Summary of Investigating code review quality: Do people and participation matter?

Zhaoyi Ge

Problem Being Solved

Low quality code reviews may miss bugs in the code changes. This paper investigates whether personal and participation factors affect the quality of code review.

New Idea

This paper studied the quality of code review process at the Mozilla project. The paper extracted commits from Mozilla's version control system, and then linked the bug ID in each commit to the corresponding bug in Bugzilla, the issue tracking system. The paper applied the SZZ algorithm and tools to identify the bug-inducing commit.

The paper selected a handful of technical, personal and participation factors. A multiple linear regression (MLR) model is applied, where the factors are explanatory variables and code review quality (buggy or not) is the response variable.

Results from the model showed that more than half of the code reviews are buggy. Moreover, some personal factors such as reviews experience has a positive correlation with code quality, and some participation factors (number of commenting developers) are positively correlated with code quality while some (number of developers on CC etc.) are negatively correlated.

Positive Points

This paper employed a simple yet comprehensive empirical methodology. The authors clearly outline each step of their data collection and analysis pipeline. Extracting commits and linking them with Bugzilla review data and applying the SZZ algorithm. The process is straightforward and transparent. The author also applied transformations and filters on the data. and designed a set of heuristics to reliably link commits to correct reviewed patches.These actions greatly minimized the threats to validity.

The paper's findings provide insights for almost every developer. Human factors are challenges that almost every development team faces. Whether in open source, industry, or academia, these insights are directly applicable. The research reminds us of human factors, which is something we commonly miss when thinking about code reviews.

Negative Points

One limitation of this paper is its reliance on multiple linear regression (MLR) as the only analysis method. It's a very simple statistical model, is it really a good model for this scenario? MLR assumes linearity and independence among factors, which may not accurately capture the complex interactions between technical, personal, and social aspects of code review.

The paper did not give clear, actionable items. While the authors identify several statistically significant factors, the paper doesn't translate these findings into practical guidance for improving real-world review processes. It's not clear what developers or project maintainers are supposed to do with this information. The results feel somewhat abstract and disconnected from actionable improvements in software engineering workflows.

Future Work

Code review is a complex process involving personal and social aspects . While the quantitative model in this paper showed strong correlation between several factors and code review quality, the numbers alone can't fully explain the human dynamics behind missed bugs. As future work, there can be investigations aimed to capture the social context, communication patterns, and interpersonal dynamics that influence review quality.

Rating

3.5/5. It's a very approachable paper for people outside of SE.

Discussion

- What are your explanations for the two surprising factors? Do you agree with the author or not?
- What are your takeaways from this paper? Would you do anything differently when doing code reviews after this paper?

- What are some of the factors that go beyond technical, people and participation? (Company-wise, Cultural, Social)

The professor talked about a professor at UofT who was involved with Mozilla Foundation (separate from the developers at Mozilla Project), who is concerned with publicising Mozilla projects as OSS (open source software) because of the nature of the Mozilla project.

The professor talked about the SZZ algorithm (founder Tom Zimmerman and advisor Andreas Zeller), a straightforward algorithm that finds when a bug is introduced to the codebase; There exists a subarea of improvements on the algorithm and many research papers on this topic.

The professor talked about the issue of distinguishing the science and engineering sides of SE research. This scientific research does not need to be actionable for engineers. The concern is that research should help us learn what to do next, but this is not necessarily required.

Youssef agreed that cultural factors played a role in code review quality. Felix talked about cultural and communication factors as well.

The professor pointed out that code reviews serve purposes other than catching bugs. Code reviews have been used to teach and share expertise with new hires in a human way. This helps them learn the culture and code review practices.

Asim talked about the human interaction element, where developers are not really excited about doing code reviews; it's just a job for them. So working on a good UI and tool that provides all information easy to access, and helps developers complete code reviews quickly, would make a great impact on the code review quality.

Amaan talked about the reviewer queue length factors and how developers have different workloads where a heavier workload may rush reviewers to complete the tasks which could impact the quality of code reviews. So the idea is to develop some AI tools to optimize the workload assignment across the developers more efficiently and effectively across developers of varying skills and levels of experience. The presenter agreed but posed a question of when a reviewer doesn’t like to do a review. Amaan suggested incorporating reviewer preference in AI tools to better assign tasks to reviewers