Code Reviews at Google (2010)

William T. Thomason

WEB-430

Assignment 6.3

Code Reviews at Google (2010)

Google is a very large scale tech company with thousands of developers who make things happen. In 2010 Google had all their developers working off a single code source they called the trunk. Developers would perform over 5,500 commits to this trunk per week. This results to over twenty changes being check in every minute and 50% of the codebase being changed every month. This is a huge task to manage and prevent issues from happening. Google implemented that all team members send their commits through mandatory code reviews that cover the following areas:

* Code readability for languages
* Ownership assignments for code sub-trees to maintain consistency and correctness
* Code transparency and code contributions across teams

With this process typically the larger the change submitted for code review, the larger the lead time required to sign off that change. Lead time is the latency between the initiation and execution of a process. For example, the lead time between the placement of an order and delivery of a new car from a manufacturer may be anywhere from 2 weeks to 6 months.

The engineering director at Google Randy Shoup had started a project to solve a technical problem that Google was experiencing. This project took months to write with over 3,000 lines of code. After Shoup completed the project he requested for the peer to review his work. This took the peer days of work to complete with the peer requesting that he doesn’t have to do this again. The peer wasn’t requesting to never review shoup’s code, he simply didn’t want to review the whole project at once. This is when Shoup discovered that working code reviews into a daily routine it streamlines the process and makes it easier to find issues. Here are some helpful hints you can use before you submit your code for review.

* Re-evaluate your code: Don’t just send the review out as soon as the tests pass. Step back and try to rethink the whole thing—can the design be cleaned up? Especially if it’s late in the day, see if a better approach occurs to you the next morning. Although this step might slow down an individual code change, it will result long-term in greater average throughput.
* Consider an informal design discussion: If there’s something you’re not sure about, pair program, talk face-to-face, or send an early diff and ask for a “pre-review” of the overall design.
* Self-review the change: Try to look at the code as critically as possible from the standpoint of someone who doesn’t know anything about it. Your code review tool can give you a radically different view of your code than the IDE. This can easily save you a round trip.
* Make the diff easy to understand: Multiple changes at once make the code harder to review. When you self-review, look for simple changes that reduce the size of the diff. For example, save significant refactoring or formatting changes for another code review.
* Don’t hide important info in the submit message: Put it in the code as well. Someone reading the code later is unlikely to look at the submit message.

References

O'Neill, T. (2017, June 19). Code Health: Too Many Comments on Your Code Reviews? Retrieved August 18, 2019, from https://testing.googleblog.com/2017/06/code-health-too-many-comments-on-your.html

Stanton, O. (2019, July 15). My Code Review Journey. Retrieved August 18, 2019, from https://medium.com/convoy-tech/code-review-tactics-3367baaa57ea

Kim, G., Debois, P., Willis, J., Humble, J., & Allspaw, J. (2017). *The DevOps handbook: How to create world-class agility, reliability, and security in technology organizations*. Portland, OR: IT Revolution Press, LLC.