Speed of Code Reviews

Why Should Code Reviews Be Fast?

At Google, we optimize for the speed at which a team of developers can produce a product together, as opposed to optimizing for the speed at which an individual developer can write code. The speed of individual development is important, it's just not *as* important as the velocity of the entire team.

When code reviews are slow, several things happen:

- **The velocity of the team as a whole is decreased.** Yes, the individual who doesn't respond quickly to the review gets other work done. However, new features and bug fixes for the rest of the team are delayed by days, weeks, or months as each CL waits for review and re-review.
- **Developers start to protest the code review process.** If a reviewer only responds every few days, but requests major changes to the CL each time, that can be frustrating and difficult for developers. Often, this is expressed as complaints about how "strict" the reviewer is being. If the reviewer requests the *same* substantial changes (changes which really do improve code health), but responds *quickly* every time the developer makes an update, the complaints tend to disappear. **Most complaints about the code review process are actually resolved by making the process faster.**
- **Code health can be impacted.** When reviews are slow, there is increased pressure to allow developers to submit CLs that are not as good as they could be. Slow reviews also discourage code cleanups, refactorings, and further improvements to existing CLs.

How Fast Should Code Reviews Be?

If you are not in the middle of a focused task, **you should do a code review shortly after it comes in.**

One business day is the maximum time it should take to respond to a code review request (i.e., first thing the next morning).

Following these guidelines means that a typical CL should get multiple rounds of review (if needed) within a single day.

Speed vs. Interruption

There is one time where the consideration of personal velocity trumps team velocity. **If you are in the middle of a focused task, such as writing code, don't interrupt yourself to do a code review or another task.** Research has shown that it can take a long time for a developer to get back into a smooth flow of development after being interrupted. So interrupting yourself while coding is actually *more* expensive to the team than making another developer wait a bit for a code review.

Instead, wait for a break point in your work before you respond to a request for review. This could be when your current coding task is completed, after lunch, returning from a meeting, coming back from the breakroom, etc.

Fast Responses

When we talk about the speed of code reviews, it is the *response* time that we are concerned with, as opposed to how long it takes a CL to get through the whole review and be submitted. The whole process should also be fast, ideally, but **it's even more important for the *individual responses* to come quickly than it is for the whole process to happen rapidly.**

Even if it sometimes takes a long time to get through the entire review *process*, having quick responses from the reviewer throughout the process significantly eases the frustration developers can feel with "slow" code reviews.

If you are too busy to do a full review on a CL when it comes in, you can still send a quick response that lets the developer know when you will get to it, suggest other reviewers who might be able to respond more quickly, or provide some initial broad comments. (Note: none of this means you should interrupt coding even to send a response like this—send the response at a reasonable break point in your work.)

It is important that reviewers spend enough time on review that they are certain their "LGTM" means "this code meets our standards." However, individual responses should still ideally be fast.

Cross-Time-Zone Reviews

When dealing with time zone differences, try to get back to the author while they have time to respond before the end of their working hours. If they have already finished work for the day, then try to make sure your review is done before they start work the next day.

LGTM With Comments

In order to speed up code reviews, there are certain situations in which a reviewer should give LGTM/Approval even though they are also leaving unresolved comments on the CL. This should be done when at least one of the following applies:

- The reviewer is confident that the developer will appropriately address all the reviewer's remaining comments.
- The comments don't *have* to be addressed by the developer.
- The suggestions are minor, e.g. sort imports, fix a nearby typo, apply a suggested fix, remove an unused dep, etc.

The reviewer should specify which of these options they intend, if it is not otherwise clear.

LGTM With Comments is especially worth considering when the developer and reviewer are in different time zones and otherwise the developer would be waiting for a whole day just to get "LGTM, Approval".

Large CLs

If somebody sends you a code review that is so large you're not sure when you will be able to have time to review it, your typical response should be to ask the developer to split the CL into several smaller CLs that build on each other, instead of one huge CL that has to be reviewed all at once. This is usually possible and very helpful to reviewers, even if it takes additional work from the developer.

If a CL *can't* be broken up into smaller CLs, and you don't have time to review the entire thing quickly, then at least write some comments on the overall design of the CL and send it back to the developer for improvement. One of your goals as a reviewer should be to always unblock the developer or enable them to take some sort of further action quickly, without sacrificing code health to do so.

Code Review Improvements Over Time

If you follow these guidelines and you are strict with your code reviews, you should find that the entire code review process tends to go faster and faster over time. Developers learn what is required for healthy code, and send you CLs that are great from the start, requiring less and less review time. Reviewers learn to respond quickly and not add unnecessary latency into the review process.
But **don't compromise on the code review standards or quality for an imagined improvement in velocity**—it's not actually going to make anything happen more quickly, in the long run.

Emergencies

There are also emergencies where CLs must pass through the *whole* review process very quickly, and where the quality guidelines would be relaxed. However, please see What Is An Emergency? for a description of which situations actually qualify as emergencies and which don't.

Next: How to Write Code Review Comments