Understanding Conflicts Arising from Collaborative Development

Paola Accioly
Informatics Center
Federal University of Pernambuco, Recife, Brazil
Email: prga@cin.ufpe.br
Website: http://www.cin.ufpe.br/~prga

***Abstract*—When working in a collaborative development environment, developers implement tasks separately. Consequently, during the integration process, one might have to deal with conflicting changes. Previous studies indicate that conflicts occur frequently and impair developers' productivity. Such evidence motivates the development of tools that try to tackle this problem. However, despite the existing evidence, there are still many unanswered questions. The goal of this research is to investigate conflict characteristics in practice through empirical studies and use this body of knowledge to improve strategies that support software developers working collaboratively.**

I. Introduction

In a collaborative development environment, tasks are assigned to developers that work separately using individual copies of project files. As a result, while trying to integrate different contributions, one might have to deal with conflicting changes. Such conflicts may hamper productivity, since detecting and solving conflicts is a tiresome and error prone activity. To learn about the occurrence of conflicts and their consequences, previous empirical studies [1]–[4] answer questions concerning how often conflicts occur and when developers detect them (during different integration activities). These studies bring evidence that conflicts occur often and impair developers' productivity. Such evidence motivates and guides the design of tools [1], [2], [4], [5] that use different strategies to both decrease integration effort and improve correctness during integrations.

The empirical studies done so far describe the frequency of conflicts, but they do not explore conflict characteristics such as what are the involved syntactic language structures, expressing the root causes in terms of the performed kinds of changes. This information can lead to an improvement of existing strategies and tools. For example, tools that provide users with information of ongoing parallel changes such as Crystal [2] and Palantír [5] can benefit from knowing most common conflict patterns to improve their performance and reduce false positives. Likewise, Cassandra [1], a tool that evaluates task constraints to recommend optimum task orders for developers, can benefit from this information to improve its constraint rules.

Therefore, in this work, we propose empirical studies to bring evidence about conflict characteristics. By analysing such evidence, we intend to derive requirements and guidelines for improving existing strategies and tools for conflict detection and resolution. As a first step in our research, we analysed the SSMerge [4] tool and derived conflict patterns abstracting the kinds of conflicts that can be detected by this tool. Each pattern captures the syntactic elements involved in a conflict; in particular, we are interested in the structure of the individual changes performed by two developers, and how they lead to a conflict. For example, the pattern *Add method or constructor with same signature and different bodies*, capture the situation when two developers working independently add methods with the same signature but with different behaviours.

After deriving the catalog, we selected so far 31 projects from GitHub, reproducing more than 7500 integration scenarios to assess the frequency of conflict patterns. Although this analysis is still in progress, our preliminary results show that most of the conflicts (84%) occur inside methods, which reveals the need for further studies to analyse performed changes inside methods. In the remainder of this paper, we detail prior work and motivation for this research (Section II). Then we describe our research approach (Section III), and discuss the remaining work and expected contributions (Section IV).

II. Background and Motivation

In a collaborative development environment, tasks are assigned to developers that work independently. After that, during the integration process, conflicts might occur during the merge step (due to *merge* conflicts), while building the system (due to *build* conflicts) or running the tests (due to semantic conflicts). To learn about these conflicts, previous studies provide evidence that they indeed occur frequently. Brun et al. [2] and Kasi and Sarma [1] found that merge conflicts occurred in a average of 15% of integration scenarios, and, in average, 31% of integration scenarios free of merge conflicts resulted in build or semantic conflicts.

Those studies motivate the development of tools that use different strategies to tackle conflicts. For example, Cassandra [1] evaluates task constraints to recommend an optimum order of tasks for developers so that the risk of having potential conflicting tasks being developed at the same time decreases. Conversely, Crystal [2] performs code integration in the background of each individual repository to warn developers if their changes conflict with other tasks before integration. Palantír [5], in contrast, gives awareness to developers when they edit the same file so that they can communicate early.

There are other tools with similar purposes available, but, for the sake of brevity, we do not mention them here.

Nevertheless, previous empirical studies do not answer a number of questions. In fact, our assumption is that, despite the success of such tools, there are still opportunities for further enhancements. We believe that by learning more about conflict characteristics, such as involved syntactic language structures, we could improve existing strategies and tools that deal with conflicts. For example, using Crystal is computationally expensive since it performs its analysis at a regular time period (10 minutes by default) if new changes are committed during this period. If we had evidence on common conflict patterns, Crystal could process the task before performing the integration routine to check if it did not contain those patterns and could delay the integration until the subsequent time period. In addition, awareness tools like Palantír have the potential to raise false positives when developers edit the same file, and our evidence could reduce false alarms. Finally, Cassandra could use this evidence to improve its task constraints rules in different situations. Namely, when ordering high priority tasks involving the same files, it could analyse which of them involved the syntactic structures with more chance of causing conflicts.

III. Proposed Research and Preliminary Results

Given the motivating context presented earlier, we want to take a step back before proposing new strategies and tools for supporting collaborative development in order to analyse conflict characteristics through empirical studies. Then, we intend to use this body of knowledge as input to discuss improved ways to better support collaborative development. As a first step in this research, we started investigating the nature of merge conflicts, mainly because they occur often and are the first to emerge during tasks' integration. Considering that we want to explore conflict patterns and their frequency, we propose the following research questions:

- RQ_1: What are the merge conflict patterns?
- RQ_2: How often such patterns occur?

In order to answer RQ_1, we chose the SSMerge [4] tool to derive conflict patterns by abstracting the kinds of conflicts that can be detected by this tool. Each pattern captures the syntactic elements involved in a conflict; in particular, we are interested in the structure of the individual changes performed by two developers, and how they lead to a conflict. In total we describe the following 6 conflict patterns for programs written in Java:

1) Make different edits (including removal) to the same area of the same method, constructor or field declaration;
2) Add methods or constructors declarations with the same signature and different bodies;
3) Add field declarations with the same id and different types or modifiers;
4) Make different edits (including removal) to the modifiers list of the same type declaration (class, interface, annotation or enum types);
5) Make different edits (including removal) to the same list of implements declaration;
6) Make different edits (including removal) to the default value of the same annotation method declaration;

Due to lack of space here we do not illustrate examples of instances for the patterns, but interested readers can find a better description of the catalog in our online appendix [6]. Moving on towards our goal of answering RQ_2, we implemented a program named Conflict Analyser. This program identifies and downloads all integration scenarios from a list of selected projects from GitHub, calls SSMerge to merge them, collects the reported conflicts and matches them with their specific pattern. In addition, the conflict analyser tool computes two metrics, the conflict ratio, that is, the percentage of integration scenarios that resulted in conflicts, and the number of occurrences for each conflict patterns.

So far we analysed a total of 31 projects that amounts to more than 7500 integration scenarios. The average conflict ratio was 8%. Notice that this average is lower than the average conflict ratio of 15% reported by Brun et al. [2] and Kasi and Sarma [1]. However, we expected lower conflict ratios since we used the SSMerge tool which reduces the number of false positives compared with a traditional merge tool. Moving on with the analysis, the most frequent conflict pattern was when developers made edits to the same method, representing 84% of conflicts. The second most frequent conflict pattern was when developers added methods with same signature and different bodies, representing 13% of conflicts. The other conflict patterns did not appear often. These results reveal the need to take a closer look on what happens inside those methods. Also, we intend to analyse build and semantic conflicts to achieve more solid evidence to help improving existing strategies and tools. On Section IV we propose further studies that address those problems.

IV. Remaining work and Expected Contributions

As mentioned earlier, the merge conflict pattern analysis is still in progress. Our goal is to bring as much evidence as possible by increasing the diversity [7] of our sample. However, there are still two directions that we intend to follow. First we want to extend the merge conflict pattern study by using the JDime tool [8] instead of SSMerge. JDime offers more precise conflict detection inside methods. Second, we need to investigate the nature of build and semantic conflicts as well. One possibility would be using program analysis tools to verify the information flow between the two tasks and investigate how they interfere with each other. One possibility is to build this analysis on top of the Java bytecode information flow control framework Joana. [1] In addition we are looking forward for computing new metrics such as what factors influence the occurrence of conflicts and how they are distributed along the development. Lastly, as our results progress, we will provide more insights to better support collaborative development.

[1] http://pp.ipd.kit.edu/projects/joana/

ACKNOWLEDGMENT

We would like to thank FACEPE Brazilian research funding agency grants IBPG-0716-1.03/12, INES, funded by CNPq and FACEPE, grants 573964/2008-4 and APQ-1037-1.03/08, for partially supporting this work.

REFERENCES

[1] B. K. Kasi and A. Sarma, "Cassandra: Proactive conflict minimization through optimized task scheduling," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. Piscataway, NJ, USA: IEEE Press, 2013.

[2] Y. Brun, R. Holmes, M. Ernst, and D. Notkin, "Early detection of collaboration conflicts and risks," *Software Engineering, IEEE Transactions on*, vol. 39, no. 10, 2013.

[3] T. Zimmermann, "Mining workspace updates in CVS," in *Proceedings of the Fourth International Workshop on Mining Software Repositories*, ser. MSR '07. Washington, DC, USA: IEEE Computer Society, 2007.

[4] 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*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011.

[5] A. Sarma, D. F. Redmiles, and A. van der Hoek, "Palantir: Early detection of development conflicts arising from parallel code changes," *IEEE Transactions on Software Engineering*, vol. 38, no. 4, 2012.

[6] "Online appendix," http://goo.gl/jmVJW7, accessed: 2014-11-14.

[7] M. Nagappan, T. Zimmermann, and C. Bird, "Diversity in software engineering research," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2013. New York, NY, USA: ACM, 2013.

[8] S. Apel, O. Lessenich, and C. Lengauer, "Structured merge with autotuning: Balancing precision and performance," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE 2012. New York, NY, USA: ACM, 2012.