Too Many Comments on Your Code Reviews?

Code reviews can slow down an individual code change, but they're also an opportunity to improve your code and learn from another intelligent, experienced engineer. How can you get the most out of them?

Aim to get most of your changes approved in the first round of review, with only minor comments. **If your code reviews frequently require multiple rounds of comments, these tips can save you time.**

Spend your reviewers' time wisely—it's a limited resource. If they're catching issues that you could easily have caught yourself, you're lowering the overall productivity of your team.

Before you send out the code 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.

When you're addressing code review comments:

- **Re-evaluate your code after addressing non-trivial comments:** Take a step back and really look at the code with fresh eyes. Once you've made one set of changes, you can often find additional improvements that are enabled or suggested by those changes. Just as with any refactoring, it may take several steps to reach the best design.
- **Understand *why* the reviewer made each comment:** If you don't understand the reasoning behind a comment, don't just make the change—seek out the reviewer and learn something new.
- **Answer the reviewer's questions in the *code*:** Don't just reply—make the code easier to understand (e.g., improve a variable name, change a boolean to an enum) or add a comment. Someone else is going to have the same question later on.

More information, discussion, and archives:

`testing.googleblog.com`

Copyright Google Inc. Licensed under a Creative Commons Attribution–ShareAlike 4.0 License (http://creativecommons.org/licenses/by-sa/4.0/).