (IFC) pode ser empregado para identificar a presença de conflitos semânticos dinâmicos entre as contribuições dos desenvolvedores em cenários de merge. No entanto, essas abordagens pode demandar horas para serem construídas, enquanto nossas análises são significativamente mais baratas.

Propomos uma técnica leve que explora o uso de análise estática para detectar interferências ao mesclar contribuições de dois desenvolvedores [14]. Entretanto, essas análises encontram-se restritas a identificar interferências ocorridas apenas dentro do mesmo método ou em métodos subsequentes, o que possivelmente não representa adequadamente a complexidade intrínseca dos sistemas. No âmbito deste estudo, nossa intenção é aprimorar essas análises, permitindo que operem com diferentes pontos de entrada, permitindo assim a detecção de interferencias em pontos distintos do programa analisado. Além disso, esses trabalhos foram avaliados principalmente em cenários de código aberto, sem a inserção direta de intervenção humana em ambientes corporativos. Nossa proposta busca realizar uma avaliação qualitativa com desenvolvedores em seus ambientes de trabalho reais, visando aprimorar nosso entendimento sobre a usabilidade de ferramentas desse tipo nesse contexto específico.

Sousa et al. [18] exploram uma técnica alternativa para detectar interferências. Eles inferem pós-condições relacionais de forma estática a partir das versões do código envolvidas no cenário de merge, estabelecendo restrições sobre as modificações dos elementos de estado por diferentes versões para prevenir interferências. Em seguida, utilizam técnicas de prova de teoremas, como a resolução SMT, para verificar a satisfação dessas restrições. Assim como alguns trabalhos supracitdos, este usa projetos do github para validar a sua aboradgem.

3 COMPLETED WORK

3.1 Semantic conflict detection with overriding assignment analysis

Propomos e implementamos uma análise de sobreposição de atribuições, que visa detectar interferência entre alterações introduzidas por dois desenvolvedores diferentes, onde caminhos de escrita, sem atribuições intermediárias, para um alvo comum indicam interferência. Para avaliar as implementações da análise proposta, foram comparados os resultados da análise para um conjunto de 78 cenários de integração de código com o Ground Truth para Overriding Assignment (OA) e Local Observable Interference (LOI). Os resultados mostraram que a análise proposta conseguiu detectar interferências entre as modificações, indicando sua eficácia como ferramenta para detecção de conflitos semânticos. No entanto, houve uma quantidade considerável de falsos negativos, sugerindo que a análise por si só não é suficiente para uma detecção confiável de interferências. A precisão da análise foi de 0,44 para a abordagem intraprocedural e 1,0 para a abordagem interprocedural, com uma acurácia de 0,7 e 0,88, respectivamente. Os resultados também destacaram a necessidade de combinar a análise proposta com outras técnicas para criar uma ferramenta mais robusta de detecção de conflitos de integração semânticos. Mais detalhes deste trabalho podem ser encontrados em nosso artigo [2]

3.2 Lightweight Semantic Conflict Detection with Static Analysis

Neste trabalho, nos concentramos em conflitos semânticos dinâmicos, que ocorrem quando os relatórios de merge não apresentam conflitos textuais, mas resulta em interferências indesejadas, causando comportamento inesperado do programa em tempo de execução. Para resolver esse problema, propomos uma técnica leve que explora o uso de quatro análises estáticas (Interprocedural Data Flow, Interprocedural Confluence, Interprocedural Override Assignment, and Program Dependence Graph, which includes the control dependency that detects the interference) para detectar interferências ao mesclar contribuições de dois desenvolvedores. Avaliamos nossa técnica usando um conjunto de 99 unidades experimentais extraídas de cenários de merge de projetos do GitHub e escolhidos principalmente de trabalhos anteriores relacionados a conflitos semânticos. Os resultados fornecem evidências de que nossa técnica apresenta capacidade significativa de detecção de interferências. Ele supera, em termos de pontuação F1 e recall, técnicas anteriores que dependem de análises baseadas em testes para detectar conflitos semânticos, mas apresentam melhor precisão. A precisão de nossa técnica é comparável às observadas em outros estudos que também utilizam análise estática ou utilizam técnicas de prova de teoremas para detectar conflitos semânticos, embora com desempenho de execução significativamente melhorado. Esses resultados evidenciam a capacidade da análise estática em detectar interferências de forma significativa durante o processo de merge de código. Mais detalhes deste trabalho podem ser encontrados em nosso artigo [14]

4 CONTRIBUTION TO KNOWLEDGE

Apesar dos avanços nas ferramentas de merge, ainda existem desafios significativos na detecção de conflitos semânticos dinâmicos durante o processo de integração de software. Diversas ferramentas buscam abordagens variadas para resolver esse problema, porém enfrentam desafios de desempenho ou limitações na detecção de interferências que ocorrem apenas dentro do mesmo método ou em métodos subsequentes. Além disso, é comum que essas ferramentas nunca tenham sido testadas quanto à sua usabilidade por seres humanos.

Em nosso trabalho, propomos a criação de uma infraestrutura que integra o processo de merge do Git, a ser implementada nas máquinas dos desenvolvedores em um contexto corporativo visando identificar interferências entre as alterações introduzidas por dois desenvolvedores diferentes e realizar um estudo qualitativo buscando aprimorar nosso entendimento sobre a usabilidade de ferramentas desse tipo, nesse contexto específico.

O trabalho de doutorado encontra-se em uma fase inicial, completando recentemente dois anos de pesquisa dedicada. Como destacado em estudos anteriores, já conseguimos validar a eficácia da análise estática em identificar conflitos semânticos em ambientes de desenvolvimento, utilizando repositórios relevantes hospedados no GitHub. Agora, almejamos avançar nesse campo, buscando aplicar nossa metodologia em ambientes reais de trabalho, com desenvolvedores ativos em empresas de diversos setores.

Nossa proposta consiste em integrar nossa ferramenta diretamente nos fluxos de trabalho desses profissionais, permitindo-nos capturar dados detalhados durante a execução. Ao coletar registros dessas interações, teremos acesso a uma visão mais precisa e abrangente do uso da ferramenta em cenários práticos. Essa abordagem nos possibilitará não apenas validar a eficácia da análise estática em um contexto mais dinâmico e complexo, mas também entender melhor as necessidades e desafios enfrentados pelos desenvolvedores no dia a dia.

A partir desses dados, poderemos gerar relatórios mais informativos e relevantes, oferecendo insights valiosos tanto para os desenvolvedores quanto para as equipes de pesquisa e desenvolvimento. Esperamos que essa colaboração direta com profissionais da área não apenas fortaleça a validade de nossa abordagem, mas também contribua para o avanço do conhecimento e práticas na detecção e resolução de conflitos semânticos em projetos de software.

ACKNOWLEDGMENTS

O autor recebeu orientação do Dr. Paulo Borda, Membro da ACM e da Sociedade Brasileira de Computação, professor de Engenharia de Software no Centro de Informática da Universidade Federal de Pernambuco e líder do Grupo de Produtividade de Software. Atualmente também é diretor do Centro de Informática.

Este trabalho conta com apoio parcial das bolsas IBPG-0567-1.03/22 da Fundação de Amparo à Ciência e Tecnologia do Estado de Pernambuco. Quaisquer opiniões, descobertas e conclusões expressas neste trabalho são de responsabilidade exclusiva do autor e não refletem necessariamente as dos patrocinadores.

We would like to thank INES (National Software Engineering Institute) and CNPQ.

REFERENCES

[1] Paola Accioly, Paulo Borba, and Guilherme Cavalcanti. 2018. Understanding semi-structured merge conflict characteristics in open-source java projects. *Empirical Software Engineering* 23, 4 (2018), 2051–2085.

[2] Matheus Barbosa, Paulo Borba, Rodrigo Bonifacio, and Galileu Santos. 2022. Semantic conflict detection with overriding assignment analysis. In *Proceedings of the XXXVI Brazilian Symposium on Software Engineering* (<conf-loc>, <city>Virtual Event</city>, <country>Brazil</country>, </conf-loc>) *(SBES '22)*. Association for Computing Machinery, New York, NY, USA, 435–445. https://doi.org/10.1145/3555228.3555242

[3] Roberto Souto Maior de Barros Filho. 2017. *Using information flow to estimate interference between same-method contributions.* Master's thesis. Federal University of Pernambuco.

[4] David Binkley, Susan Horwitz, and Thomas Reps. 1995. Program integration for languages with procedure calls. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 4, 1 (1995), 3–35. https://doi.org/10.1145/201055.201056

[5] Christian Bird and Thomas Zimmermann. 2012. Assessing the value of branches with what-if analysis. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM, 45.

[6] Yuriy Brun, Reid Holmes, Michael D Ernst, and David Notkin. 2013. Early detection of collaboration conflicts and risks. *IEEE Transactions on Software Engineering* 39, 10 (2013), 1358–1375. https://doi.org/10.1109/TSE.2013.28

[7] Leuson Da Silva, Paulo Borba, Wardah Mahmood, Thorsten Berger, and João Moisakis. 2020. Detecting semantic conflicts via automated behavior change detection. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 174–184. https://doi.org/10.1109/ICSME46990.2020.00026

[8] Christian Hammer and Gregor Snelting. 2009. Flow-Sensitive, Context-Sensitive, and Object-Sensitive Information Flow Control Based on Program Dependence Graphs. *Int. J. Inf. Secur.* 8, 6 (oct 2009), 399–422. https://doi.org/10.1007/s10207-009-0086-1

[9] Susan Horwitz, Jan Prins, and T. Reps. 1989. Integrating noninterfering versions of programs. *ACM Trans. Program. Lang. Syst.* 11 (1989), 345–387.

[10] Susan Horwitz, Thomas Reps, and David Binkley. 1990. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 12, 1 (1990), 26–60. https://doi.org/10.1145/77606.77608

[11] Gleydson de Azevedo Ferreira Lima. 2014. Uma abordagem para evolução e reconciliação de linhas de produtos de software clonadas. (2014).
[12] Tom Mens. 2002. A state-of-the-art survey on software merging. *IEEE transactions on software engineering* 28, 5 (2002), 449–462.
[13] Fabrizio Pastore, Leonardo Mariani, and Daniela Micucci. 2017. BDCI: Behavioral driven conflict identification. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 570–581. https://doi.org/10.1145/3106237.3106296
[14] Galileu Santos, Paulo Borba, Rodrigo Bonifacio, and Matheus Barbosa. 2024. Poster: Lightweight Semantic Conflict Detection with Static Analysis. In *International Conference on Software Engineering*.
[15] Anita Sarma, David F Redmiles, and Andre Van Der Hoek. 2011. Palantir: Early detection of development conflicts arising from parallel code changes. *IEEE Transactions on Software Engineering* 38, 4 (2011), 889–908. https://doi.org/10.1109/TSE.2011.64
[16] Danhua Shao, Sarfraz Khurshid, and Dewayne E Perry. 2009. SCA: a semantic conflict analyzer for parallel changes. In *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. 291–292. https://doi.org/10.1145/1595696.1595747
[17] Léuson Mário Pedro da Silva. 2022. *Detecting, understanding, and resolving build and test conflicts*. Ph. D. Dissertation. Federal University of Pernambuco.
[18] Marcelo Sousa, Isil Dillig, and Shuvendu K Lahiri. 2018. Verified three-way program merge. *Proceedings of the ACM on Programming Languages* 2, OOPSLA (2018), 1–29. https://doi.org/10.1145/3276535
[19] Chungha Sung, Shuvendu K Lahiri, Mike Kaufman, Pallavi Choudhury, and Chao Wang. 2020. Towards understanding and fixing upstream merge induced conflicts in divergent forks: An industrial case study. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. 172–181. https://doi.org/10.1145/3377813.3381362
[20] Sheikh Shadab Towqir, Bowen Shen, Muhammad Ali Gulzar, and Na Meng. 2022. Detecting Build Conflicts in Software Merge for Java Programs via Static Analysis. In *37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13. https://doi.org/10.1145/3551349.3556950
[21] Wuu Yang, Susan Horwitz, and Thomas Reps. 1992. A program integration algorithm that accommodates semantics-preserving transformations. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 1, 3 (1992), 310–354. https://doi.org/10.1145/131736.131756
[22] Jialu Zhang, Todd Mytkowicz, Mike Kaufman, Ruzica Piskac, and Shuvendu K Lahiri. 2022. Using pre-trained language models to resolve textual and semantic merge conflicts (experience paper). In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 77–88. https://doi.org/10.1145/3533767.3534396
[23] Thomas Zimmermann. 2007. Mining workspace updates in CVS. In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 11–11.