CS846 Week 9: Modern Code Review: A Case Study At Google

Mohammad Jaffer Iqbal

Problem Being Solved:

This paper highlights the gap in literature pertaining to a focused and longitudinal perspective on modern code review practice. A major study in 2013 focused on converging code review practices across different contexts (open-sourced and proprietary systems using async and tool-based processes) and found them to converge on the following properties: (i) the code review process being lightweight and quick, (ii) the process requiring few reviewers (optimally 2) and the purpose of the reviews being a group problem solving activity. Amongst these works, there was a gap pertaining to a focused study. To this end, the authors chose to investigate the practice in Google to see if they are in accordance with the converging properties. Google, being the tech giant that it is, was chosen as it was one of the early adopters of code review and performs plenty of reviews daily (with 25000 developers making 20000 source code changes daily).

New Idea:

To address the knowledge gap, the paper splits its objective into 3 research questions: (i) What are the motivations for code review at Google, (ii) What is the code review practice at Google (in terms of form, size, frequency, etc) and (iii) How do developers at Google perceive these code reviews? To address RQ1, the authors conduct 12 extensive interviews with developers at Google. These interviewees were chosen using a Snowball sampling approach while ensuring diversity to keep results generalizable. An open coding approach followed by a separate closed coding session to identify themes from the interviews. To address RQ2, the authors perform a quantitative study spanning 9M reviews on CRITIQUE (Google's in house tool for code reviews). Finally, to address RQ3, the authors include insights from the interviews and use a survey (44 participants) asking their thoughts about a review on their recent specific code changes (to mitigate recall bias).

Their results show some key differences in comparison to the 2013 study. Specifically, the goal for performing code review at Google was not to be a problem-solving exercise, but to ensure code readability, maintainability and its educational value. These goals also shifted in the case of different positions – in the case of a higher-ranking reviewer and a lower ranking code change author, the goal of the review would primarily be educational value. The process was also lightweight and flexible, while but was much quicker and had smaller changes as compared to the results in the 2013 study. Google preferred to have 1 reviewer for most reviews (instead of 2 or more). The most interesting insights pertained to communication issues leading to the breakdown of the code review process. These were distance (geographical and teams), tone (harsh/ non-constructive criticism in reviews), power (dragging out reviews or withholding approvals to intimidate), mismatched expectations and lack of context. Overall, the study found that 97% of the survey participants agreed with the usefulness of the code review process.

Positive Points:

The paper is comprehensive in its investigation as it combines its extensive qualitative insights (from interviews and surveys) with quantitative insights (from the 9M CRITIQUE logs). The authors clearly state the various biases that may result from their methodology and take measures to mitigate them. The paper also paints a complete picture of the internals of code review at Google, how they have a monolithic repository with owned directories and how CRITIQUE and its automated test cases integrate with its pipeline. The insights about the specific communication issues leading to breakdowns are very realistic and practical. Finally, the methodology for interviews (4 interviewers containing one scribe and two Google employees followed by open and closed coding to identify themes) is well thought out.

Negative Points:

Despite having access to the extensive CRITIQUE dataset, the paper provides no qualitative insights about the efficacy (how many bugs it catches and overlooks) of the fast-paced code review process at Google. The paper mentions how its objective is to perform a "focused and longitudinal study", but the longitudinal aspect of their objective is left un-addressed, with the survey and interview questions being primarily about the current practices and the thought behind them. Even the qualitative insights span 2 years, which do not provide any information about the evolution of the code review process at Google over a longer period (e.g. a decade). Finally, the paper mentions how it uses its surveys to counter the self-selection bias that results from voluntary interviews – however, the survey itself is voluntary as well, so this justification does not make sense.

Future Work:

One direction could be to solve the communication challenges that result in the code review process breakdown. Specifically, an LLM based solution could review code changes to provide the author with insights about where to provide context, while also reviewing the code reviews themselves to make them more descriptive and constructive, while removing harsh commentary. Furthermore, the CRITIQUE dataset could be expanded to span 5 years, and an analysis could be performed to correlate Google's review practices (review comment types, size, frequency) with long-term code quality metrics (e.g. bugs overlooked, maintainability).

Rating:

4/5 – the paper paints a thorough picture of the internals and expectations of code review at Google.

Discussion Points:

(i) What are the pros and cons for using Snowball sampling for interviews? Could we make do with random sampling?

Snowball sampling is particularly applicable in this senario as it allows you to first interview the people who are more likely to answer your interview questions. In the context of the paper, the authors interview one of the very early employees at Google to understand the motivations for adopting modern code review practices. It also helps to find initial willing participants through contacts first. However, there is a tradeoff when it comes to using random sampling. While such an approach garuntees better generalizability, it is not the best approach when trying to get your questions answered that only specific personel can. For example, a random sample could have sampled only junior to mid-level developers who would have lesser insights about the introduction of the modern code review proces.

(ii) Can having questions on recent code changes (in the survey) lead to a recency bias?

While the paper adds questions on recent code changes in the survey to mitigate recall bias, this strategy can introduce recency bias, where the survey respondants' answered may be skewed by the usefulness of the recent review. This too shows a tradeoff.

(iii) Could different insights be gained from specialized code reviews (security/performance teams)?

Yes, for example, the security team could provide insights on a more rigorously qualitative review with respect to best and safe coding practices, while a performance team could have more benchmark/scalability centric code reviews. There quantity and size could also vary across these specialized teams. In critical cases, "super reviews" could be helpful, ensuring stringent checks are perform where required.

(iv) Is breaking down changes into smaller more isolated changes (to facillitate quick reviewing) always helpful?

While Google's strategy helps in maintaining a fast-paced review cycle, it also means that larger code changes when split into smaller, more isolated, changes can have the potential of removing context thus requiring more back and forth tallying with previously completed reviews.