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

Desenvolvedores frequentemente trabalham colaborativamente e precisam integrar seu código em uma versão principal do sistema. Esse processo pode causar conflitos de mesclagem, afetando a produtividade da equipe. Alguns desses conflitos exigem compreensão do comportamento do software (conflitos semânticos), e as ferramentas de controle de versão atuais não são capazes de detectá-los. Neste trabalho, temos como objetivo implementar, testar e avaliar técnicas de análise estática para detectar conflitos semânticos de integração de código. Dessa forma, buscamos abordar os desafios de integrar contribuições de software de forma eficaz. Por meio de uma revisão abrangente da literatura, implementação de ferramentas, experimentos em projetos significativos do GitHub e testes em cenários de trabalho reais, este estudo visa contribuir para a melhoria das práticas de integração contínua de software usando ferramentas de mesclagem semântica.

KEYWORDS

conflitos de integração de código, ferramenta de mesclagem semântica

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, 3 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Com a evolução nos processos de linhas de produção de *softwares*, criou-se ambientes de desenvolvimento colaborativos e com controle de versões de arquivos. Tarefas são atribuídas a vários desenvolvedores, que trabalham separadamente utilizando cópias locais dos projetos, desenvolvendo suas aplicações através de um repositório central. Promovendo assim, eficiência quando se trabalha de forma paralela e em equipe. Porém, ao tentar realizar a integração do projeto, as diferentes contribuições podem apresentar conflitos. Estes, podem ser detectados durante o *merge*, *build* ou teste, prejudicando a produtividade da equipe, pois entendê-los e corrigi-los é uma tarefa árdua e propensa a erros, além de trazer custos adicionais e disposição de tempo para resolvê-los.[4, 7, 14, 19].

Conflitos de *merge*, geralmente ocorrem quando são alterados os mesmos artefatos, quando dois ou mais desenvolvedores editam as mesmas linhas, ou linhas consecutivas do mesmo método, ou declaração do construtor[1]. Porém, edições em métodos dependentes também causam conflitos, principalmente quando um desenvolvedor altera um método invocado por outro método alterado por outro membro da equipe, visto que não gera conflito de *merge*, mas pode aumentar as chances de ocorrência de outros conflitos[13].

Uma solução parcial seria alertar os desenvolvedores no momento em que estão codificando que as suas alterações podem entrar em conflito com as mudanças de outros desenvolvedores, antes mesmo que elas sejam fundidas com o ramo principal. Com esse intuito, algumas ferramentas vêm sendo propostas [6, 9, 11, 15]. O uso desse tipo de abordagem é extremamente rápida e amplamente utilizado na indústria, porém são limitadas, atendendo apenas a detecção e resolução de conflitos de caráter textual. Outro ponto negativo é a alta possibilidade de disparar falsos positivos que tiram o foco do desenvolvedor. Por isso, podem ser tão prejudiciais quanto os próprios conflitos. Outra possível solução seria utilizar ferramentas de fusão semiestruturadas como o s3m [1].

Mas esse não é o único problema, existe um tipo de conflito que demanda um esforço ainda mais significativo para ser identificado e corrigido, pois, necessita-se de conhecimento das mudanças a serem mescladas, por isso não são detectados pelas ferramentas de controle de versão atuais. Duas contribuições advindas de versões *Left* e *Right* para um programa *Base* (versões envolvidas em um *three-way merge*[2], onde *Base* é o ancestral comum mais recente às duas versões *Left* e *Right*), originam um conflito semântico se as especificações que as versões se propõem a cumprir em isolado não são satisfeitas na versão integrada.[10]

Com conflitos semânticos em mente, se fazem necessárias ferramentas que possam detectar conflitos desse tipo no processo de integração, de modo a evitar bugs e facilitar a resolução dos mesmos. Nesse sentido, esse trabalho propõe um estudo sobre o uso de ferramentas de detecção de interferência com desenvolvedores reais em seus ambientes de trabalho.

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

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

[1] Ferramenta de mesclagem semiestruturada para aplicações (Java). https://github.com/guilhermejccavalcanti/jFSTMerge
[2] Uma mesclagem de três vias envolve três instantâneos. Dois são os que estão envolvidos em uma fusão bidirecional e o terceiro é o arquivo base ou ancestral comum com o qual esses dois arquivos serão comparados.

2 BACKGROUND AND RELATED WORKS

Esta seção apresenta das fontes bibliográficas que abordam a temática em questão, diante de um levantamento inicial da literatura existente sobre o tema. Algumas propostas, como, por exemplo, Kasi e Sarma[12] e Brun et. al.[7] analisaram cenários de conflitos de projetos open source, visando medir a frequência com que os diferentes tipos de conflitos acontecem. Ambos também apresentaram uma proposta de solução através da implementação de ferramentas, Cassandra e Crystal respectivamente.

No trabalho de Accioly at. al.[2], foi analisado a eficácia de dois tipos de mudanças de código (edições do mesmo método e edições para métodos diretamente dependentes) como preditores de conflito. Foi analisando parte da história do desenvolvimento de 45 projetos Java de GitHub e Travis CI, incluindo 5.647 cenários de fusão, para computar *precision* e o *recall* para os preditores de conflito. Os resultados indicam que os pré-ditadores combinados têm uma precisão de 57,99% e um recall de 82,67%.

Semelhante ao trabalho supracitado, temos Cavalcanti et. al.[8], que apresenta uma comparação entre os métodos de fusão textual ou não estruturada e semiestruturada. Foram reproduzidos 30.000 *merges* de 50 open source identificando conflitos relatados incorretamente por uma abordagem, mas não pela outra (falsos positivos), e conflitos relatados corretamente por uma abordagem, mas perdidos pela outra (falsos negativos). Nos resultados e análises complementares indicam melhores resultados para a fusão semiestruturada. Também foi implementada uma ferramenta de mesclagem semiestruturada que combina as duas abordagens para reduzir os falsos positivos e os negativos negativos da mesclagem semiestruturada. Encontraram-se evidências de que a ferramenta, quando comparada à mesclagem não estruturada na amostra estudada, reduz o número de conflitos relatados pela metade, não possui falsos positivos adicionais, tem pelo menos 8% menos falsos negativos.

Seguindo a mesma linha, em Shridhar et. al.[18] investiga-se qualitativamente a história de 18 projetos open source dos ecossistemas Apache e Eclipse, ao longo de um período de quatorze meses. As alterações de build "corretivas", "adaptativas" e "novas funcionalidades" introduzem uma rotatividade consideravelmente maior e são mais invasivas, enquanto muitas alterações são identificadas por acidente durante o desenvolvimento regular. Os autores também alertam que ter especialistas dedicados ao processo de build permite que projetos de *software* façam mudanças adaptativas mais invasivas.

Já em Perry et. al.[16] foi realizado um estudo de caso observacional, em que foram analisados a história de um sistema legado para delinear e compreender a natureza dos problemas encontrados no desenvolvimento paralelo. Os resultados mostram que o grau de paralelismo é muito mais alto quando comparado ao considerado pelos construtores de ferramentas e existem múltiplos níveis de paralelismo, havendo uma correlação significativa entre o grau de trabalho paralelo num determinado componente e o número de problemas apresentados pelos mesmos.

Por fim, em Bird e Zimmermann[5], caracteriza-se como os desenvolvedores usam ramos em um grande projeto industrial e os problemas comuns enfrentados. Um dos maiores problemas mencionados foi o longo atraso que uma alteração leva para mover-se de um time para outro, problemas este, frequentemente causado pela existência de muitas *branches*. Com o objetivo de monitorar a saúde dos ramos, foi realizada uma análise para avaliar estruturas de ramos com relação a duas propriedades: isolamento e vivacidade. Este modelo de análise foi aplicado em diversos cenários, inclusive no Windows, onde, segundo os autores, as alterações teriam economizado 8 a 9 dias de atraso.

3 COMPLETED WORKS

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 [3]

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 mesclagem 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 mesclagem de código. Mais detalhes deste trabalho podem ser encontrados em nosso artigo [17]

4 CONTRIBUTION TO KNOWLEDGE

Apesar dos avanços nas ferramentas de controle de versão, ainda existem desafios significativos na detecção de conflitos semânticos durante o processo de integração contínua de software. A falta de ferramentas adequadas para identificar esses conflitos pode resultar em bugs e atrasos no desenvolvimento do software.

Para abordar o problema dos conflitos semânticos, propomos uma ferramenta que engloba alguns tipos de análises estáticas, visando identificar interferências entre as alterações introduzidas por dois desenvolvedores diferentes.

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] Paola Accioly, Paulo Borba, Léuson Silva, and Guilherme Cavalcanti. 2018. Analyzing conflict predictors in open-source java projects. In *Proceedings of the 15th International Conference on Mining Software Repositories*. ACM, 576–586.

[3] 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

[4] 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.

[5] Christian Bird and Thomas Zimmermann. 2012. Assessing the value of branches with what-if analysis. In *SIGSOFT FSE*.

[6] Yuriy Brun, Reid Holmes, Michael D. Ernst, and David Notkin. 2011. Proactive Detection of Collaboration Conflicts. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (Szeged, Hungary) *(ESEC/FSE '11)*. Association for Computing Machinery, New York, NY, USA, 168–178. https://doi.org/10.1145/2025113.2025139

[7] 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.

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

[9] Lile Hattori and Michele Lanza. 2010. Syde: A Tool for Collaborative Software Development. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2* (Cape Town, South Africa) *(ICSE '10)*. Association for Computing Machinery, New York, NY, USA, 235–238. https://doi.org/10.1145/1810295.1810339

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

[11] Bakhtiar Khan Kasi and Anita Sarma. 2013. Cassandra: Proactive Conflict Minimization through Optimized Task Scheduling. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13)*. IEEE Press, 732–741.

[12] Bakhtiar Khan Kasi and Anita Sarma. 2013. Cassandra: Proactive conflict minimization through optimized task scheduling. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 732–741.

[13] Gleydson de Azevedo Ferreira Lima. 2014. Uma abordagem para evolução e reconciliação de linhas de produtos de software clonadas. (2014).

[14] Tom Mens. 2002. A state-of-the-art survey on software merging. *IEEE transactions on software engineering* 28, 5 (2002), 449–462.

[15] Martin Nordio, H. Estler, Carlo Furia, and Bertrand Meyer. 2011. Collaborative Software Development on the Web. *Computing Research Repository - CORR* (05 2011).

[16] Dewayne E. Perry, Harvey P. Siy, and Lawrence G. Votta. 2001. Parallel Changes in Large-Scale Software Development: An Observational Case Study. *ACM Trans. Softw. Eng. Methodol.* 10, 3 (jul 2001), 308–337. https://doi.org/10.1145/383876.383878

[17] Galileu Santos, Paulo Borba, Rodrigo Bonifacio, and Matheus Barbosa. 2024. Poster: Lightweight Semantic Conflict Detection with Static Analysis. In *International Conference on Software Engineering*.

[18] Mini Shridhar, Bram Adams, and Foutse Khomh. 2014. A Qualitative Analysis of Software Build System Changes and Build Ownership Styles. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Torino, Italy) *(ESEM '14)*. Association for Computing Machinery, New York, NY, USA, Article 29, 10 pages. https://doi.org/10.1145/2652524.2652547

[19] Thomas Zimmermann. 2007. Mining workspace updates in CVS. In *Fourth International Workshop on Mining Software Repositories (MSR'07: ICSE Workshops 2007)*. IEEE, 11–11.