Summary Review: Investigating Code Review Quality: Do People and Participation Matter?

Eimaan Saqib
e2saqib@uwaterloo.ca
University of Waterloo
Waterloo, Ontario, Canada

ACM Reference Format:
Eimaan Saqib. 2018. Summary Review: Investigating Code Review Quality: Do People and Participation Matter?. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 PAPER SUMMARY

The paper code review quality in terms in the Mozilla codebase. Code review is a critical quality assurance (QA) practice in software development. It involves peers inspecting code changes (patches) before they're merged into the main codebase. But sometimes, bugs still slip through.

Mozilla uses a two-tiered review process for submitted patches. First, a review is performed by the owner of the module in question. If the patch requires integration or modifies the core Mozilla infrastructure, then it is reviewed by a super reviewer. Every patch is reviewed using the Bugzilla issue-tracking system. If it is approved, it is committed to the main codebase.

The authors first extracted 44k+ commits from mozilla-central. They collected different metrics for each commit including the commit size, review flag, and the commit message. Then they used pattern matching to extract bug IDs from the commit messages that were fixing some bug and eliminated the ones without review tags. They used the bug ID to get different metrics from Bugzilla including when the bug was submitted, a list of proposed patches, review flags of the patches, etc. They were unable to extract information for some bugs that required special permissions for access. They linked each commit to its corresponding patch and review-related information. Then they used the SZZ algorithm to build a list of revisions that are candidates for bug-inducing changes. They also extracted some information to serve as technical, personal, and participation factors in code reviews. They created an MLR model with these explanatory variables that predicts whether or not a bug is detected in a review.

Findings indicated that more than half the reviews are buggy. Longer review queue, larger size of the commit and number of files modified, more reviewer comments, and more people on the CC list all lead to lower quality reviews that are buggy. On the other hand, super reviews, greater reviewer experience, number of commenting developers, and comments per developer all lead to better quality and less buggy reviews.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

2 POSITIVE POINTS

(1) The SZZ algorithm is a widely accepted method for identifying bug-inducing changes. While imperfect, its application was transparent, and the authors acknowledged its 9% false-positive rate, manually validating a subset of results.
(2) Controlling for technical factors (e.g., patch size) isolated the impact of human/social variables.
(3) Variables with high VIF scores (e.g., overlapping metrics like "number of comments" vs. "number of commenting developers") were systematically removed, improving model reliability.
(4) The authors openly acknowledged threats (e.g., Bugzilla's ambiguous CC list purpose, SZZ's false positives) and mitigated them where possible (e.g., manual validation of bug links).

3 NEGATIVE POINTS

(1) SZZ links bug-fixing commits to earlier changes, but Mozilla's long history means some bug-inducing commits might predate the study window (2013–2014), leading to undercounting.
(2) The study focused solely on Mozilla, an open-source project with unique practices (e.g., super reviews). Findings may not apply to closed-source projects or smaller teams with less formal processes.
(3) 188 bugs were excluded due to restricted access, introducing potential selection bias (e.g., security-critical bugs might be overrepresented in restricted reports)
(4) Reviewer/writer experience was measured as total reviews or patches submitted, which oversimplifies expertise. A developer with 100 trivial patch reviews may not have the same expertise as one with 50 complex ones.
(5) Adjusted R^2 values (12.8–17.3%) indicate the models explain only a fraction of variance. Unmeasured factors—like code complexity, reviewer motivation, or time pressure—likely play significant roles.
(6) The counterintuitive link between reviewer comments and bugs (more comments → more bugs) wasn't fully explained. Is this due to contentious reviews, unclear feedback, or poorly written patches? The study flags this but leaves it unresolved.

4 FUTURE WORK

(1) Conduct interviews/surveys with Mozilla developers to understand why reviewers miss bugs. Do time pressures or unclear guidelines contribute? How do reviewers prioritize depth vs. speed?
(2) Explore the social dynamics behind metrics like "number of reviewer comments": Are contentious reviews or unclear feedback causing more bugs?
(3) Replicate the study in closed-source projects (e.g., Microsoft, Google) to compare with Mozilla's open-source dynamics, and smaller teams/startups to assess scalability of findings.
(4) Investigate whether AI tools (e.g., ChatGPT for code analysis, static analyzers) reduce human oversight gaps. Do they miss bugs that humans catch, or vice versa?
(5) Study how time-of-day, deadlines, or sprint cycles impact review quality.

5 RATING

4/5

The study advances understanding of human/social factors in code review but leaves much variance unexplained, urging future work to explore additional drivers of review quality.

6 DISCUSSION POINTS

(1) Is a 54% defect rate an inevitable trade-off for rapid development, or a sign of systemic failure? Should Mozilla slow down reviews to catch more bugs, risking slower innovation? What if competitors (e.g., Chrome) move faster? If a missed bug causes a data breach in Firefox, who bears responsibility: the reviewer, the developer, or Mozilla's process? Should "acceptable" bug rates depend on the software's domain? (E.g., a video game vs. medical device software?)
   - Finding bugs is hard and complicated. No matter what you do, you need to expect bugs will get through. But on the other hand, you can iterate faster on the bug you found if the review process is faster. There is a trade-off but it's worth it.
   - Is 54% a bad or good number? There is always a snowball effect. Once a review is performed where the bug was missed, you typically don't review that patch again very thoroughly.
   - The SZZ algorithm only looks at removed lines, not added lines. But added lines need to be considered as well to have a complete picture of bug-inducing changes.
(2) The study found that adding more people to a bug's CC list increased missed bugs, while active commenting reduced them. Does CC'ing large groups create a "diffusion of responsibility," where everyone assumes someone else will review? How can teams encourage meaningful participation without overwhelming developers with notifications? Is 'more participation' always better, or does it risk creating noise without value?
   - A downside of this paper is the overall lack of qualitative insights. It would be useful if researchers talked to some developers to gain these insights.
(3) Mozilla uses "super reviews" by 30 experts for critical changes, which reduced bugs in the study. Can super reviews work in startups or smaller teams without dedicated experts? If it only explains a small percentage of bugs, should teams focus on it or hunt for bigger factors (e.g., code complexity)?
   - For smaller teams, you need experts to work on developing the projects. Super review is not sustainable and affordable. It would be beneficial for smaller teams to have workshops or seminars to let experts share their thoughts and guidance with other reviewers. Limiting the number of people who work on the critical development tasks is not viable.
   - A possible solution could be one expert in a small team acting as a super reviewer, and the rest are developers. The expert spends three days developing, and two days on super reviews.
   - Eventually you need something like super reviews for large patches, and also people reviewing things if it requires integration among different modules.