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

The functionalities of the version control system enable developers to carry out their development tasks autonomously. Additionally, these tools simplify incorporating changes through merge operations and identify text conflicts. Textual conflicts are significant adversities, however, there are other forms of conflicts that can be even more harmful and are not identified by currently available merge tools. Dynamic semantic conflicts refer to situations in which there are no apparent textual conflicts during the merge reports, however, they result in unwanted interference that can cause unexpected behavior of the program during its execution. There are currently a variety of tools available that take different approaches to addressing this challenge. These range from static analyzes to the implementation of automated tests. However, it is observed that the test-centric approach generally results in a higher number of false negatives, while static analyzes tend to have a higher incidence of false positives. Additionally, these tools are limited to identifying interferences that occur within the same method or in subsequent methods, which may not reflect the real complexity of the systems. Furthermore, they were evaluated mainly in open source scenarios, without the direct insertion of human intervention in corporate environments. In this study, we propose improving static analyzes to enable their execution with different entry points, aiming to identify interferences in different areas of a system. Furthermore, we propose the creation of an infrastructure that integrates the Git merge process, to be implemented on developers' machines in a corporate context. This will enable the collection of merge reports and the conduct of qualitative assessments with the active participation of the professionals involved.

KEYWORDS

conflitos de integração de código, ferramenta de merge 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

Text merge tools, which operate based on lines of code, identify conflicts when merging two versions that apply changes to the same or consecutive lines [1]. Although these *textual conflicts* or *merge* conflicts are common and extensively studied, they do not represent the only problem when integrating the code of two developers.

Due to their limitation in considering only the combination of lines, textual merge tools cannot detect incompatible changes that occur in areas of the code separated by at least a single line. For example, if a developer changes the signature of a method and another adds a call to the method with the original signature but on different lines, textual merge tools integrate the code without conflicts or alerts. However, the resulting code may not compile, which sets up a *static semantic conflict* [6, 15, 17, 19, 20, 22]. Also called *Build* conflict.

Additionally, these tools cannot identify *dynamic semantic conflicts* [3, 6, 7, 9, 13, 16, 18, 21, 22], which occur when a developer's changes affect a state element accessed by code changed by another developer, resulting in unexpected behavior during program execution and since it does not generate a *merge* conflict, it may increase the chances of other [11] conflicts occurring. *test* and *production* (and undetected) conflicts correspond to dynamic semantic conflicts.

semantic conflicts, have the potential to harm both the development efficiency and the quality of software products. This impact is particularly notable in projects with divergent forks, as discussed in previous studies [5, 6, 12, 19, 22, 23]. However, even in projects that have only one remote repository, these conflicts can pose significant challenges.

Several approaches have been explored to deal with semantic conflict detection. For example, [17] test generation has been used, but faces the challenge of high false negative rates as it checks for interference based on specific input values during execution of the tested code.

On the other hand, static analysis algorithms have been proposed to identify semantic conflicts [3, 4, 9], using complex graphs, such as System Dependence Graphs. However, as the size of code bases increases, these approaches suffer significant performance degradation, making them impractical for applications on real code bases with more than 50 thousand lines of code. Given this scenario, there is a need to explore simplifications in existing algorithms, aiming to detect semantic conflicts with lower computational cost.

Recently, in our research, we introduced the use of simplified [14] static analyses, focused exclusively on the merged version of the code. Our approach involves tagging this version with metadata that indicates which instructions were modified or added by each developer. However, this approach faces two significant challenges. First, these tools are restricted to identifying interferences that manifest within the same or subsequent methods, which may not adequately represent the complexity inherent in real systems. Second, their evaluation was predominantly conducted in open source environments, where direct human intervention in corporate contexts was not considered.

In this context, this study proposes improvements in static analyses, aiming at adapting them to operate with different entry points, in order to identify interferences in different areas of a system. Furthermore, the article presents an investigation into the use of simplified static analyzes to detect interference, with the participation of real developers in their work environments.

2 BACKGROUND AND RELATED WORK

To effectively identify semantic conflicts, Da Silva *et al* [17] introduced a methodology based on the automated generation of unit tests, which serve as partial specifications for detecting interference in integration situations. The authors employ test passing criteria that suggest interference: if a generated test fails the base version, passes one of the developers, and then fails again in the merge version, this indicates that the change made by one of the developers did not is preserved in the merged version. In contrast, our approach relies on static analysis. Furthermore, we expect our performance to be better than theirs, although the information provided is limited. The combination of these two techniques could be a promising area for future investigation.

In the field of static analyses, Horwitz et al. were pioneers in the study of dynamic semantic conflicts [9, 10, 21] and formalized the definition of interference, a central concept adopted in this study. Their approach involves building Program Dependence Graphs (PDGs)[9] and System Dependence Graphs (SDGs)[10] for the four versions of the program in a merge scenario, analyzing the differences between them to detect and resolve interferences. In a more recent work, Barros Filho [3] using the JOANA framework[8] proposed the use of analyzes with the objective of investigating whether Information Flow Control (IFC) can be used to identify the presence of dynamic semantic conflicts between developer contributions in merge scenarios. However, these approaches can take hours to build, while our analyses are significantly cheaper.

We propose a lightweight technique that exploits the use of static analysis to detect interference when merging contributions from two [14] developers. However, we identified that existing analyzes are limited to identifying interferences within the same method or subsequent methods, which may not adequately reflect the complexity of the systems. Our goal is to improve these analyze to operate with different entry points, enabling the detection of interference in different parts of the program. Furthermore, we propose a qualitative evaluation with developers in their real work environments, aiming to better understand the usability of tools of this type in this specific context.

Sousa et al. [18] explore an alternative technique to detect interference. They statically infer relational post conditions from the code versions involved in the merge scenario, establishing restrictions on modifications of state elements by different versions to prevent interference. They then use theorem proving techniques, such as SMT solving, to verify the satisfaction of these constraints. Like some works mentioned above, this one uses github projects to validate its approach.

3 COMPLETED WORK

3.1 Semantic conflict detection with overriding assignment analysis

We propose and implement an attribution overlap analysis, which aims to detect interference between changes introduced by two different developers, where writing paths, without intermediate attributions, to a common target indicate interference. To evaluate implementations of the proposed analysis, analysis results for a set of 78 code integration scenarios were compared with Ground Truth for Overriding Assignment (OA) and Local Observable Interference (LOI). The results showed that the proposed analysis was able to detect interference between modifications, indicating its effectiveness as a tool for detecting semantic conflicts. However, there were a considerable amount of false negatives, suggesting that analysis alone is not sufficient for reliable interference detection. The precision of the analysis was 0.44 for the intraprocedural approach and 1.0 for the interprocedural approach, with an accuracy of 0.7 and 0.88, respectively. The results also highlighted the need to combine the proposed analysis with other techniques to create a more robust tool for detecting semantic integration conflicts. More details of this work can be found in our article [2]

3.2 Lightweight Semantic Conflict Detection with Static Analysis

In this work, we focus on dynamic semantic conflicts, which occur when merge reports do not present textual conflicts, but result in unwanted interference, causing unexpected program behavior at runtime. To solve this problem, we propose a lightweight technique that explores the use of four static analyzes (Interprocedural Data Flow, Interprocedural Confluence, Interprocedural Override Assignment, and Program Dependence Graph, which includes the control dependency that detects the interference) to detect interference when merging contributions from two developers. We evaluate our technique using a set of 99 experimental units extracted from GitHub project merge scenarios and chosen mainly from previous work related to semantic conflicts. The results provide evidence that our technique has significant interference detection capabilities. It outperforms, in terms of F1 score and recall, previous techniques that rely on test-based analysis to detect semantic conflicts, but has better accuracy. The accuracy of our technique is comparable to those observed in other studies that also use static analysis or use theorem proving techniques to detect semantic conflicts, although with significantly improved execution performance. These results demonstrate the ability of static analysis to significantly detect interference during the code merge process. More details of this work can be found in our article [14]

4 CONTRIBUTION TO KNOWLEDGE

Despite advances in merge tools, there are still significant challenges in detecting dynamic semantic conflicts during the software integration process. Several tools pursue varying approaches to solving this problem, but face performance challenges or limitations in detecting interferences that occur only within the same method or in subsequent methods. Furthermore, it is common that these tools have never been tested for their usability by humans.

In our work, we propose the creation of an infrastructure that integrates the Git merge process, to be implemented on developers’ machines in a corporate context, aiming to identify interferences between changes introduced by two different developers and carry out a qualitative study seeking to improve our understanding of the usability of tools of this type in this specific context.

The doctoral work is at an early stage, having recently completed two years of dedicated research. As highlighted in previous studies, we have already been able to validate the effectiveness of static analysis in identifying semantic conflicts in development environments, using relevant repositories hosted on GitHub. Now, we aim to advance in this field, seeking to apply our methodology in real work environments, with active developers in companies from different sectors.

Our proposal is to integrate our tool directly into these professionals’ workflows, allowing us to capture detailed data during execution. By collecting records of these interactions, we will have access to a more accurate and comprehensive view of the tool’s use in practical scenarios. This approach will enable us to not only validate the effectiveness of static analysis in a more dynamic and complex context, but also better understand the needs and challenges developers face on a daily basis.

From this data, we will be able to generate more informative and relevant reports, offering valuable insights for both developers and research and development teams. We hope that this direct collaboration with professionals in the field will not only strengthen the validity of our approach, but also contribute to the advancement of knowledge and practices in detecting and resolving semantic conflicts in software projects.

ACKNOWLEDGMENTS

The author received guidance from Dr. Paulo Borda, Member of the ACM and the Brazilian Computing Society, professor of Software Engineering at the Informatics Center of the Federal University of Pernambuco and leader of the Software Productivity Group. He is currently also director of the Information Technology Center.

This work is partially supported by grants IBPG-0567-1.03/22 from the Foundation for the Support of Science and Technology of the State of Pernambuco. Any opinions, findings, and conclusions expressed in this work are solely those of the author and do not necessarily reflect those of the sponsors. We would like to thank INES 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.