Peer review work practices

The purpose of this document is to educate data science team members about the why and how of peer review in our context in order to:

- Improve the quality of technical output
- Foster peer learning between the team members

Peer review in the technical fields is sometimes called code review or software review. This document will use the phrase *code review* since our technical output in the data science team is primarily Jupyter notebooks.

Why and how is code review done?

The purpose of code review is to find opportunities to grow, not criticise. Rather than aiming to write perfect code, you should try to improve on your code iteratively by incorporating feedback from your peers. Code review is a less formal process than auditing, which is done to ensure compliance with standards or contractual obligations. It aims to find and fix problems with the code and prevent errors from propagating throughout a project.

Code review can be done through a standardised process known as pull requests (a code review request feature is available on tools such as GitHub). If you prefer not to formally commit your work to GitHub for the whole team to see, you can also do informal "over the shoulder" code reviews in a Teams call, or by sharing files with a team member.

All members of the data science team should contribute to doing code review for their peers.

Code review best practices

For code authors:

Share your code regularly. This allows you to get timely feedback, so you aren't repeating mistakes.

Request specific feedback. This increases the efficiency for manual code review. You can use the title and description of a pull request to specify what areas you want the reviewers to look at.

Keep pull requests small and single purpose.

For code reviewers:

Give a "what" and a "why". Include what needs to be fixed and why the change should be made.

Code should be improved iteratively. Initial reviews should focus on the fundamental functionality, later reviews can address smaller things such as style guide adherence.

Say good things too. Don't just focus on mistakes, but also tell the developer what they did right.

Keep your feedback short and focused. There is no such thing as "perfect" code, and you should not give the author feedback on every issue.

Don't review too many lines of code at once. Reviewing too much in one sitting impacts your ability to give useful feedback[1].

Share the work. You can assign someone else if you would like another peer to review the code (either informally, or by adding them as a reviewer on GitHub). This is particularly helpful if you are new to code review as you can learn from how your peers give feedback.

[1] According to a study on code review by Cisco.

Guiding questions

These questions can help you give feedback during code review. They don't all need to be answered for each code review.

Functionality

- What is the code doing? (You can also ask the author directly if they haven't specified this when requesting review)
- Does the code work as the author intended? Look for any errors or parts that take a long time to run.
- Is what the author intended best for the users of this code? (Here users could be readers of the published use case, anyone who might be maintaining this code in the future)

Reusability

- Does it follow DRY principles (Don't Repeat Yourself)?

Readability

- Are there good names? A good name should be long enough to fully communicate the purpose of a variable or method, without being so long it is difficult to read.
- Does it follow a style similar to other code in the same project, or some other conventions?
- Are the comments all understandable, and actually necessary? Comments generally should explain why code exists, not what it is doing (which should be clear from the code itself).

Maintainability

- Is the code too complex? Meaning, is it likely someone trying to change the code will cause errors?

These questions don't cover code review principles related to more general code review in other environments, such as error handling, security, test coverage, or code architecture (encapsulation, modularisation, single responsibility principle). Please see the *Further Reading* section if you are interested in in reading more about code review in general, and the underlying principles.

Relevant style guides

Style Guide for Python Code

The Hitchhiker's Guide to Python (also in the Deakin Library as an ebook)

Further reading

Carullo, G 2020, *Implementing effective code reviews : how to build and maintain clean code*, Apress.

Phillips, D., Kasampalis, S. and Giridhar, C. (2016) *Python: Master the Art of Design Patterns*. Birmingham, UK: Packt Publishing (Learning Path).

Author

Hannah Smith (version 1, September 22 2022)