Code Review for Scientific Software: Experiences building an online tutorial

Date: 20 March 2024
Presented by: Helen Kershaw (NSF National Center for Atmospheric Research)

(The slides are available via the link in the sidebar of the page linked above.)

Q: Have you thought about using a code review process for AI generated code?

A: Great question. I think the more tools such as Github copilot become more and more common, much more of creating code becomes reviewing - assessing the value of AI generated code.

Q: How do you archive the reviews for the long-term? I.e. the code can be moved transparently using git, but how would you access the review comments etc. if Microsoft shuts github down in the future? (I ran into this issue when Atlassian/Bitbucket suddenly sunset mercurial support.)

Audience answers:

- An answer from a listener: everything worthy of being archived should be in the commit messages! *(Great point! There was a tool (hg-review) that put mercurial reviews in a parallel repo, but it was abandoned.)*
- GitLab has a community edition you can run on prem. (also Heptapod which supports mercurial), which is why we originally preferred it. Perhaps GitHub does too. Full export, however (issues, PRs, etc.) are not yet supported (but see Full Project Mirroring).
- GitHub has an API, so in principle, you could write a script to archive issues, PRs, etc. off of the site

A: I don't have a great answer to this. I believe you can scrape/capture all of the activity on a GitHub repository. I think the other option is to run an on premise version.

Q: Does our running the tutorial result in the creation of a repository that we'll need to delete afterward? Thanks.

A: Yes. Unless you use the take-a-look version.

Q: Is this all done in a fork?

A: Yes. Unless you use the take-a-look version.

Q: Would a walk through first sometimes prejudice the reviewer to the coder's viewpoint and thus miss some problems?

A: Yes this is a good point. Similar to the problem you have in meetings where the HIPPO "Highest Paid Person's Opinion" carries significant weight. It is a balance between collaboration and Q&A.

Comment: One of the best ways we found to pre-empt friction points before review is to have clear repo guidelines around PRs - not just around style but also scope e.g. 'only one feature per PR'

A: ditto to this comment.

Q: Do you think having some kind of "design reviews" before code reviews can solves some of the friction with code reviews?

A: I love this idea. It is something we as a group try to do, but often we get the code at the pull request stage vs. the design stage.

Q: Do you have some suggestions/tips for working with scientists/non-software engineers on the code review process?

A: I think a good tip is to ask them what they are looking for in the review, e.g. is there a part of the code that they are particularly concerned about, or is critical. Review has to be helpful, otherwise people won't do it. So if you can get an early 'win' with people by improving their code or solving a problem they have, that can be a good way to build relationships.

Comment: An interesting read:
https://github.blog/2022-06-30-write-better-commits-build-better-projects/

Comment: To facilitate real-time code-editing etc. (especially for remote collaboration) we use the CoCalc (https://cocalc.com/) platform. They have tools like collaborative whiteboards, interactive Jupyter notebooks, and a full virtual platform so collaborators do not need to install anything locally.

Q: I'm also wondering how much we can learn from other engineering domains that often have to build large, complex things together.

A: Me too.

Q: How do Agile practices interface with your experiences and recommendations for code reviews?

A: How you review becomes part of your retrospective. Part of agile is people over process, so I guess as part of a sprint (or other agile mechanism) taking a look at what worked with review, what didn't.

Comment: One point here is that getting ones code in early will enable others to use it sooner - thus better testing, more exposure, etc.

A: Yes totally agree, better to get people using the code.

Q: What do you think about changing history in the repo?

A: So in DART, we in general don't rewrite history because we have various external groups developing with varying git experience (occasionally we do rewrite history on branches if it is only the core team working on the code, e.g. a common one is students not setting up their git config, so we do rewrite history so they get GitHub 'credit' for their contributions, and occasionally we will rebase a before merging a pull request.). However,when I am looking at other software, I do appreciate linear commit histories that tell a clear story. But we don't squash as a general rule.

Q: Does code review include reviewing code documentation as well?

A: yes definitely. Good documentation that is easy to navigate I think takes the same thought that you would put into code design. We are looking at our mountain of documentation at the moment and the future plan is to look at UX of how different audiences interact with it, e.g. developers, users, curious people. We've not got a good solution yet - I think that would be a great topic for another HPC-BP seminar.