Code Reviews Do Not Find Bugs

How the Current Code Review Best Practice Slows Us Down

Jacek Czerwonka, Michaela Greiler, Jack Tilford

Microsoft Corporation
Redmond, WA 98075

{ jacekcz, mgreiler, jtilford }@microsoft.com

***Abstract*—Because of its many uses and benefits, code reviews are a standard part of the modern software engineering workflow. Since they require involvement of people, code reviewing is often the longest part of the code integration activities. Using experience gained at Microsoft and with support of data, we posit (1) that code reviews often do not find functionality issues that should block a code submission; (2) that effective code reviews should be performed by people with specific set of skills; and (3) that the social aspect of code reviews cannot be ignored. We find that we need to be more sophisticated with our guidelines for the code review workflow. We show how our findings from code reviewing practice influence our code review tools at Microsoft. Finally, we assert that, due to its costs, code reviewing practice is a topic deserving to be better understood, systematized and applied to software engineering workflow with more precision than the best practice currently prescribes.**

***Index Terms*—Software engineering workflow, code reviews, code integration**

I. Introduction

In software engineering, we use code reviews for several reasons. Among them: to find defects, to ensure code's long-term maintainability, as a knowledge sharing tool, and to broadcast ongoing progress [3]. Additionally, in open-source, a flavor of a code review known as a pull request is used to agree on whether a code change is worthy of inclusion in the mainline source.

These are different purposes but the common thread is that code reviews allow a group of people to communicate over a shared view of an artefact undergoing a change. Because of their many uses and benefits, code reviews are a standard part of the modern software engineering workflow. Since they require heavy involvement of people, code reviewing is often the lengthiest part of code integration. The confluence of many goals in one activity does not make it easy to understand where code reviews are most beneficial and how to best inject them into the overall engineering workflow so that the time spent waiting for the opinions of others is always justified.

Keeping the above tension in mind, we ask: Do we currently use code reviews in the most efficient way or is it merely adequate? In what situations, do code reviews provide more value than others? What is the value of consistency of applying code reviews equally to all code changes?

II. Discussion

Modern code reviewing traces back its roots to the process of inspection [1]. Inspections were originally conceived as formal meetings, to which participants would prepare ahead of time. Unlike inspections, code reviews do not require participants to be in the same place nor do they happen at a fixed, prearranged time. Aligning with a distributed nature of many projects, code reviews are asynchronous and frequently supporting geographically distributed reviewers. Code review tools are now built with these characteristics in mind and are well-integrated in the modern engineering workflows.

With abundance of data coming from the engineering systems and having a diverse set of projects to observe [4], we have an opportunity to better understand the costs and benefits of the code review process.

Contrary to the often stated primary goal of code reviews, they often do not find functionality defects that should block a code submission. Only about 15% of comments provided by reviewers indicate a possible defect, much less a blocking defect. Rather, it is feedback related to the long-term code maintainability that comprises a much larger portion of comments provided by reviewers; at least 50% of all.

Code reviews take deliberation and are performed by people with a specific set of skills. The social aspect of code reviews cannot be ignored: people's roles on the team and their standing in team's hierarchy influence the outcome. Often it is not only the author of the change but also the reviewers who find themselves under scrutiny.

The usefulness of code review comments—as judged by the author of a code change—is positively correlated with reviewers' experience. Without prior exposure to the part of code base being reviewed, on average only 33% of any reviewer's comments are deemed useful by the author of a change. However, reviewers typically learn very fast. When reviewing the same part of code base for the third time, the usefulness ratio increases to about 67% of their comments. By the fourth time, it is equivalent to the project's long-term average.

Code review usefulness is negatively correlated with the size of a code review. That is, the more files there are in a single review, the lower the overall rate of useful feedback. The decrease however only starts to be noticeable for reviews with 20 or more changed files.

Modern code review process is expensive. Developers spend on average six hours a week reviewing changes of others [6]. Not only is it a significant effort in terms of time spent but also it forces the reviewer to switch context away from their current work.

The median time from a review being requested to receiving all necessary sign-offs is about 24 hours, with many lasting days if not weeks [5]. A long time in review causes process stalls and affects anyone who might be waiting to take a dependency on the new code. In addition, the longer the review time, the harder is for the author to switch back to the change and incorporate the feedback of the reviewers without potentially introducing new defects.

III. Conclusions

The above examples provide a view into the code review process and demonstrate our attempts to further understand it. The high cost of code reviews and reviewing having benefits that may not match the assumptions, often lead us to using them in our workflows in ways that are not efficient. We need to be more sophisticated with our guidelines surrounding the code review workflow. Our findings from code reviewing practice "in the very large" and studies conducted in collaboration with Microsoft Research, influence code review tools at Microsoft. Due to its cost and importance, this is a topic deserving to be better understood, systematized and applied to software with more precision than the best practice and research currently prescribes.

With this talk, the attendees from industry receive a detailed experience report on benefits and costs of reviewing practices and an overview of supporting tools used at Microsoft. Attendees from academia get an overview of how research findings and data-driven software engineering techniques shape the tooling and practices landscape in industry, as well as which open issues remain unanswered. ·

IV. Speaker

Jacek Czerwonka is a software engineering manager on the Tools for Software Engineers team at Microsoft. His work focuses on creating systems increasing code development velocity and tools for software verification. His team, which includes the co-authors, works on understanding of software engineering organizations and improving engineering processes at Microsoft. His interests revolve around data-driven decision making on software projects, engineering process measurement, process improvement, software testing and quality assurance.

References

[1] Lawrence G. Votta, Jr.. 1993. Does every inspection need a meeting?. *SIGSOFT Softw. Eng. Notes* 18, 5 (December 1993), 107-114.

[2] Mika V. Mantyla and Casper Lassenius. 2009. What Types of Defects Are Really Discovered in Code Reviews?. *IEEE Trans. Software Eng.* 35, 3 (May 2009), 430-448

[3] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering* (ICSE '13). IEEE Press, Piscataway, NJ, USA, 712-721.

[4] Jacek Czerwonka, Nachiappan Nagappan, Wolfram Schulte, and Brendan Murphy. 2013. CODEMINE: Building a Software Development Data Analytics Platform at Microsoft. *IEEE Software* 30, 4 (July 2013), 64-71.

[5] Peter C. Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (ESEC/FSE 2013). ACM, New York, NY, USA, 202-212.

[6] Amiangshu Bosu and Jeffrey Carver. 2013. Impact of Peer Code Review on Peer Impression Formation: A Survey. *Proceedings of the 7th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (ESEM), 2013. Baltimore, MD, USA, 133-142.