Case Study

Code Reviews at Google (2010)

Matthew R. Howard

Bellevue University

Google moved to a code review process where it was incorporated into their daily work. This came about, because Google wanted to “force developers to write code that other developers could understand because they believe that code must act as a teacher for future developers.” (Sadowski, Söderberg, Church, Sipko, 2018). Additionally, it was thought that code review would be able to make more than one person familiar with each piece of code, and thereby increasing the knowledge of staying in the company (Sadowski, Söderberg, Church, Sipko, 2018). Though, the true reason for introducing code review at Google, was not only to find bugs, but to improve code understandability and maintainability (Sadowski, Söderberg, Church, Sipko, 2018).

In 2010, Google engineers were producing 20+ changes to the trunk every minute, which was 50% of their code base being changed every month (Kim, Humble, Debois, & Willis, 2016).  This meant that lead time to get the changes made had to increase because the size of the changes that needed to be reviewed was increased as well. This made code reviewers focus on the code’s style for readability and language, made code more transparent across teams, and created consistency for sub-trees (Kim, Humble, Debois, & Willis, 2016). And came out of it was: checking for consistency of style and design; ensuring adequate tests; and improving security by making sure no single developer can randomly commit code without oversight (Sadowski, Söderberg, Church, Sipko, 2018).

Today, Google developers expect to code review to educate in regards of teaching or learning with the principles first set in place when code review came about, maintain norms in the way of formatting or API usage patterns, and prevent accidents by catching bugs, defects, or other quality related issues (Sadowski, Söderberg, Church, Sipko, 2018).

References

Kim, G., Humble, J., Debois, P., & Willis, J. (2016). *The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations* (First ed.). Portland, OR: IT Revolution.

Sadowski, C., Söderberg, E., Church, L., Sipko, M. (2018), Modern Code Review: A Case Study at Google, Retrieved: August 17, 2018, From: <https://storage.googleapis.com/pub-tools-public-publication-data/pdf/80735342aebcbfc8af4878373f842c25323cb985.pdf>

Code Review Best Practices

Matthew R. Howard

Bellevue University

Some best practices for code review include: reviewing less than 400 lines of code at a time. This is because the brain can only process so much information at a time. A study conducted by SmartBear on Cisco Systems programming found that when a developer tries to review more than 400 lines of code, the ability to find error, bugs, and defects were diminished (SmartBear, n.d.).

Additionally, a team should set certain metric when reviewing. Before beginning, the team should decide how it will measure the effectiveness of peer review as well as name a few quantifiable goals (SmartBear, n.d.). A process should also be established for fixing the bugs that are found. One method recommended by SmartBear is to use a collaborative code review tool that will allow reviewers to log bugs and defects, discuss them with the developer who authored the code, and finally approve the changes in the code (SmartBear, n.d.).

Finally, a positive code review culture can help lighten the mood and make this tedious process somewhat bearable. Not to mention, having your work judged and critiqued by your peers is not fun. Managers should create a culture of collaboration, teamwork, and learning in their work centers. This will also help foster better coding. Knowing that your peer will be scrutinizing your work will drive them write cleaner code. Peer review has been found to nurture cleaner commits, the sharing of knowledge, consistency and legibility in code writing, prevent accidental errors, and meet compliance demands (Medium, 2018). In the same study that was mentioned earlier, conducted by SmartBear, they “found that “spot-checking” 20% - 33% of the code resulted in lower defect density with minimum time expenditure. The say that if someone’s code has a 1-in-3 change of being reviewed, that’s enough of an incentive to double-check your work.” (SmartBear, n.d.)

References

SmartBear, (n.d.), Best Practices for Code Review, Retrieved: August 17, 2018 from: <https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/>

Medium, (2018, Code Review Best Practices, Retrieved: August 17, 2018 from: <https://medium.com/palantir/code-review-best-practices-19e02780015f>