How to Review Code

An Opinionated Guide

version 0.1.0

Regan Koopmans

Introduction

After some years of working in professional software development, I have reached the conclusion that reviewing code is at least as important as writing code. Reviewing code can be an avenue for improving programming skills and encourages critical thinking about a code base outside of your own contributions. I have written this guide to myself as a reminder to maintain habits that I think generate better code reviews. This guide is oriented toward reviewing patches of changes in the style characterised by pull requests on GitHub and similar git hosting services.

Question whether the change is necessary

If a code-level solution is not the optimal way to solve the problem, it makes no sense to review the code in the first place. It is important to establish this upfront because it might save a great deal of redundant energy in arguing the appropriate name of a variable in a code block that should not exist.

Check the code out locally

This is a big one. It might be tempting to review code purely through the browser. Checking out someone's branch and potentially stashing some of your own changes can seem like an unnecessary interruption to your own workflow. However, this makes for better code reviews. Code should be reviewed with the same tools it is written with. Modern IDEs are incredibly powerful tools and will be able to spot unused variables and a number of common code smells. You might assume that the author of the changes would catch these, but that is not always the case.

Furthermore, code changes do not exist in isolation. Code changes exist within the surrounding code which should be considered. This context is often lost in browser-only reviews. Checking out the branch also makes it easier to "play" with the code: trying to write an expression more succinctly or in a more performant manner.

Saying this, it is not essential that every patch be reviewed locally. There is a class of pull requests for which this is less useful (updates to dependency versions, copy text changes). Use your discretion to skim these, but try to review most code locally.

Optimise for high-level suggestions

Reviewing code is time-consuming. There are several reasons for this, predominantly that most changes come with a context that needs to be internalised by the reviewer. To review a bugfix one should know the bug, at least at the surface level. There are suggestions that can be made in a pull-request that are less dependent on context (styling, naming), but these should be automated as CI build steps as far as possible. Adopting a standard code-formatter within a project is great for this.

When reviewing code you should optimise for quality over quantity. Suggestions in code-reviews exist on a spectrum of abstraction. This ranges from "Does this code fulfil the intended purpose?" to "Should this field be final?" Try to focus your energy on obtaining deeper, more fundamental suggestions first. This heuristic is based on the following:

1. Higher-level suggestions are less likely to be identified by other team members and thus less likely to be identified at all if not by you.

2. Higher-level suggestions tend to generate more changes than lower-level suggestions and may even render the lower-level suggestions redundant. Perhaps a certain process should be asynchronous, which would eliminate some hacky workaround in the code being reviewed. It clearly makes more sense to focus on the asynchronous suggestion rather than pull apart the specifics of its inferior hacky substitute.

Common code deficiencies

Pull-requests tend to be lacking in some consistent ways, here these are discussed such that they might be identified more readily.

Tests

If a patch modifies application behaviour, either tests should be modified, or new tests are introduced, or both.