2017 IEEE/ACM 39th IEEE International Conference on Software Engineering Companion

Should we replace our merge tools?

Guilherme Cavalcanti, Paulo Borba and Paola Accioly
Informatics Center
Federal University of Pernambuco
Recife, Brazil
{gjcc,phmb,prga}@cin.ufpe.br

***Abstract*—While unstructured merge tools try to automatically resolve merge conflicts via textual similarity, semistructured merge tools try to go further by partially exploiting the syntactic structure and semantics of the involved artefacts. Previous studies compare these merge approaches with respect to the number of reported conflicts, showing, for most projects and merge situations, a reduction in favor of semistructured merge. However, these studies do not investigate whether this reduction actually leads to integration effort reduction (Productivity) without negative impact on the correctness of the merging process (Quality). To analyze this, and to better understand how these tools could be improved, we propose empirical studies to identify spurious conflicts reported by one approach but not by the other, and interference reported as conflict by one approach but missed by the other.**

***Keywords*-software merging; collaborative development; version control systems; empirical studies**

I. Introduction and Motivation

In a collaborative development environment, developers often perform tasks in an independent way, using individual copies of project files. As a result, when integrating changes from each task, one might have to deal with conflicting changes and dedicate substantial effort to resolve them. These conflicts might be detected during merging, building, and testing, impairing development productivity. They might also not be detected during integration and testing, escaping to a production version and compromising correctness [3], [4], [8]. To deal with these problems, tools using different strategies to decrease integration effort and improve integration correctness have been proposed. For example, unstructured merge tools, which are widely used in industry, are purely text-based and resolve conflicts via textual similarity [6]. Alternatively, a semistructured merge tool tries to resolve conflicts by partially exploiting the syntactic structure and semantics of the involved artefacts. For program elements whose structure is not explored by semistructured merge, it simply applies the usual textual resolution from unstructured merge [1].

Previous studies compare those two merge approaches with respect to the number of reported conflicts, showing, for most projects and merge situations, a reduction in favor of semistructured merge [1], [5]. This reduction is mainly due to the automatic resolution of some of the obvious unstructured merge reported spurious conflicts (false positives), which happens when, for example, developers add different methods to the same file text area. This evidence, however, needs to be further verified to justify industrial adoption of semistructured merge, since the observed reduction might be obtained at the expense of missing actual interference between developers changes (false negatives). Moreover, given that the set of conflicts reported by semistructured merge in previous studies is often smaller but not a subset of the set reported by unstructured merge, semistructured merge could even be introducing other kinds of false positives that might be harder to resolve than the ones it eliminates. As even a minor disadvantage of a new tool can become a huge barrier for its adoption in practice, if we want to move forward on the state of the practice on merge tools, it is important to have solid evidence and further knowledge about false positives and false negatives resulting from those two merge approaches. This way developers can choose the merge approach to be used in terms of their impact on integration effort (Productivity) and correctness (Quality)—factors that are critical to decide which approach should be used in practice.

II. Proposed Research

The main challenge for comparing the unstructured and semistructured merge approaches is establishing ground truth for integration conflicts (and therefore false positives and false negatives), or, more generally, unplanned interference between development tasks. In this context, interference is actually *not computable* [2]. Semantic approximations through static analysis or testing are imprecise and expensive, in the case of information flow analysis. Finally, experts who understand the integrated code (possibly developers of each analyzed project) would be necessary to determine truth, without completely eliminating the risk of missing false positives and false negatives. So, due to their associated cost, these alternatives would significantly reduce the analyzed sample without guarantees of precision. To avoid that, we prefer to relatively compare merge approaches with regard to the *added* occurrence of false positives and false negatives from one approach in relation to the other. We can do that by simply observing when they report different results.

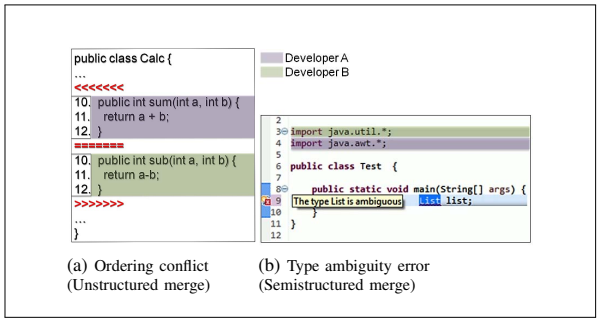
So, to compare the unstructured and semistructured merge approaches, we need first to understand their difference in behavior. We empirically and systematically analyzed their implemented algorithms to observe how and when they behave differently, and how this might lead to false positives and false negatives. Table I summarizes the observed added false positives and false negatives of each merge approach. For example, one of the main weaknesses of unstructured merge is its inability to detect rearrangeable declarations. In Java, for instance,

978-1-5386-1589-8/17 $31.00 © 2017 IEEE
DOI 10.1109/ICSE-C.2017.103

Table I: Unstructured and Semistructured merge added false positives and false negatives

False Positives		
Name	*Description*	*Merge Approach*
ordering conflicts	conflicts due to changes in the order of commutative and associative declarations	unstructured merge
renaming conflicts	conflicts due to changes in body and signature of method or constructor declarations	semistructured merge
False Negatives		
Name	*Description*	*Merge Approach*
duplicated declarations error	when developers add declarations with the same name/signature in different text areas	unstructured merge
type ambiguity error	when developers import members with the same name but from different packages	semistructured merge
new element referencing old one	when one developer adds a new element that references an existing one that is edited by the other developer	semistructured merge
anonymous blocks	when developers make changes in static blocks	semistructured merge

a change in the order of methods and fields, as in Figure 1(a), has no semantic impact on program behavior, but unstructured merge might report false positives in such cases— the so-called *ordering conflicts*. In contrast, this is not reported by semistructured merge. That ability to identify commutative and associative declarations, however, might also cause problems with semistructured merge. The semistructured merge algorithm assumes that the order of import declarations does not matter, allowing developers to add import declarations in the same text area of the program. However, this might lead to a *type ambiguity error* and build, or behavioral issues because the import declarations might involve members with the same name but from different packages. This is illustrated in Figure 1(b), where both developers imported packages with a `List` class. In that case, an unstructured tool would report a conflict when the import statements appear in the same or adjacent lines of the code.



(a) Ordering conflict (Unstructured merge)

(b) Type ambiguity error (Semistructured merge)

Figure 1: Ordering Conflict and Type Ambiguity Error. Conflict markers in red.

A. Empirical Evaluation

Our evaluation aims to investigate whether semistructured merge reduction on the number of reported conflicts in relation to unstructured merge [1], [5] actually leads to integration effort reduction (Productivity) without negative impact on the correctness of the merging process (Quality). We intend to do that by reproducing merges from the full development history of different GitHub projects, while collecting evidence about the occurrence of conflicts, and kinds of diverging false positives and false negatives described in Table I. In particular, we investigate the following research questions:

- **RQ1** *When compared to unstructured merge, does semistructured merge reduce unnecessary integration effort by reporting less spurious conflicts?*
- **RQ2** *When compared to unstructured merge, does semistructured merge compromise integration correctness by missing more non spurious conflicts?*

To answer **RQ1**, we compute the *number of false positives added by semistructured merge* (spurious conflicts reported by semistructured merge and not reported by unstructured merge) metric. We also compute the *number of false positives added by unstructured merge* (spurious conflicts reported by unstructured merge and not reported by semistructured merge) metric. As different conflicts might demand different resolution effort [7], comparing conflict numbers might not be enough for understanding the impact on integration effort. So, to better understand the effort needed to resolve different kinds of conflicts, we intend to manually analyze a sample of the identified false positives to estimate the impact on integration effort. Our goal with this analysis is to simply check that the computed metrics are not obviously bad choices as proxies for integration effort. Finally, for answering **RQ2**, we compute the *number of false negatives added by semistructured merge* (conflicts missed by semistructured merge and correctly reported by unstructured merge), and the *number of false negatives added by unstructured merge* (conflicts missed by unstructured merge and correctly reported by semistructured merge) metrics.

It is important to remark that we do not need to measure the occurrence of false positives and negatives when both approaches behave identically. Although important for establishing accuracy in general, these are not useful for relatively comparing merge approaches.

III. Conclusions

Previous studies provide evidence that semistructured merge reduces, for most but not all projects and scenarios, the number of conflicts in relation to unstructured merge [1], [5]. However, practitioners would hardly adopt a new merge tool without knowing whether this reduction actually leads to merge effort reduction without compromising the correctness of the merging process. So, as even a minor disadvantage of a new tool can become a huge barrier for its adoption in practice, in this paper, we propose a research to relatively compare these merge approaches with respect to the resulting occurrences of false positives and false negatives. In particular, false positives represent unnecessary integration effort, which decreases productivity, because developers have to resolve conflicts that actually do not represent interference. Besides that, false negatives represent build or behavioral errors, negatively impacting software quality and correctness of the merging process.

REFERENCES

[1] S. Apel, J. Liebig, B. Brandl, C. Lengauer, and C. 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. ACM, 2011.

[2] V. Berzins. On merging software extensions. *Acta Informatica*, 1986.

[3] C. Bird and T. 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. ACM, 2012.

[4] Y. Brun, R. Holmes, M. D. Ernst, and D. 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. ACM, 2011.

[5] G. Cavalcanti, P. Accioly, and P. Borba. Assessing semistructured merge in version control systems: A replicated experiment. In *Proceedings of the 9th International Symposium on Empirical Software Engineering and Measurement*, ESEM'15. ACM, 2015.

[6] S. Khanna, K. Kunal, and B. C. Pierce. A formal investigation of diff3. In *Proceedings of the 27th International Conference on Foundations of Software Technology and Theoretical Computer Science*, FSTTCS'07. Springer-Verlag, 2007.

[7] T. Mens. A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering*, 2002.

[8] T. Zimmermann. Mining workspace updates in cvs. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR'07. IEEE Computer Society, 2007.