Detecting, Understanding and Resolving Build and Test Conflicts

Léuson Silva
Informatics Center
Federal University of Pernambuco, Brazil
lmps2@cin.ufpe.br

***Abstract*—Collaborative software development allows developers to contribute to the same project simultaneously performing different activities. Although this might increase development productivity, it also brings conflicts among developers contributions. Different kinds of conflicts can arise, but previous studies have often focused on merge conflicts. So we aim to further investigate build and test conflicts occurrence, that are conflicts revealed by failures when building and testing integrated code, respectively. For that, we intend to study the causes of build and test conflicts, their adopted resolution patterns, and the factors that are associated with the conflict occurrence. Based on these results, we plan to develop a tool for helping developers when resolving build and test conflicts. Our initial results, analyzing Java projects, show that most build conflicts are caused by missing declarations removed or renamed by one developer but referenced by the changes of another developer. We also verified these conflicts are often resolved by removing the dangling reference. Based on such finding, we developed a prototype that recommends fixes for these build conflicts.**

***Index Terms*—Collaborative development, build fails, build conflicts, test conflicts, conflicting contributions**

I. Introduction

When collaborating, developers create and change software artifacts often without full awareness of changes being made by other team members. While such independence is essential for non-small teams, and might increase development productivity, it might also result in conflicts when integrating developers changes. In fact, high degrees of parallel changes and integration conflicts have been observed in a number of industrial and open-source projects that use different kinds of version control systems [1]–[3]. This has been observed even when using advanced merge tools [4]–[6] that avoid common spurious conflicts identified by the state of the practice tools.

Resolving such integration conflicts might be time-consuming and is an error-prone activity [7], negatively impacting development productivity. So, to avoid dealing with them, developers have been adopting risky practices such as rushing to finish changes first [7], and partial check-ins. Similarly, partially motivated by the need to reduce conflicts, or at least avoid large conflicts, development teams have been adopting techniques such as trunk-based development and feature toggles, which are important to support actual Continuous Integration, but might lead to extra code complexity. Although evidence in the literature is mostly limited to merge conflicts [1], [6] and other kinds of build errors [8], [9], a couple of studies investigate the frequency of build and test conflicts [2], [3], that is, conflicts revealed by failures when building and testing integrated code. Better understanding these aspects might help us to derive guidelines for avoiding conflicts, improve awareness tools to better assess conflict risk, and develop new tools for helping developers when resolving conflicts.

Thus, this thesis aims to investigate how build and test conflicts occur and their impact during software development. The remainder of this paper is organized as follows. Section II describes our research questions and the associated research approach proposed to address this thesis problem. Section III states our initial answers, expected contributions, and future work.

II. Problem Statement and Research Approach

To guide our work while trying to address this thesis problem, we define three research questions that allow us to study the characteristics of conflicts, and the development of tools responsible for helping developers. Next, we present our research questions (*RQs*):

RQ1. *What are the frequencies, causes and resolution patterns associated with builds and test conflicts?*
Previous studies [2], [7] investigate build and test conflicts with focus on identifying superficial characteristics of conflicts, like frequency and limited structure of changes. To have a better and general understanding of the characteristics of conflicts, this *RQ* investigates the structure of changes performed during different contributions (merge scenarios), that are responsible for the occurrence of conflicts after integration. Additionally, for each identified conflict, its resolution pattern is also investigated.

RQ2. *What are the human-technical factors that are associated with the occurrence of build and test conflicts?*
With the findings of the previous *RQ*, we aim to investigate the factors associated with build and test conflicts. Previous studies investigate factors that are associated with the occurrence of conflicts during software development, but none of them focuses on build and test conflicts [10]. Once these factors are identified, they could be applied into predictive models contributing to the reduction of conflict occurrence.

RQ3. *How assistive tools could help developers when resolving build and test conflicts?*
The third research question investigates new alternatives for helping the resolution of build and test conflicts. For that, we

intend to develop an assistive tool. As a result, we expect that the time spent to fix the conflicts decrease contributing to the developer productivity. Additionally, the quality of the released product would be positively impacted once resolving conflicts represents an error-prone activity.

To address each *RQ*, we adopt different research methods. For *RQ1*, we intend to analyze Java project repositories aiming to verify the occurrence of conflicts. We have initial answers that are presented in Section III. Based on the answers of *RQ1*, for *RQ2* we intend to analyze the found conflicts identifying factors that can be associated with the occurrence of these conflicts. We also plan to analyze technology web forums and interview software engineers exploring different dimensions of software development. We are interested on aspects related to the development process, task assignments, and team and projects characteristics. We also plan to explore task duration aspects, like size and number of commits, time spent and number of developers involved in the whole task.

Finally, to address *RQ3*, we intend to develop a tool aiming to help developers when resolving conflicts. This tool may recommend a possible fix for the conflict, and the developer may evaluate whether he/she accepts the fix. For each conflict type, a specific recommendation will be defined by the tool. These recommendations will be extracted from the catalogue of resolution patterns identified during the *RQ1*.

III. Results

We have initial answers for the first research question (*RQ1*). For that, an empirical study evaluated 57065 merge scenarios from a sample of 504 open source Java projects hosted on GitHub and are active on Travis CI. As a result, we find consistent cases of build conflicts, 647 conflicts spread on 128 merge scenarios, deriving a catalogue of causes. Additionally, we also identified a catalogue of common conflict resolution patterns.

The frequency of build conflicts shows that most conflicts are caused by declarations removed or renamed by one developer but referenced by the changes of another developer. Moreover, these conflicts are often resolved by removing the dangling reference. Based on this finding, we developed a prototype that recommends to developers fixes for build conflicts caused by *Unavailable Symbol*. Considering a build breakage caused by a dangling reference for a variable, the tool identifies the new variable name. Thus, it asks the developer if he/she wants to accept the recommendation. Once the answer is positive, the tool applies the required changes and creates a new commit for the implemented fix.

Although we identified a consistent catalogue of build conflicts, the results for test conflicts are superficial and limited yet. For the next steps, we intend to select more projects analyzing private and developer workspace repositories from local companies. Our claim is that build and test conflicts happen more often than we verified, but they do not come to remote repositories as developers treat them locally, in their particular workspaces.

A. Contributions and Future Work

This thesis may bring insights about software maintenance and evolution, but from the perspective of contribution integration. As a result, it is expected to be provided:

- datasets and catalogues of build and test conflicts and their resolution patterns *(RQ1)* that could be explored and support further research;
- a set of technical-human factors associated with conflict occurrence *(RQ2)*, which might be applied into predictive models;
- a tool for helping developers to resolve these conflicts *(RQ3)* that could be integrated into the developer workspace.

I has just started my second PhD year, and before starting to write my PhD thesis, I plan to execute the proposed research questions as they are presented here. This year I will continue to work on *RQ1* focusing on test conflicts. Next year, I plan to work nine months in *RQ2* starting with the interviews, and then additional steps described on Section II. After that, I will start the development of the proposed tool (*RQ3*) and its evaluation. The last six months will be saved to work on the PhD thesis, which may be defended by February 2022.

IV. Acknowledgment

I thank my adviser Paulo Borba for his continuous support and patience. This work is partially supported by INES (National Software Engineering Institute), and the Brazilian research funding agencies CNPq (grant 309741/2013-0), FACEPE (IBPG-0692-1.03/17 and APQ/0388-1.03/14) and CAPES.

References

[1] T. Zimmermann, "Mining workspace updates in CVS," in *International Workshop on Mining Software Repositories*. IEEE, 2007, pp. 11–11.

[2] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Proactive detection of collaboration conflicts," in *European software engineering conference and Foundations of Software Engineering*. ACM, 2011, pp. 168–178.

[3] B. K. Kasi and A. Sarma, "Cassandra: Proactive conflict minimization through optimized task scheduling," in *International Conference on Software Engineering*. IEEE, 2013, pp. 732–741.

[4] S. Apel, O. Leßenich, and C. Lengauer, "Structured merge with auto-tuning: balancing precision and performance," in *International Conference on Automated Software Engineering*. ACM, 2012, pp. 120–129.

[5] G. Cavalcanti, P. Borba, and P. Accioly, "Evaluating and improving semistructured merge," *ACM on Programming Languages*, vol. 1, no. OOPSLA, p. 59, 2017.

[6] P. Accioly, P. Borba, and G. Cavalcanti, "Understanding semi-structured merge conflict characteristics in open-source java projects," *Empirical Software Engineering*, pp. 1–35, 2017.

[7] 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*, pp. 889–908, 2012.

[8] H. Seo, C. Sadowski, S. Elbaum, E. Aftandilian, and R. Bowdidge, "Programmers' build errors: a case study (at Google)," in *International Conference on Software Engineering*. ACM, 2014, pp. 724–734.

[9] T. Rausch, W. Hummer, P. Leitner, and S. Schulte, "An empirical analysis of build failures in the continuous integration workflows of java-based open-source software," in *International Conference on Mining Software Repositories*. IEEE Press, 2017, pp. 345–355.

[10] M. Cataldo and J. D. Herbsleb, "Coordination breakdowns and their impact on development productivity and software failures," *IEEE Transactions on Software Engineering*, vol. 39, no. 3, pp. 343–360, 2013.