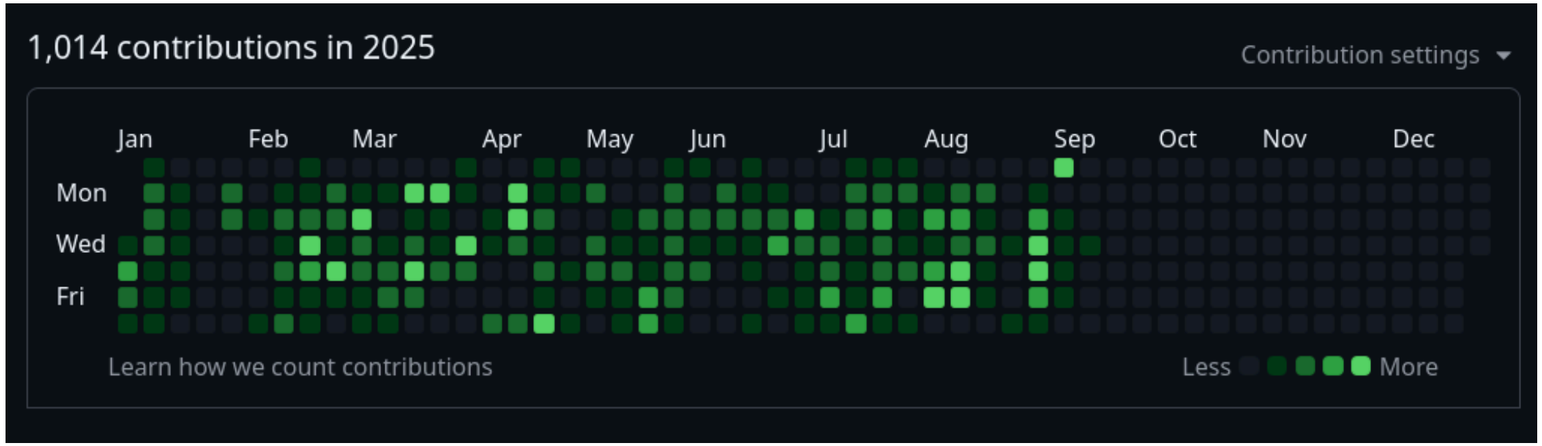
Navigating Code Reviews

as a code author

Ben Deane / CppCon Lightning Talks / 2025-09-18

I Write Code

I'm senior, but I still write code.



Almost every day.
(This is about half my contributions – I have personal and work accounts.)

Making things better for everyone

As a code author:

- I want my code to contribute to the project
- I want the project to be high quality
- I want to see my code in the main line ASAP
- I have more things to do after this

It's in my interest for my code to be easy to review.
And that's also in the reviewer's interest!

First, it's not a surprise

Many times, new code comes along with a new design.
(However large or small.)

The code review should not be the first time
the reviewer hears about this design.

When you're thinking about the way you'll go,
just grab someone for 10 minutes to talk it through.

Make the review easy

Don't hit the reviewer with a wall of code.

A PR can be more than one commit!

- Keep the commits small and contained.
- The codebase should build (and pass tests) at every commit.
- Separate robotic commits (they can be larger).
- Don't mix in "drive-by" fixes.

Tools of the trade

Get used to changing (local) history.
(If you're using git and *not* changing local history,
you're not getting most of the benefit.)

At minimum, learn to use:

- `git add -p` or `git add -i`
- `git rebase -i`

Commit as you develop! You *can* make drive-by fixes *if* you keep them separate.
Getting your work ready for review by reworking the commits is part of the workflow.

Write good commit messages

There is tons of advice online about this.
People are still lazy.

Do *something* sensible.

- maybe adopt a template
- enforce with a hook if you like
- require ticket numbers/links if that's how your team works
- `https://conventionalcommits.org/`
- `https://gitmoji.dev/`

Write good tests

(And, as a reviewer, maybe start by looking at the tests.)

You don't have to follow test-driven development.
You *do* have to have tests that exercise your code.

Make the tests good. Make them clear. Make them self-contained.
If the tests are easy to read, the reviewer can know:

- the code works
- *which parts* of the code correspond

Comment on your own PR

You talked with someone about the design. You separated commits.
You have a good commit message and good tests.

There are still places in your code, where when the rubber met the road,
you had to make a compromise.

You are allowed to comment on your own PRs!

You can point out places that reviewers should pay attention to.
Places where you had to change the design a bit.

When issues arise, be clear about resolution

When an issue comes up in a PR,
both sides must be clear about the resolution process.

Is this something:

- that the author can fix and resolve themselves?
- that the author should fix but requires further review?
- that is optional in this PR?
- that should be done in a follow-up PR?

Don't leave a dangling open comment.
Consider `https://conventionalcomments.org/`

Reciprocate!

If you expect good, timely code reviews,
then you must offer good, timely code reviews.

Use your github (or other tool) notifications tab!
Make it part of your daily workflow.

You *don't* have to do things on-demand.
You *do* have to make it a regular habit.
I start every day by looking at my github notifications.

Happy Coding!

Discuss design up front.

Structure commits well.

Write good commit messages, tests and comments.

The review is a discussion, not an exam.

Be clear about issue resolution.

Reciprocate code reviews.