Out of the abyss

How to peer review like-a-boss

By Aaron Carlino (aka unclecheese)

SilverStripe

Perfecting the art of Peer Review

Use and share these tips to peer review like-a-boss!

Why we peer review?

Developers are introverts. Code is art. We're artists, and like all artists, we love to create beautiful things. Alone. Our jobs rarely afford us that luxury, though, and that's not such a bad thing. Coming up for air once in awhile can be really beneficial and healthy. It's important to remind yourself that what you're creating isn't just about you.

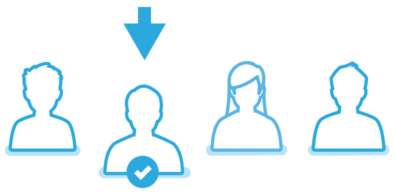
One great way to get out of your head is to engage in peer reviews. Whether you already have a peer review process established, or are new to the idea, these ten tips can help you become a more aware, outward-thinking developer.

For the Reviewee:

1

Write code that's reviewable

It may seem like this goes without saying, but you'd be surprised how often peer reviews don't move forward simply because the code is not of good enough quality to warrant a peer review. That the code is functioning is not enough to qualify it for peer review. It should be clean and easily readable by another developer.



2

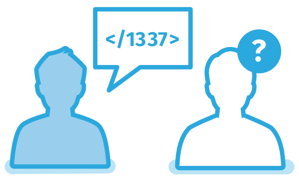
Select the right peer reviewer

Just because a peer is more senior or more experienced than you does not intrinsically qualify that person to peer review your code. Prefer someone of lower seniority with the right skillset to someone of higher seniority who lacks everyday experience with the technologies your code is using.

3

If you don't understand the feedback, don't implement it

Has your peer review come back littered with comments on concepts that are over your head? If so, this is not your queue to simply blindly implement the suggested patches. This is your time to learn about those things, not just so that you understand what it is you're doing, but more importantly, so you'll know better next time.



For the Reviewer:

4

Be timely

Once a body of code gets stale, it becomes much more difficult to rebase and merge. As a reviewer, be quick to get involved before things cool off. Depending on the project, you may find that the author of the code forgets about it after a while and may have moved on to other things. Let the author know you're interested. If you care, your reviewee will follow suit.

5

Don’t be afraid to say “no”

This can be a really difficult aspect of a peer review, but at the end of the day, it’s not really a review if you’re not pushing back.
Here are a few example questions you can ask to help figure out if you should be pushing back.

- Is there a potential for regressions?
- Does this change violate semver?
- Does this change only serve an edge case?
- Does this change make use of all existing and available APIs in the framework?
- Are variable names appropriate for their context?
- Do the source code comments make sense?

6

Encourage risk taking

A tennis player aims to get 70-75% of first serves in. A player who has a higher success rate than that is not hitting hard enough. As a reviewer, it’s your job to encourage a healthy level of risk taking. If 100% of the code you review is getting accepted, your review process is doing absolutely nothing for you or your reviewee.

7

Understand the shelf life

Be sure to have appropriate expectations for how long the reviewee’s work will live in the codebase. Is this a feature that’s likely on the chopping block next quarter? If so, lower the bar a bit. Don’t spend an inappropriate amount of time picking apart every little detail if you know it will be eclipsed in the near future.

8

Be proactive

Many of the best peer reviews are done proactively, when one team member asks to look at what someone else is working on, in the moment. Remember, the end game isn’t code, it’s a great product. Helping out your team members is one way to increase your chances of getting there.

9

Get to know your robots

Much of a peer review is objective. You’re enforcing coding conventions, running unit tests, and maybe even checking for performance impacts. All of that can be done by robots. Get familiar with scritinizr and/or travis, and stop spending time on things that machines can do faster and better than you.

10

Scope it

There’s a fable in open-source communities that when a pull request is 20 lines, everyone picks it apart to eternity, but when it’s 20,000 lines, everyone just glosses over it. We tend to seek the most economical ways to use our attention and brain power. For large bodies of code, scope your review to just a manageable section. Your review will mean much more if it garners all of your energy, and not just a fractional handout thereof.

Learn more development tips on the SilverStripe blog

Read our original blog on peer reviewing practices

www.silverstripe.com