eng-practices

What to look for in a code review

Note: Always make sure to take into account The Standard of Code Review when considering each of these points.

Design

The most important thing to cover in a review is the overall design of the CL. Do the interactions of various pieces of code in the CL make sense? Does this change belong in your codebase, or in a library? Does it integrate well with the rest of your system? Is now a good time to add this functionality?

Functionality

Does this CL do what the developer intended? Is what the developer intended good for the users of this code? The "users" are usually both end-users (when they are affected by the change) and developers (who will have to "use" this code in the future).

Mostly, we expect developers to test CLs well-enough that they work correctly by the time they get to code review. However, as the reviewer you should still be thinking about edge cases, looking for concurrency problems, trying to think like a user, and making sure that there are no bugs that you see just by reading the code.

You *can* validate the CL if you want—the time when it's most important for a reviewer to check a CL's behavior is when it has a user-facing impact, such as a **UI change**. It's hard to understand how some changes will impact a user when you're just reading the code. For changes like that, you can have the developer give you a demo of the functionality if it's too inconvenient to patch in the CL and try it yourself.

Another time when it's particularly important to think about functionality during a code review is if there is some sort of **parallel programming** going on in the CL that could theoretically cause deadlocks or race conditions. These sorts of issues are very hard to detect by just running the code and usually need somebody (both the developer and the reviewer) to think through them carefully to be sure that problems aren't being introduced. (Note that this is also a good reason

Downloads google_github_io_eng_p… 100% Clear

very complex to do code reviews or understand the code.)

Complexity

Is the CL more complex than it should be? Check this at every level of the CL—are individual lines too complex? Are functions too complex? Are classes too complex? "Too complex" usually means **"can't be understood quickly by code readers."** It can also mean **"developers are likely to introduce bugs when they try to call or modify this code."**

A particular type of complexity is **over-engineering**, where developers have made the code more generic than it needs to be, or added functionality that isn't presently needed by the system. Reviewers should be especially vigilant about over-engineering. Encourage developers to solve the problem they know needs to be solved *now*, not the problem that the developer speculates *might* need to be solved in the future. The future problem should be solved once it arrives and you can see its actual shape and requirements in the physical universe.

Tests

Ask for unit, integration, or end-to-end tests as appropriate for the change. In general, tests should be added in the same CL as the production code unless the CL is handling an emergency.

Make sure that the tests in the CL are correct, sensible, and useful. Tests do not test themselves, and we rarely write tests for our tests—a human must ensure that tests are valid.

Will the tests actually fail when the code is broken? If the code changes beneath them, will they start producing false positives? Does each test make simple and useful assertions? Are the tests separated appropriately between different test methods?

Remember that tests are also code that has to be maintained. Don't accept complexity in tests just because they aren't part of the main binary.

Naming

Did the developer pick good names for everything? A good name is long enough to fully communicate what the item is or does, without being so long that it becomes hard to read.

Comments

Did the developer write clear comments in understandable English? Are all of the comments

Downloads google_github_io_eng_p… 100% Clear

should not be explaining *what* some code is doing. If the code isn't clear enough to explain itself, then the code should be made simpler. There are some exceptions (regular expressions and complex algorithms often benefit greatly from comments that explain what they're doing, for example) but mostly comments are for information that the code itself can't possibly contain, like the reasoning behind a decision.

It can also be helpful to look at comments that were there before this CL. Maybe there is a TODO that can be removed now, a comment advising against this change being made, etc.

Note that comments are different from *documentation* of classes, modules, or functions, which should instead express the purpose of a piece of code, how it should be used, and how it behaves when used.

Style

We have style guides at Google for all of our major languages, and even for most of the minor languages. Make sure the CL follows the appropriate style guides.

If you want to improve some style point that isn't in the style guide, prefix your comment with "Nit:" to let the developer know that it's a nitpick that you think would improve the code but isn't mandatory. Don't block CLs from being submitted based only on personal style preferences.

The author of the CL should not include major style changes combined with other changes. It makes it hard to see what is being changed in the CL, makes merges and rollbacks more complex, and causes other problems. For example, if the author wants to reformat the whole file, have them send you just the reformatting as one CL, and then send another CL with their functional changes after that.

Consistency

What if the existing code is inconsistent with the style guide? Per our code review principles, the style guide is the absolute authority: if something is required by the style guide, the CL should follow the guidelines.

In some cases, the style guide makes recommendations rather than declaring requirements. In these cases, it's a judgment call whether the new code should be consistent with the recommendations or the surrounding code. Bias towards following the style guide unless the local inconsistency would be too confusing.

If no other rule applies, the author should maintain consistency with the existing code.

Downloads google_github_io_eng_p… 100% Clear

Either way, encourage the author to file a bug and add a TODO for cleaning up existing code.

Documentation

If a CL changes how users build, test, interact with, or release code, check to see that it also updates associated documentation, including READMEs, g3doc pages, and any generated reference docs. If the CL deletes or deprecates code, consider whether the documentation should also be deleted. If documentation is missing, ask for it.

Every Line

In the general case, look at *every* line of code that you have been assigned to review. Some things like data files, generated code, or large data structures you can scan over sometimes, but don't scan over a human-written class, function, or block of code and assume that what's inside of it is okay. Obviously some code deserves more careful scrutiny than other code—that's a judgment call that you have to make—but you should at least be sure that you *understand* what all the code is doing.

If it's too hard for you to read the code and this is slowing down the review, then you should let the developer know that and wait for them to clarify it before you try to review it. At Google, we hire great software engineers, and you are one of them. If you can't understand the code, it's very likely that other developers won't either. So you're also helping future developers understand this code, when you ask the developer to clarify it.

If you understand the code but you don't feel qualified to do some part of the review, make sure there is a reviewer on the CL who is qualified, particularly for complex issues such as privacy, security, concurrency, accessibility, internationalization, etc.

Exceptions

What if it doesn't make sense for you to review every line? For example, you are one of multiple reviewers on a CL and may be asked:

- To review only certain files that are part of a larger change.
- To review only certain aspects of the CL, such as the high-level design, privacy or security implications, etc.

In these cases, note in a comment which parts you reviewed. Prefer giving LGTM with comments .

If you instead wish to grant LGTM after confirming that other reviewers have reviewed other parts

Downloads google_github_io_eng_p… 100% Clear

CL has reached the desired state.

Context

It is often helpful to look at the CL in a broad context. Usually the code review tool will only show you a few lines of code around the parts that are being changed. Sometimes you have to look at the whole file to be sure that the change actually makes sense. For example, you might see only four new lines being added, but when you look at the whole file, you see those four lines are in a 50-line method that now really needs to be broken up into smaller methods.

It's also useful to think about the CL in the context of the system as a whole. Is this CL improving the code health of the system or is it making the whole system more complex, less tested, etc.? **Don't accept CLs that degrade the code health of the system.** Most systems become complex through many small changes that add up, so it's important to prevent even small complexities in new changes.

Good Things

If you see something nice in the CL, tell the developer, especially when they addressed one of your comments in a great way. Code reviews often just focus on mistakes, but they should offer encouragement and appreciation for good practices, as well. It's sometimes even more valuable, in terms of mentoring, to tell a developer what they did right than to tell them what they did wrong.

Summary

In doing a code review, you should make sure that:

- The code is well-designed.
- The functionality is good for the users of the code.
- Any UI changes are sensible and look good.
- Any parallel programming is done safely.
- The code isn't more complex than it needs to be.
- The developer isn't implementing things they *might* need in the future but don't know they need now.
- Code has appropriate unit tests.
- Tests are well-designed.

Downloads google_github_io_eng_p… 100% Clear

- Comments are clear and useful, and mostly explain *why* instead of *what*.
- Code is appropriately documented (generally in g3doc).
- The code conforms to our style guides.

Make sure to review **every line** of code you've been asked to review, look at the **context**, make sure you're **improving code health**, and compliment developers on **good things** that they do.

Next: Navigating a CL in Review

This site is open source. Improve this page.

Downloads google_github_io_eng_p… 100% Clear