# INVESTIGATING CODE REVIEW QUALITY: DO PEOPLE AND PARTICIPATION MATTER?

OLEKSII KONONENKO, OLGA BAYSAL, LATIFA GUERROUJ, YAXIN CAO, AND MICHAEL W. GODFREY

## BACKGROUND

CODE REVIEW

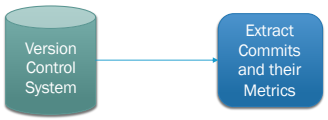
- Code Review:
  - Critical quality assurance practice
  - Catch defects, ensure design consistency, share knowledge
- Mozilla:
  - Large mature open-source project
  - Strict code review policies
  - Higher contributor diversity
- Prior work focused on code review coverage and time, not quality

## STUDY SETUP

- **Timeframe:** January 2013 – January 2014
- **Modules analyzed:** Mozilla-all (entire codebase), plus three largest modules: Core, Firefox, Firefox for Android
- Data Sources:
  - **Version Control (Mercurial)**: 44,595 commits initially, filtered to 27,270 after preprocessing
  - **Bugzilla**: Linked bug reports, patches, and review history

## METHODOLOGY

- Extracted 44,594 commits
- Collected information: identifier, name/email, date added, description, size
- To calculate size, exclude changes to binary files

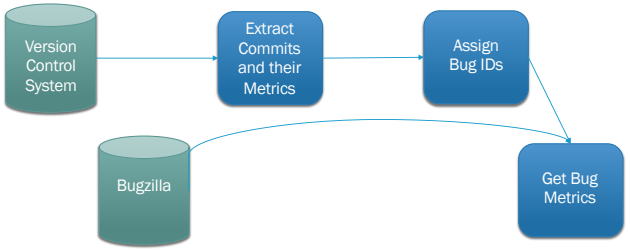


## METHODOLOGY

- Detect changes that fix bugs
- Extract bug ID from commit summary using regex
- Eliminate matches with unreviewed commits
- Assigned bug IDs to 35,668 commits

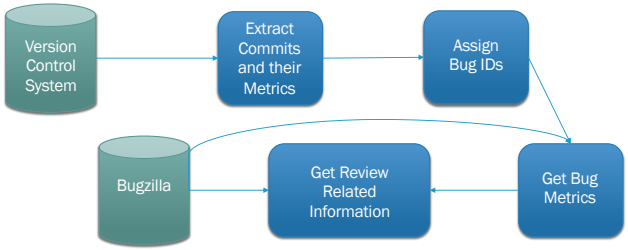

## METHODOLOGY

- Metrics: date of submission, name/email, severity and priority, module affected, proposed patches
- Patches: author, submission date, review flags
- Review flags: date and time, email




## METHODOLOGY

- Linked each commit to patch and review related information
- Matched with newest patch that had a positive review and review was granted before commit time




## METHODOLOGY

- Execute difference between bug fix commit and parent
- Consider only removed lines
- Get identifier of revision that modified bug-inducing lines
- 9% false positives

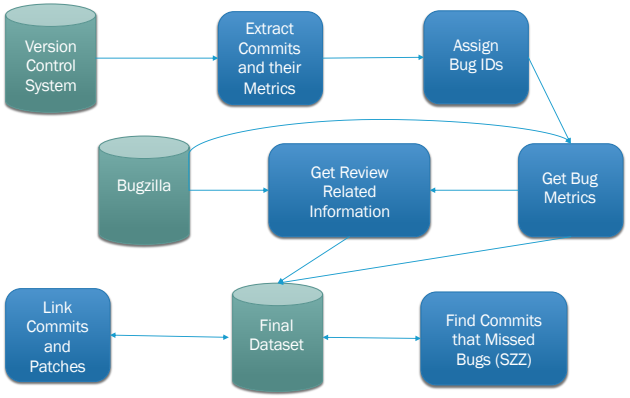


TABLE II: A taxonomy of considered technical, personal, and participation metrics used.

| Type | Metric | Description | Rationale |
|---|---|---|---|
| Technical | Size (LOC) | The total number of added and removed lines of code. | Large commits are more likely to be bug-prone [15]; thus the intuition is it is easier for reviewers to miss problems in large code changes. |
| | Chunks | The total number of isolated places (as defined by `diff`) inside the file(s) where the changes were made. | We hypothesize that reviewers are more likely to miss bugs if the change is divided into multiple isolated places in a file. |
| | Number of files | The number of modified files. | Similar, reviews are more likely to be prone to bugs if the change spread across multiple files. |
| | Module | The name of the Mozilla module (e.g., Firefox). | Reviews of changes within certain modules are more likely to be prone to bugs. |
| | Priority | Assigned urgency of resolving a bug. | Our intuition is that patches with higher priority are more likely to be rushed in and thus be more bug-prone than patches with lower priority levels. |
| | Severity | Assigned extend to which a bug may affect the system. | We think that changes with higher levels of severity introduce less bugs because they are often reviewed by more experienced developers or by multiple reviewers. |
| | Super review | Indicator of whether the change required super review or not | Super review is required when changes affect core infrastructure of the code and, thus, more likely to be bug-prone. |
| | Number of previous patches | The number of patches submitted before the current one on a bug. | Developers can collaborate on resolving bugs by submitting improved versions of previously rejected patches. |
| | Number of writer's previous patches | The number of previous patches submitted by the current patch owner on a bug. | A developer can continue working on a bug resolution and submit several versions of the patch, or so called resubmits of the same patch, to address reviewers concerns. |
| Personal | Review queue | The number of pending review requests. | While our previous research [14] demonstrated that review loads are weakly correlated with review time and outcome; we were interested to find out whether reviewer work loads affect code review quality. |
| | Reviewer experience | The overall number of completed reviews. | We expect that reviewers with high overall expertise are less likely to miss a bug. |
| | Reviewer experience for module | The number of completed reviews by a developer for a module. | Reviewers with high reviewing experience in a certain module are less likely to miss defects; and on the contrary, reviewers with no past experience in performing code reviews for some modules are more likely to fail to catch bugs. |
| | Writer experience | The overall number of submitted patches. | Developers who contribute a lot to the project — have high expertise — are less likely to submit buggy changes. |
| | Writer experience for module | The number of submitted patches for a module. | Developers who make few changes to a module are more likely to submit buggy patches. |
| Participation | Number of developers on CC | The number of developers on the CC list at the moment of review decision. | Linus's law states that "given enough eyeballs, all bugs are shallow" [16]. |
| | Number of comments | The number of comments on a bug. | The more discussion happens on a bug, the better the quality of the code changes [5]. |
| | Number of commenting developers | The number of developers participating in the discussion around code changes. | The more people are involved in discussing bugs, the higher software quality [5]. |
| | Average number of comments per developer | The ratio of the comment count over the developer count. | Does the number of comments per developer has an impact on review quality? |
| | Number of reviewer comments | The number of comments made by a reviewer. | Does reviewer participation in the bug discussion influence the quality of reviews? |
| | Number of writer comments | The number of comments made by a patch writer. | Does patch writer involvement in the bug discussion affect review quality? |

## DETERMINING EXPLANATORY FACTORS

## MODEL CONSTRUCTION AND ANALYSIS

- Used an MLR model
- Transformed categorical variables
- Identified collinearity
- Evaluation using R-squared

TABLE IV: Number of code reviews that missed bugs.

| System | # Reviews | # Buggy Reviews | % Buggy Reviews |
|---|---|---|---|
| Mozilla-all | 28,127 | 15,188 | 54.0 % |
| Core | 18,759 | 10,184 | 54.3 % |
| Firefox | 2,668 | 1,447 | 54.2 % |
| Firefox4Android | 2,160 | 1,210 | 56.0 % |

## DO CODE REVIEWERS MISS MANY BUGS?

## DO PERSONAL FACTORS AFFECT THE QUALITY OF CODE REVIEW?

(a) Technical and personal factors.

| | Mozilla | Core | Firefox | FF4A |
|---|---|---|---|---|
| Adjusted $R^2$ | 0.128 | 0.123 | 0.173 | 0.138 |
| Size (LOC) | 0.102*** | 0.098 *** | 0.108*** | 0.115*** |
| Chunks | † | † | † | † |
| Number of files | 0.058*** | 0.059 *** | 0.109*** | 0.062* |
| Module | ⋆ | n/a | n/a | n/a |
| Priority | ⋆ | ⋆ | ‡ | · |
| Severity | ‡ | ‡ | · | ‡ |
| Super review | -0.139** | -0.177*** | · | n/a |
| Review queue | 0.017*** | 0.0204*** | 0.038** | 0.045** |
| Reviewer exp. | -0.013*** | -0.012*** | -0.029*** | -0.041*** |
| Reviewer exp. (mod.) | † | † | ‡ | 0.018* |
| Writer exp. | · | -0.004* | ‡ | ‡ |
| Writer exp. (module) | † | † | ‡ | · |
| # prev patches | † | † | † | -0.045*** |
| # writer patches | -0.012*** | · | · | † |

†Disregarded during VIF analysis (VIF coefficient $> 5$).
⋆ "It's complicated": for categorical variables see explanation of the results in Section IV.
FF4A = Firefox for Android.

# DOES PARTICIPATION IN CODE REVIEW INFLUENCE ITS QUALITY?

(b) Technical and participation metrics.

| | Mozilla | Core | Firefox | FF4A |
|---|---|---|---|---|
| Adjusted $R^2$ | 0.134 | 0.128 | 0.173 | 0.147 |
| Size (LOC) | 0.105*** | 0.103*** | 0.105*** | 0.117*** |
| Chunks | † | † | † | † |
| Number of files | 0.060*** | 0.059*** | 0.090*** | 0.067*** |
| Module | ⋆ | n/a | n/a | n/a |
| Priority | ‡ | ⋆ | ‡ | ⋆ |
| Severity | ⋆ | ‡ | ⋆ | ‡ |
| Super review | -0.124*** | -0.160*** | ‡ | n/a |
| # of devs on CC | 0.053*** | 0.056*** | ‡ | 0.049* |
| # comments | † | † | † | † |
| # commenting devs | -0.124*** | -0.102*** | -0.075*** | -0.176*** |
| # comments/ # dev | -0.039*** | -0.029** | ‡ | ‡ |
| # reviewer comments | 0.010** | ‡ | 0.026* | ‡ |
| # writer comments | · | · | · | -0.047** |

‡Disregarded during stepwise selection.
Statistical significance: '***' $p < 0.001$; '**' $p < 0.01$; '*' $p < 0.05$; '·' $p \geq 0.05$.

## POSITIVE POINTS

- 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.
- **Multiple Linear Regression (MLR)**: The use of MLR collinearity checks (VIF < 5) strengthened the statistical rigor. Controlling for technical factors (e.g., patch size) isolated the impact of human/social variables.
- Variables with high VIF scores (e.g., overlapping metrics like "number of comments" vs. "number of commenting developers") were systematically removed, improving model reliability.
- 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).

## NEGATIVE POINTS

- 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.
- 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.
- 188 bugs were excluded due to restricted access, introducing potential selection bias (e.g., security-critical bugs might be overrepresented in restricted reports)
- Reviewer/writer experience was measured as *total reviews/patches submitted*, which oversimplifies expertise. A developer with 100 trivial patch reviews may not have the same expertise as one with 50 complex ones.
- 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.
- 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.

## FUTURE WORK

- 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?
- Explore the **social dynamics** behind metrics like "number of reviewer comments": Are contentious reviews or unclear feedback causing more bugs?
- 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.
- 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?
- Study how time-of-day, deadlines, or sprint cycles impact review quality.

## RATING

4/5

## DISCUSSION POINTS

- 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?)
- 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?
- 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)?