Modern Code Review

A case study at Google

Presented by Zachary Hayden

Introduction to the Paper

Questions of focus:

1. Why does Google need code review?
2. What does a code review at Google consist of?
3. What are the perceptions of code review by Google developers?

Research Methods:

- Semi-structured interviews with Google developers
- Logs from Google's internal code review tool: *CRITIQUE*
- Survey to non-developing Google employees

Goals:

- Identify which features and practices in modern code review are most effective and why

The State of Code Review: The Old vs. the New

Old

Peer code review originally put reviewers and the author together to reason about the application logic and primarily identify defects. This approach was formalized in 1976 [4], called code inspections, and was successful in improving the quality of software projects [2,3].

Although successful, this approach came with drawbacks. It was a time consuming and labor intensive task because it involved multiple reviewers going through the codebase line-by-line, and due to its synchronous and highly structured nature, required a lot of logistical coordination among development teams and management.

There had to be a better way…

New

The modern approach for code review utilizes tooling to automate tasks and create organized asynchronous workflows for improved efficiency.

A common example that has been demonstrated in class is git pull requests wherein entire teams are able to asynchronously review and comment about source code before it is ultimately merged to production. For this purpose Google uses Critique, an all encompassing collaborative code review tool. This allows for a more streamlined flow and focuses on reviewing code changes rather than the codebase in its entirety, allowing for improved efficiency.

Motivation

Google initiated in code review processes since their inception. The purpose of these reviews however, weren't specifically intended to find errors (although that was great if they did), but instead to ensure and prioritize readability of code. This created an environment where the codebase itself would stand as an education tool for the rapidly growing development team.

With a focus on code readability, the motivation for reviews grew to include ensuring uniform style / design, safety, history tracking, and testing.

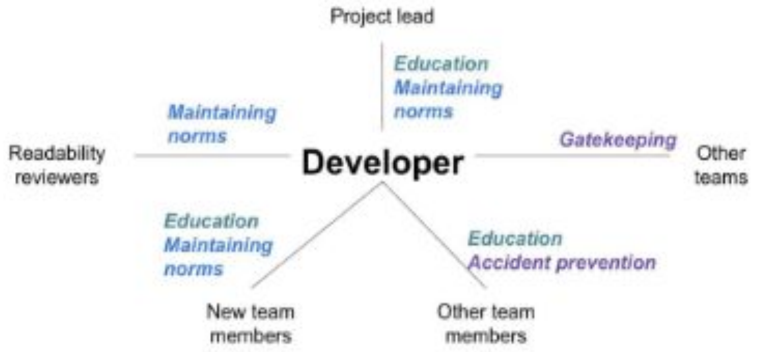

Figure 1: Relationship diagram describing which themes of review expectations appeared primarily within a particular author/reviewer context.

Taken from: Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '18). Association for Computing Machinery, New York, NY, USA, 181–190. https://doi.org/10.1145/3183519.3183525

Foundation for Review Process

Ownership

- Google uses a hierarchical model for representing code ownership
- Each directory in the codebase is “owned” by a set of developers
- The ownership set is in charge of mainly writing, as well as reviewing / approving changes made to their directory

Readability

- Given Google’s intense focus on readability, it follows that there is an internal readability certification for each language
- Only developers granted a readability certification may write and/or review code
- Readability is granted by reviewers that ensure the author is utilizing clean code practices and has a deep grasp on the language’s features

Workflow and Critique

The flow of code review very much emulates that of using Critique, Google's internal code review tool, which follows a procedural approach. Code is first created, then previewed with static analysis output, published to reviewers, audited / annotated, iterated, and finally approved.

One interesting note about the selection of reviewers chosen for a specific change, Critique will recommend who to review based on how recently they've previously reviewed that code and how many changes they've made in the past to that code.

Google integrates pre-commit hooks that ensure no commit will cause build failure nor will it fail to adhere to style checks. The results of these along with other static analyses are displayed as color coded comments in the margin via Critique's interface.

Statistics

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli (2018) reported:

- 80% of developers make < 7 changes a week
- Median developer makes 3 changes a week and reviews 4
- 80% of reviewers audit < 10 changes a week
- Median latency for entire code review is < 4 hours
- 35% of edits only modify one file
- Median number of lines modified in a change = 24
- < 25% of changes receive more than one reviewer
- 70% of edits are committed within a day of being published for initial review

The takeaways from this are that having smaller changes in each commit and fewer reviewers that are well versed with the code at hand is more effective than the alternative

Review Challenges

The researchers of the paper identified 5 key themes that posed as challenges for the code review process.

1. Distance
2. Social Interactions
3. Subject
4. Context
5. Customization

Distance is a challenge that many people are familiar with notably in the last couple years with covid. Being physically far from someone reviewing your code reduces streamlined communication pathways. Additionally inter-department reviews can pose as a distance of sorts where expectations and norms are different.

Social interactions of any type give rise to potential character conflicts such as using a demeaning tone or trying to exert their power / authority in a review exchange. This leads to overall frustration and reduced productivity.

Discrepancies in what each party thought the topic of discussion is can also lead to inefficiencies in the process. This is why it’s essential to clearly state the intention of the review especially for design reviews.

There are varying levels of necessity for making a change and misunderstanding around the severity can cause communication problems.

Observations and Takeaways of Modern Code Review

The authors findings boiled down to the following:

- Lighter is better
  - Having only 1 reviewer improves productivity
  - More efficient than original code inspections
- Less is more
  - Having more frequent but less cumbersome commits is better for streamlining the entire process from edit to commit
  - Large edits reduces quality of comments and lengthens the review process overall
- Google has the most lightweight and quick code review process out of its industry peers
- Google's Critique review tool largely guides and facilitates the workflow of code review and analysis
  - Static analysis integration is priceless
- With experience at Google comes the quality over quantity approach in terms of code review and authorship

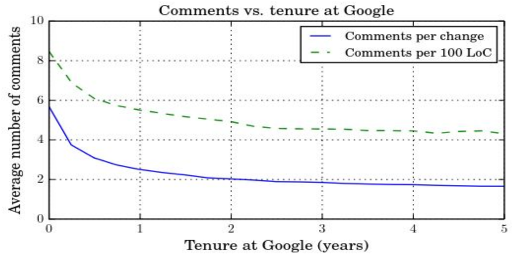

Figure 2: Reviewer comments vs. author's tenure at Google

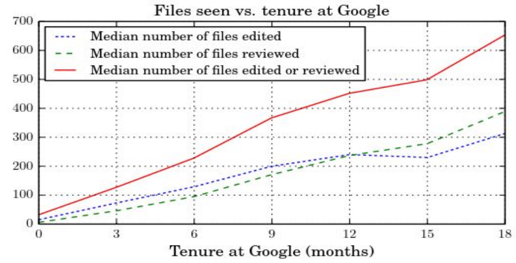

Figure 3: The number of distinct files seen (edited or reviewed, or both) by a full-time employee over time.

Taken from: Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '18). Association for Computing Machinery, New York, NY, USA, 181–190.

References

[1] Rachel Potvin and Josh Levenburg. 2016. Why Google StoresBillions of Lines of Code in a Single Repository.Commun. ACM(2016).

[2] A.F. Ackerman, L.S. Buchwald, and F.H. Lewski. 1989. Softwareinspections: An effective verification process.IEEE Software6,3 (1989), 31–36.

[3] A.F. Ackerman, P.J. Fowler, and R.G. Ebenau. 1984. Softwareinspections and the industrial production of software. InSym-posium on Software validation: inspection-testing-verification-alternatives.

[4] M.E. Fagan. 1976. Design and code inspections to reduce errorsin program development.IBM Systems Journal15, 3 (1976),182–211.

Original paper:

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '18). Association for Computing Machinery, New York, NY, USA, 181–190. https://doi.org/10.1145/3183519.3183525