Modern Code Review: A Case Study at Google

10/8/25

Presented by

Youssef Souati

UNIVERSITY OF WATERLOO

Paper background

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko at Google; Alberto Bacchelli at the University of Zurich.

Published at the International Conference on Software Engineering, Software Engineering in Practice track, Gothenburg, Sweden, May 27 to June 3, 2018.

Exploratory study using 12 interviews, a survey with 44 respondents, and logs of about nine million reviewed changes to examine motivations, practice, and developer experience at Google.

UNIVERSITY OF WATERLOO

What is modern code review

Before this, classic code inspections were formal, scheduled, and synchronous, with planning, preparation, an in-person inspection meeting, rework, and follow-up.

Email based reviews in open source were asynchronous but unstructured.

Modern code review is informal and tool-based. It is asynchronous. It focuses on small diffs tied to a change. Discussion happens as line-level threaded comments and one or more approvers sign off in the tool.

UNIVERSITY OF WATERLOO

Why Google

Google has required code review since early in its history and refined it for over a decade. Most development happens in a single monorepo with a uniform review workflow in Critique.

The scale is huge. About 20,000 code changes each workday. More than 25,000 authors and reviewers. Dozens of offices around the world.

UNIVERSITY OF WATERLOO

Methodology

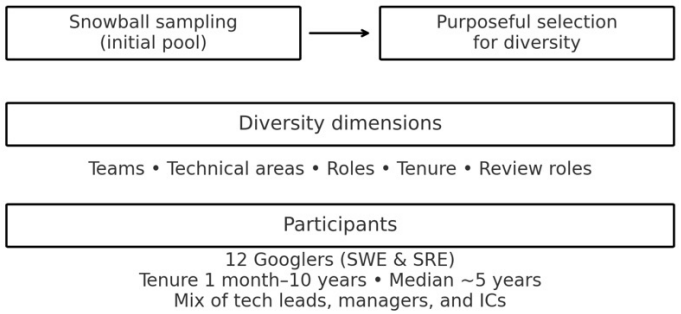
- Mixed-methods design using three lenses.
  - Semi-structured interviews
  - Critique tool logs
  - A short targeted survey on a fresh review
- Direct comparison to Rigby and Bird's convergent practices. Quantitative analysis centers on flow, speed and frequency, change size, and number of reviewers to check whether Google shows the same patterns.

Research Questions

- RQ1: motivations for review at Google.
- RQ2: how the process runs in practice at scale.
- RQ3: how developers perceive review and where it breaks.

- They complement Rigby and Bird's cross-project work by doing a focused, deep case study inside one company.
- Rigby and Bird had identified five convergent practices.
  - Lightweight flexible process. (CP1)
  - Early, quick, frequent reviews. (CP2)
  - Small changes. (CP3)
  - Two reviewers often optimal. (CP4)
  - Review as group problem solving. (CP5)

Research Method: Interviews



Research Method: Critique tool logs

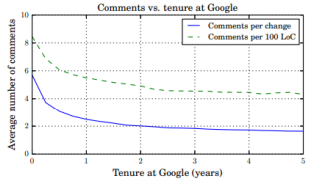
- Scope: Critique logs; only changes with ≥1 approver (here, "reviewer" = approver).
- Scale & time: Jan 2014–Jul 2016; ~9M changes, ~13M comments, 25k+ devs; ~20k changes/day.
- Clean + metrics: Main codebase; exclude uncommitted/zero-diff; filter bots; measure latency (first response/approval), size (lines/files), reviewers/comments, iterations.

Research Method: Survey

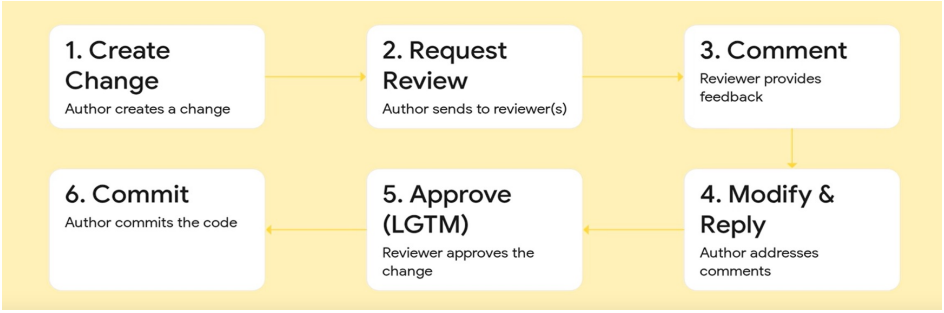
- Targeting: 98 engineers, each about one just-reviewed change to cut recall bias.
- Instrument: 3 Likert value questions, 1 effects multiple-choice with “other,” plus 1 open-ended.
- Responses: 44 valid replies (45% response rate).
- Role: Triangulate interview themes and capture immediate perceptions.

Results: RQ1

- Origin: enforce clarity; code should teach future engineers; avoid single-owner knowledge
- Today (finding 1): Review focuses on readability and maintainability; defect finding is welcomed but not the only focus.
- Philosophy: education + shared ownership as the core purpose
  - (finding 2) Expectations for a review depend on the work relationship between author and reviewers.



Results: RQ2- How review runs



(finding 3) Process is lightweight and flexible; ownership and readability are explicit; tool includes reviewer recommendation and code analysis.

Results: RQ2- Key metrics at scale

- Typical size: median ~24 lines; ~90% of changes touch <10 files
- Speed: first response often <1 hour for small changes; median time-to-approval <4 hours
- Participation: median 1 approver; <25% have >1
- Iteration: >80% of reviews finish in a single address-and-resend round

(finding 4) Reviews are markedly faster with smaller changes than prior studies; one reviewer is often sufficient vs. two elsewhere.

Results: RQ3- Breakdowns developers hit

- Distance: geo and org distance cause delays and misunderstandings.
- Social interactions: tone issues and power plays can sour reviews; negative-tone comments are less useful.
- Tool customization: team-specific policies not always supported; new option added to require all reviewers to sign off.

(finding 5) Despite years of refinement, code review still faces breakdowns rooted in complex interactions.

Results: RQ3- Satisfaction and time

(finding 5) Developers still consider review highly valuable and spend about three hours per week reviewing.

Positive points

- Culture + process + tooling as one coherent system. The paper shows how readability and ownership ideals become explicit rules and then become Critique features like reviewer suggestion and inline analyzers, making the workflow fast but accountable.
- Practical details teams can copy. Pre-commit gates, analyzer feedback with “Please fix” and “Not useful,” and lightweight iteration policies are described clearly enough to inform policy and tool decisions elsewhere.
- When they set out to measure review time, they didn’t take the easy self-report route. They instrumented Critique, grouped events into review sessions with a 10 minute inactivity gap over five weeks, and got an objective estimate of about 3.2 hours per week on average and 2.6 median (Low compared to the 6.4 self-reported for OSS projects).

Negative points

- Unclear interview geography. The paper does not state where interviewees were located. Experiences can vary by region, especially socially, so important breakdowns in other offices may be missed.
- Limited value from the survey. After the interviews, a small targeted survey adds little. It would have been more useful with people the team could not meet in person, especially in other locations. The authors say the survey adds confidence in the coded themes, but the options were those themes, which likely biased responses.
- Satisfaction measured at the wrong level. Reporting that 97% are satisfied with “code review” is unsurprising. Satisfaction should be measured for the process itself, such as turnaround time, clarity of feedback, tool support, and friction points.

Future work

- Run a focused study: follow new developers from day one to see how the cold start affects the findings of this paper.
- Smoother integration: start with co-reviewing changes alongside owners to build understanding and ownership of specific directories, then graduate to authoring small changes.
- Guardrails: flag large first diffs and suggest splitting into smaller changes.
- Measure impact: time to first response, number of iterations, new-dev satisfaction, and time to first owner-level approval.

Rating

I would give this paper a 4 out of 5. It offers a clear, data-backed playbook for modern code review that most teams can adopt today.

Discussion points

- Is Google's one-approver norm (less than other companies) mainly viable because ownership and readability create clear accountability, and would it still hold in teams without those guardrails or in tightly coupled code?
- The paper suggests very large diffs often include mass deletes but they didn't mention why; why should these be handled differently?
- Would an employee's geographic location and time zone meaningfully affect their satisfaction?
- Could LLMs automate enough to allow guarded auto-merge for narrow, well-tested changes, or will human reviewers always be required for design, security, and accountability?