How Do Developers Resolve Merge Conflicts? An Investigation Into the Processes, Tools, and Improvements

Caius Brindescu
Oregon State University
Corvallis, OR
brindesc@oregonstate.edu

ABSTRACT

Most software development is done in teams. When more than one developer is modifying the source code, there is a change that their changes will conflict. When this happens, developers have to interrupt their workflow in order to resolve the merge conflict. This interruption can lead to frustration and lost productivity. This makes collaboration, and the problems associated with it, an important aspect of software development. Merge conflicts are some of the more difficult issues that arise when working in a team.

We plan to bring in more information about the strategies developers use when resolving merge conflicts. We will gather information through in-situ observations and interviews of developers resolving conflicts when working on real development tasks, combined with analytical methods. The information obtained can then be used to improve the existing tools and make it easier for developers when working in a collaborative environment.

This work is advised by Prof. Dr. Carlos Jensen and Prof. Dr. Anita Sarma.

CCS CONCEPTS

• **Software and its engineering → Software version control**; **Programming teams**;

KEYWORDS

Merge Conflicts, Version Control Systems, Information Foraging Theory, Developer Processes

ACM Reference Format:
Caius Brindescu. 2018. How Do Developers Resolve Merge Conflicts? An Investigation Into the Processes, Tools, and Improvements. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3236024.3275430

1 RESEARCH PROBLEM

All of today's software is developed in teams. For example, in 2017 over 4000 individuals submitted and had changes accepted in the Linux Kernel. Over 83,000 changesets have been merged in the kernel over the same period [9]. This scale of development exacerbates the problems that merge conflicts pose. The larger the development team, the more likely it is that merge conflicts will occur.

Previous work has looked at how to detect merge conflicts early. This would allow developers to approach them before they became too complex, or difficult to solve. It has also investigated ways to increase developer awareness within a team. This would help developers coordinate, and, avoid conflicts if possible. However, the core problem, the resolution, still remains unaddressed. Once a conflict occurs, a developer has to resolve it, if they want to integrate their work into the final product. The resolution becomes even more critical if the changes important updates, such as bug-fixes or security updates.

Currently, we have little to no understanding of the how developers approach a merge conflict. How often do merge conflicts occur? What are their strategies for solving them? What is the information that they need? Do the toolsets available provide the information and support developers need to successfully resolve the merge conflicts? We cannot improve existing tools without a deeper understanding of the resolution process.

Answering these questions will help tool builders provide developers with better support for merge conflict resolution. It will also bring awareness of the current difficulties developers face. This awareness will benefit the entire research community and open new avenues of research into the difficulties faced during collaborative development.

2 RELATED WORK

2.1 Workspace Awareness

Biehl et al. [4] propose *FastDASH* and da Silva et al. [10] propose *Lighthouse*, providing a dashboard that shows the files that are checked out, modified, and staged by other members of the team.

Sarma et al. [23, 24] go a step further and propose *Palantír*. Palantír monitors other developer's workspaces, and, depending on the changes, will notify the developer, in a non-obtrusive manner, if a conflict has happened. Similarly, Hatori and Lanza [14] propose *Syde* that monitors the changes at an Abstract Syntax Tree (AST) level.

Brun et al. [8] propose *Crystal*, which monitors selected branches in the repository. Crystal preemptively merges the branches in the background and will notify the developers of any conflicts that arise.

Kasi and Sarma [16] take a more proactive approach and propose a novel task scheduling approach that aims to minimize the number

ESEC/FSE '18, November 4–9, 2018, Lake Buena Vista, FL, USA

ACM ISBN 978-1-4503-5573-5/18/11.
https://doi.org/10.1145/3236024.3275430

of conflicts. However, with all these approaches, once a conflict has occurred, it is up to the developer to resolve it.

2.2 Merging tools

Currently, all version control systems treat source code files as text. Therefore, merging is done at a textual level, ignoring all structure that the files might contain. Several researchers have looked at ways to improve this status quo.

Westfechtel [27] proposes a merging technique that uses the structural (i.e. lexical) information of a language when performing a merge. However, such tools are language dependent and the required algorithms are computationally expensive. Apel et al. [2] propose *JDime*, which improves existing structured merging techniques by only using structural information when the unstructured (i.e. text only) merge has failed. Binkley et al. [5] propose using call graph information to correctly merge different versions of the program. Lippe and van Oosterom [17] go a different way and propose a new merging technique, *operation-based merging* that would replay the changes that were performed on the two branches, in the order in which they were performed. Dig et al. [11] uses this technique and shows empirically that many more merge conflicts could be solved by a tool that understood the semantics of change operations.

The previous work has focused on various way to improve the resolution process, and the tool support for it. However, we currently don't have an understanding of *why* merge conflict are difficult. My work aims to bridge the gap between our understanding of merge conflicts and the tool support for them.

2.3 Empirical Studies

Guzzi et al. [13] conducted an exploratory investigation and tool evaluation for supporting collaboration in teamwork within the IDE. The found that, while automatic merge tools were used, developers did not trust them, and manually checked the end result.

Finally, McKee et al. [18] present an empirical study where they investigate developers' perceptions regarding merge conflicts. They find that developers initial perception of a merge conflict has a significant impact on the tools and processes they use for resolving it. They also find that developers use relatively simple metrics (lines of code in conflict, the complexity of the conflicting code) and their own expertise in the area of the conflict code to judge the conflicts difficulty. While this work focuses on the perceptions of developers when they encounter merge conflict, it does not tackle the resolution of a merge conflict.

2.4 Information Foraging Theory

Ragavan et al. [22] propose a variant of Information Foraging Theory (IFT), that models how developers forage in the presence of software history, or "variations," as they define it. This model highlights the needs of developers attempting to understand variations in code, whereas I examine how developers resolve a conflict. More specifically, I examine how developers approach the conflict, the tools they use to solve and the information they need to successfully resolve it.

Flemming et al. [12] use Information Foraging Theory to identify *design patterns* that successful tools use. Their study investigates only refactoring, debugging and software reuse tools. We will look at how to apply these design patterns to improve tools used for resolving merge conflicts.

2.5 Program Comprehension

Borg et al. [6], through interviews, look at how tools support *Change Impact Analysis*. Wang et al. [26] propose a technique that uses version history and previous bug reports to help developers localize bugs. Panichella et al. [21] explores how collaboration affects the structure of the source code. Tao et al. [25] investigate how developers understand software changes, in an industrial setting.

While existing work looks at understanding changes to source code, or working with history, no research has been done to investigate the comprehension of the changes that are involved in a merge conflict.

3 PROPOSED APPROACH

In my previous work I have shown that program elements involved in a merge conflict are more likely to be buggy [1]. I have also shown that changing the tools does influence the developers' behavior, as developer have different committing habits, depending if they use SVN or Git [7]. Therefore, by understanding the developers needs, and using this understanding to improve the current tool sets developers use, we can aim to improve the quality of the underlying code. I plan to do this in two steps: first understand what developers' needs are when resolving merge conflicts and, second, by evaluating the tools to see if they provide the right support developers need. The next two subsections will present each of the two steps in detail.

3.1 Understanding the Processes Developers Use When Resolving Merge Conflicts

To get the most accurate data, the best option is to observe developers while they are resolving merge conflicts. This will provide us with "ground truth" data, that can then be used to better understand the developer's needs, and if the tools currently at their disposal satisfy those needs. We plan to collect this data by performing field studies that will involve several professional software development teams.

Direct observation also eliminates some of the threats that come with interviews or surveys. For example, participants might present us a "polished" version of events, or might not provide relevant information that they might consider to be unimportant [3]. Similarly, bringing people in a laboratory for a controlled study will affect the external validity of the findings. We are also at risk of losing valuable information because we did now allow the developers to use their own tools and techniques for solving merge conflicts, in an environment they are familiar with.

3.2 Evaluating Existing Tool Support

Resolving merge conflicts requires the developer to use and find information regarding the changes they are about to merge. In many cases, they are unaware of the incoming changes, as they have written by another developer. Do the tools they use actually help them find the relevant information they need? Is this information presented in an understandable way?

First, we will need to understand what information developers use when resolving merge conflicts. Using the model derived from the field study (Section 3.1), we will perform an Information Foraging Theory (IFT) analysis to understand developers' information needs.

On the other hand, we need to understand the information the tools provide, when it comes to resolving merge conflicts. Using IFT, we can understand and quantify the information provided by existing tools. We will use the methodology presented by Fleming et al. [12].

These two approaches will allow us to identify gaps between what the developers need and what the tools provide. I will then provide improvements using the IFT *design patterns* suggested by [12].

We plan to incorporate individual differences into our evaluation (such as experience level, gender etc.), in order to help tool builders build tools that will help individual developers as well as than the "generic" developer. Such individual difference are not considered in the merge conflict literature, but have been shown to be important factors [15, 19, 20].

4 CONTRIBUTIONS

The high-level goal of my work is to understand the problems merge conflicts pose for developers. Currently, we have no understanding of *how* developers resolve a merge conflict. What tools do they use? What information do they need? How do they get that information? Without knowing the answers to these questions, we cannot improve upon the state of the art.

We will present a model of the process developers use when resolving merge conflicts. This model can then be used to evaluate the developer's information needs, and the information provided by existing toolsets.

Secondly, we will examine the existing tools to see if they support the current developer's practices. We expect to see important gaps between what developers need and what tools offer. Our hypothesis is that those gaps increase the effort developers have to expend in order to successfully resolve a merge conflict. Identifying those gaps can then inform the development of better merge resolution tools.

We will also provide a set of improvements, based on the IFT analysis, of how those gaps can be closed, or at least reduced.

Our expectation is that some gaps will be relatively easy to fill while closing other others might require significant reengineering by tools builders. By building a model of the process developers' use when resolving a merge conflict, we can bring more understanding of their needs. Our operationalization of this model, through the evaluation of existing tools, will help make merge conflict resolution less painful, and, hopefully, less error-prone.

5 EVALUATION PLAN

During the field study, we will collect screen recordings of developers resolving merge conflicts, and perform retrospective interviews. The data from the field study will be analyzed using qualitative methods, such as open coding, to identify the processes developer use when resolving a merge conflict. In order to validate the results, we will triangulate our findings by collecting data from multiple organizations. This will improve the generalizability of our results.

For evaluating the tool support we will use known analytical techniques (such as IFT), we can evaluate the existing tools in terms of their "fitness" for the processes developers employ. In order to suggest improvements to existing tools, we will use an approach similar to Flemming et al. [12]. These improvements will take the form of IFT *"design patterns,"* which, like software engineering design patterns, are common solutions to frequent information needs. By using the model as our starting point, and evaluating the tools through the lens of this model, the improvements will aim at directly reducing some of the pain points developers encounter while resolving a merge conflict.

Finally, to validate our findings, both in terms of the merge resolution patterns, as well as our tool evaluation, we will conduct a survey. This will give us a wide sample to evaluate the generalizability of our findings.

6 ACKNOWLEDGMENTS

The author is advised by Prof. Dr. Carlos Jensen and Prof. Dr. Anita Sarma. All of them are members of the Software Engineering, Human Computer Interaction and Programming Languages group at Oregon State University.

REFERENCES

[1] I. Ahmed, C. Brindescu, U. A. Mannan, C. Jensen, and A. Sarma. 2017. An Empirical Examination of the Relationship between Code Smells and Merge Conflicts. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 58–67. https://doi.org/10.1109/ESEM.2017.12 ISSN:.

[2] Sven Apel, Olaf Lessenich, and Christian Lengauer. 2012. Structured Merge with Auto-Tuning: Balancing Precision and Performance. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE 2012)*. ACM, Essen, Germany, 120–129. https://doi.org/10.1145/2351676.2351694

[3] Howard S Becker and Blanche Geer. 1957. Participant observation and interviewing: A comparison. *Human organization* 16, 3 (1957), 28–32.

[4] Jacob T. Biehl, Mary Czerwinski, Greg Smith, and George G. Robertson. 2007. FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. ACM, New York, NY, USA, 1313–1322. https://doi.org/10.1145/1240624.1240823

[5] David Binkley, Susan Horwitz, and Thomas Reps. 1995. Program Integration for Languages with Procedure Calls. *ACM Trans. Softw. Eng. Methodol.* 4, 1 (Jan. 1995), 3–35. https://doi.org/10.1145/201055.201056

[6] Markus Borg, Emil Alégroth, and Per Runeson. 2017. Software engineers' information seeking behavior in change impact analysis: an interview study. In *Proceedings of the 25th International Conference on Program Comprehension, ICPC 2017, Buenos Aires, Argentina, May 22-23, 2017*. 12–22. https://doi.org/10.1109/ICPC.2017.20

[7] Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. 2014. How Do Centralized and Distributed Version Control Systems Impact Software Changes?. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014)*. ACM, Hyderabad, India, 322–333. https://doi.org/10.1145/2568225.2568322

[8] 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

[9] Jonathan Corbet and Greg Kroah-Hartman. 2017. *2017 Linux Kernel Development Report*. Technical Report. The Linux Foundation.

[10] Isabella A. da Silva, Ping H. Chen, Christopher Van der Westhuizen, Roger M. Ripley, and André van der Hoek. 2006. Lighthouse: Coordination Through Emerging Design. In *Proceedings of the 2006 OOPSLA Workshop on Eclipse Technology eXchange (eclipse '06)*. ACM, New York, NY, USA, 11–15. https://doi.org/10.1145/1188835.1188838

[11] Danny Dig, Kashif Manzoor, Ralph E. Johnson, and Tien N. Nguyen. 2008. Effective Software Merging in the Presence of Object-Oriented Refactorings. *IEEE Trans. Softw. Eng.* 34, 3 (May 2008), 321–335.

[12] Scott D. Fleming, Chris Scaffidi, David Piorkowski, Margaret Burnett, Rachel Bellamy, Joseph Lawrance, and Irwin Kwan. [n. d.]. An Information Foraging Theory Perspective on Tools for Debugging, Refactoring, and Reuse Tasks. 22, 2 ([n. d.]), 1–41. https://doi.org/10.1145/2430545.2430551

[13] Anja Guzzi, Alberto Bacchelli, Yann Riche, and Arie van Deursen. 2015. Supporting Developers' Coordination in the IDE. ACM Press, 518–532. https://doi.org/10.1145/2675133.2675177

[14] 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 (ICSE '10)*. ACM, Cape Town, South Africa, 235–238. https://doi.org/10.1145/1810295.1810339

[15] Charles G. Hill, Maren Haag, Alannah Oleson, Christopher J. Mendez, Nicola Marsden, Anita Sarma, and Margaret M. Burnett. 2017. Gender-Inclusiveness Personas vs. Stereotyping: Can We Have it Both Ways?. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, May 06-11, 2017*. 6658–6671. https://doi.org/10.1145/3025453.3025609

[16] 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 (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 732–741.

[17] Ernst Lippe and Norbert van Oosterom. 1992. Operation-Based Merging. In *Proceedings of the Fifth ACM SIGSOFT Symposium on Software Development Environments (SDE 5)*. ACM, New York, NY, USA, 78–87. https://doi.org/10.1145/142868.143753

[18] Shane McKee, Nicholas Nelson, Anita Sarma, and Danny Dig. 2017. Software Practitioner Perspectives on Merge Conflicts and Resolutions. In *2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Shanghai, China, September 17-22, 2017*. 467–478. https://doi.org/10.1109/ICSME.2017.53

[19] C. Mendez, H. S. Padala, Z. Steine-Hanson, C. Hildebrand, A. Horvath, C. Hill, L. Simpson, N. Patil, A. Sarma, and M. Burnett. 2018. Open Source Barriers to Entry, Revisited: A Sociotechnical Perspective. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. 1004–1015.

[20] C. Mendez, A. Sarma, and M. Burnett. 2018. Gender in Open Source Software: What the Tools Tell. In *2018 IEEE/ACM 1st International Workshop on Gender Equality in Software Engineering (GE)*. 21–24.

[21] Sebastiano Panichella, Gerardo Canfora, Massimiliano Di Penta, and Rocco Oliveto. 2014. How the Evolution of Emerging Collaborations Relates to Code Changes: An Empirical Study. In *Proceedings of the 22Nd International Conference on Program Comprehension (ICPC 2014)*. ACM, New York, NY, USA, 177–188. https://doi.org/10.1145/2597008.2597145

[22] Sruti Srinivasa Ragavan, Bhargav Pandya, David Piorkowski, Charles Hill, Sandeep Kaur Kuttal, Anita Sarma, and Margaret Burnett. [n. d.]. PFIS-V: Modeling Foraging Behavior in the Presence of Variants. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017) *(CHI '17)*. ACM, 6232–6244. https://doi.org/10.1145/3025453.3025818

[23] Anita Sarma, Zahra Noroozi, and André Van Der Hoek. 2003. Palantir: Raising Awareness among Configuration Management Workspaces. 444–454.

[24] Anita Sarma, David Redmiles, and André van der Hoek. 2008. Empirical Evidence of the Benefits of Workspace Awareness in Software Configuration Management. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '08/FSE-16)*. ACM, New York, NY, USA, 113–123. https://doi.org/10.1145/1453101.1453118

[25] Yida Tao, Yingnong Dang, Tao Xie, Dongmei Zhang, and Sunghun Kim. 2012. How do software engineers understand code changes?: an exploratory study in industry. In *20th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-20), SIGSOFT/FSE'12, Cary, NC, USA - November 11 - 16, 2012*. 51. https://doi.org/10.1145/2393596.2393656

[26] Shaowei Wang and David Lo. 2014. Version History, Similar Report, and Structure: Putting Them Together for Improved Bug Localization. In *Proceedings of the 22Nd International Conference on Program Comprehension (ICPC 2014)*. ACM, New York, NY, USA, 53–63. https://doi.org/10.1145/2597008.2597148

[27] Bernhard Westfechtel. 1991. Structure-Oriented Merging of Revisions of Software Documents. In *Proceedings of the 3rd International Workshop on Software Configuration Management (SCM '91)*. ACM, New York, NY, USA, 68–79. https://doi.org/10.1145/111062.111071