45426: Teste e Qualidade de Software

Code improvement through peer-reviews (in the CI pipeline)

Ilídio Oliveira

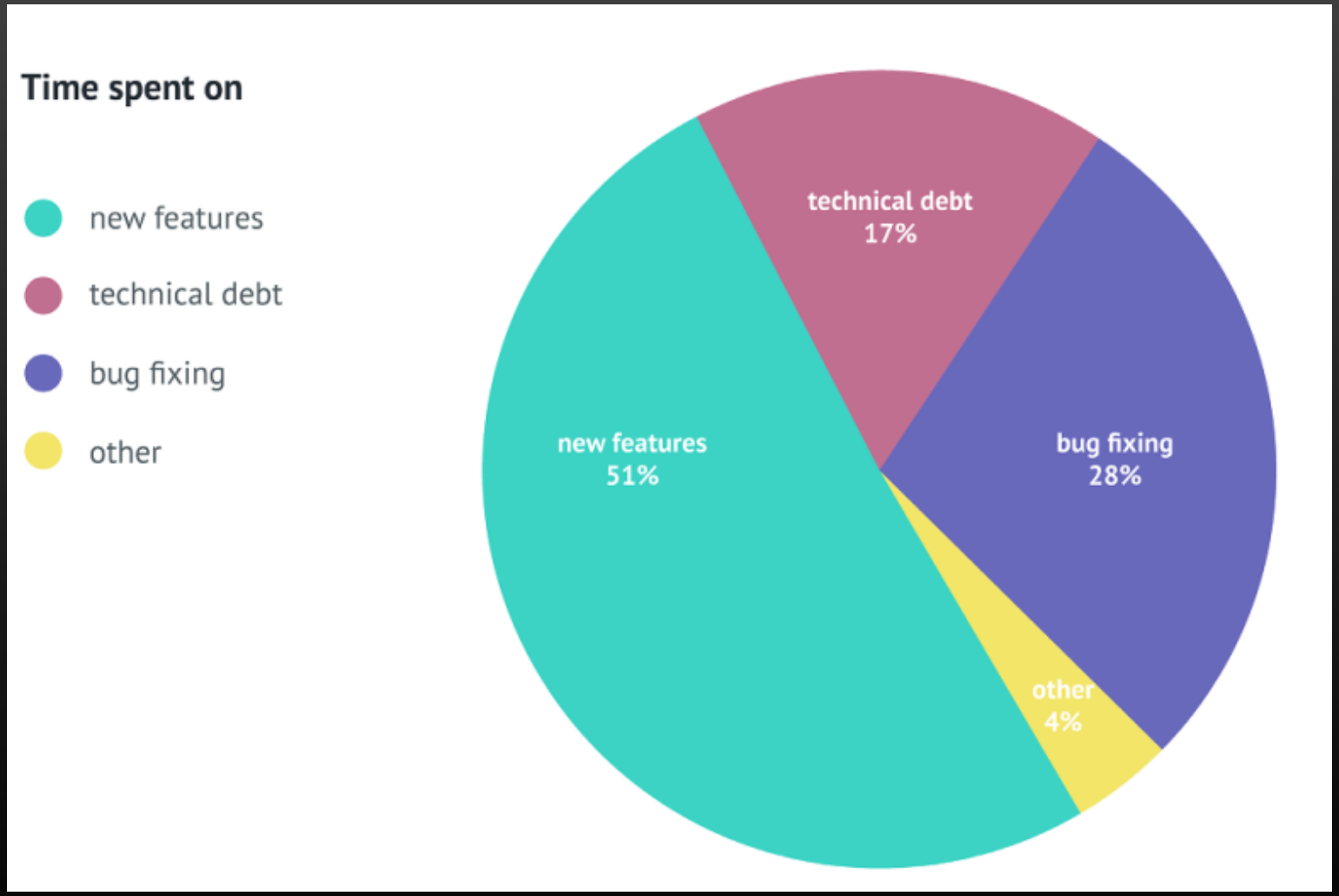
v2024-05-21

universidade de aveiro
departamento de eletrónica, telecomunicações e informática
deti

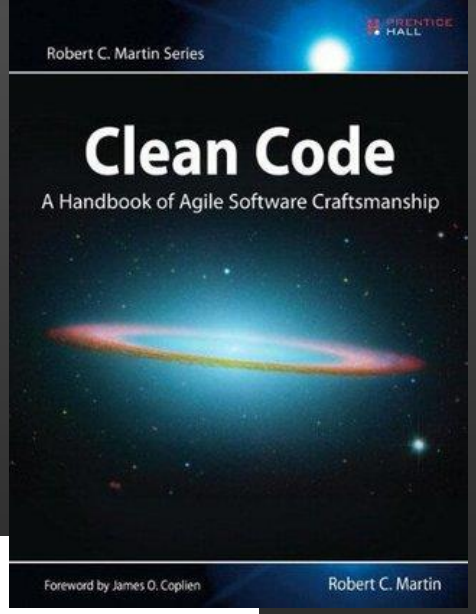
Learning objectives

- Describe the goals of formal and informal code reviews.
- Enumerate sample problems that can corrected in code reviews
- Identify best practices to conduct code reviews
- Explain the role of code Styles towards software maintenance

Survey among +600 developers



https://www.codacy.com/ebooks/guide-to-code-reviews



Chapter 1: Clean Code

...

You get the drift. Indeed, the ratio of time spent reading vs. writing is well over 10:1. We are *constantly* reading old code as part of the effort to write new code.

Because this ratio is so high, we want the reading of code to be easy, even if it makes the writing harder. Of course there's no way to write code without reading it, so *making it easy to read actually makes it easier to write.*

You can make the code easier to maintain

Practices to find problems in the code

Static code analysis

Code reviews

…

Culture for clean/readable code

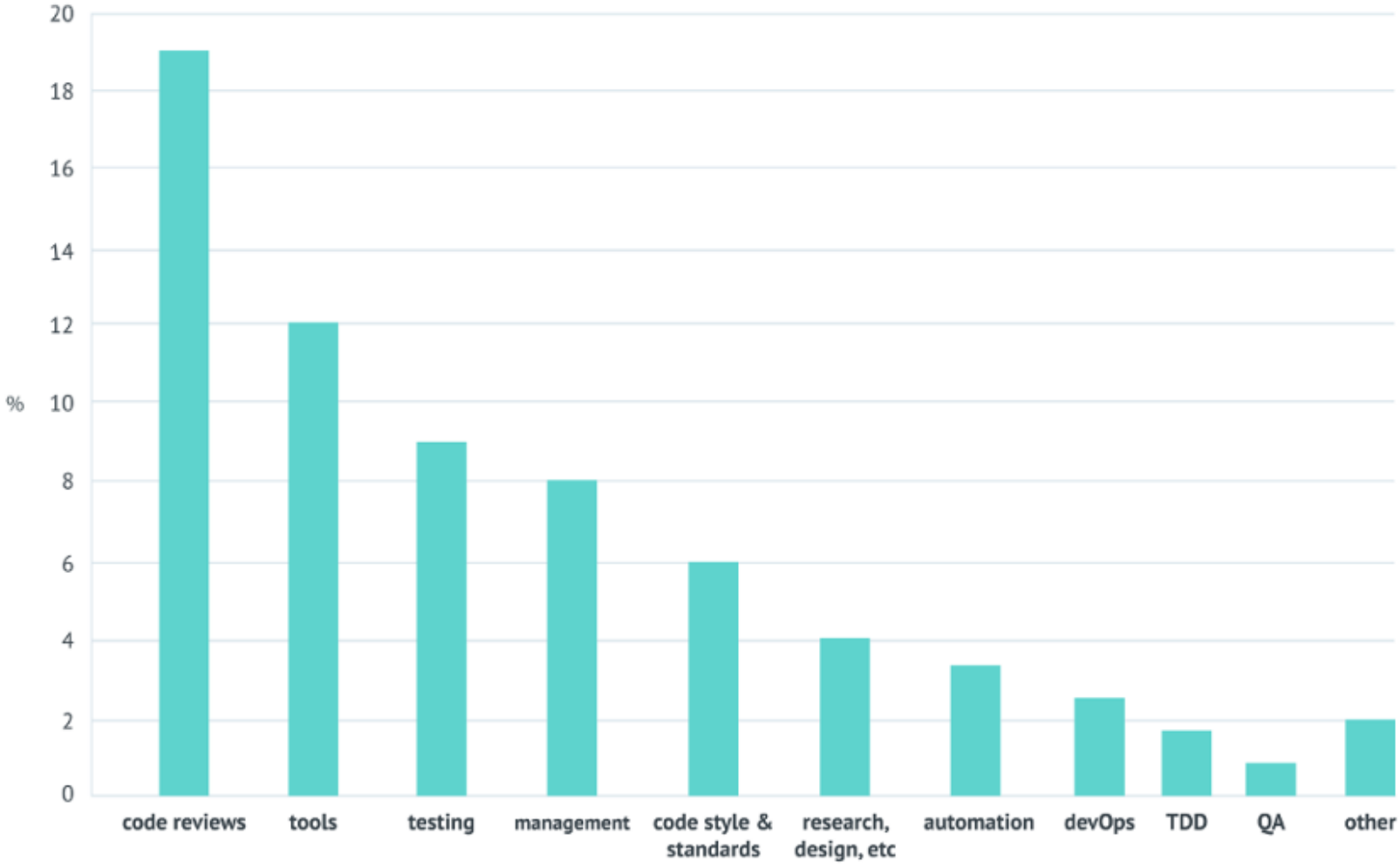
Developer/style guidelines

Definition of Done

Patterns (reuse)

…

What change in your development process had the biggest impact to code quality?*



https://www.codacy.com/ebooks/guide-to-code-reviews

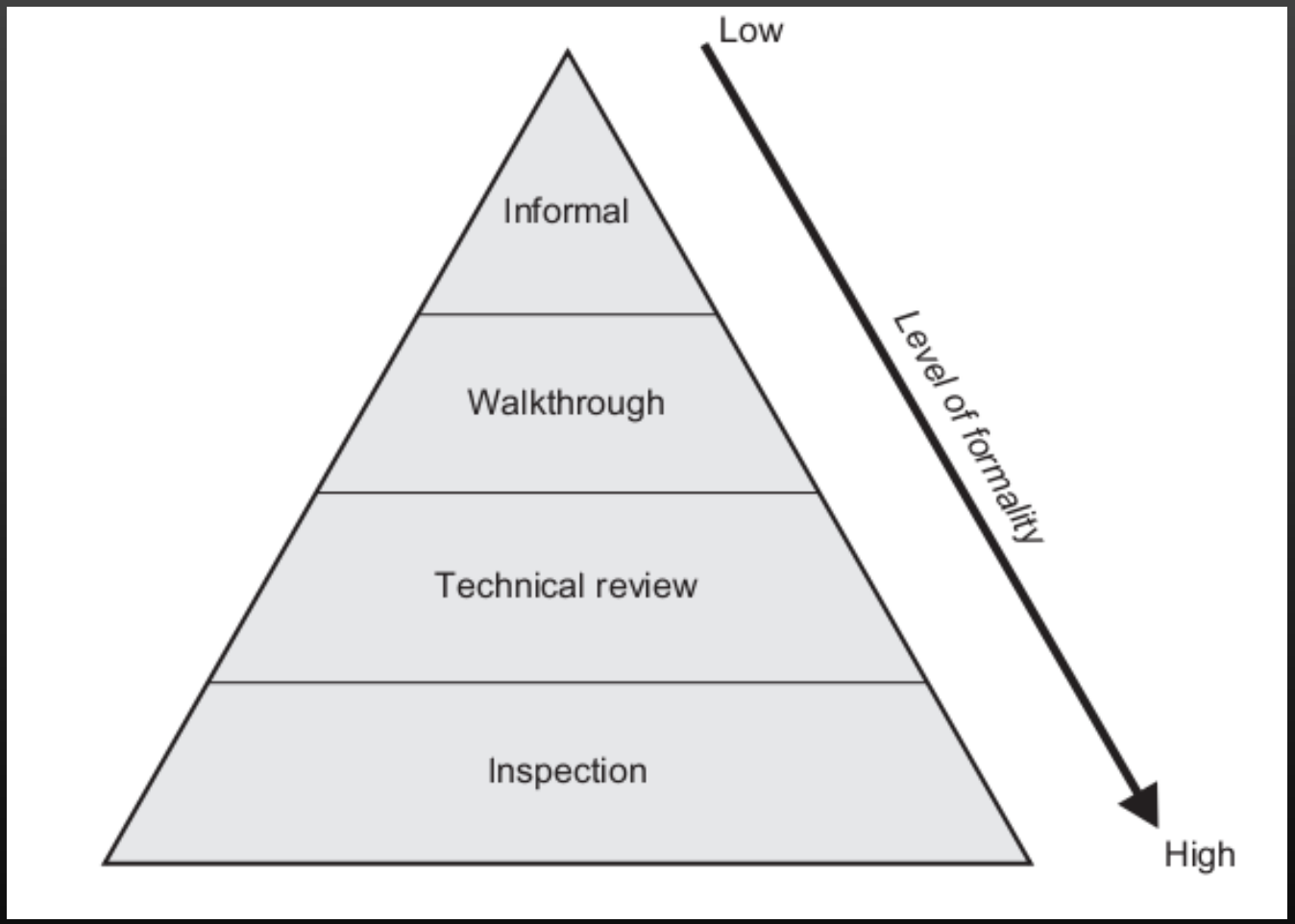
*The question was open-ended to avoid leading the respondents into specific answers.

Issues (likely to address) in a code review

- Verify feature requirements
- Assess readability (clean code)
- Maintainability issues
- Check for security risks/best practices
- Consider speed and performance
- Confirm adequate documentation
- Inspect naming conventions
- Design issues (general and project-related)

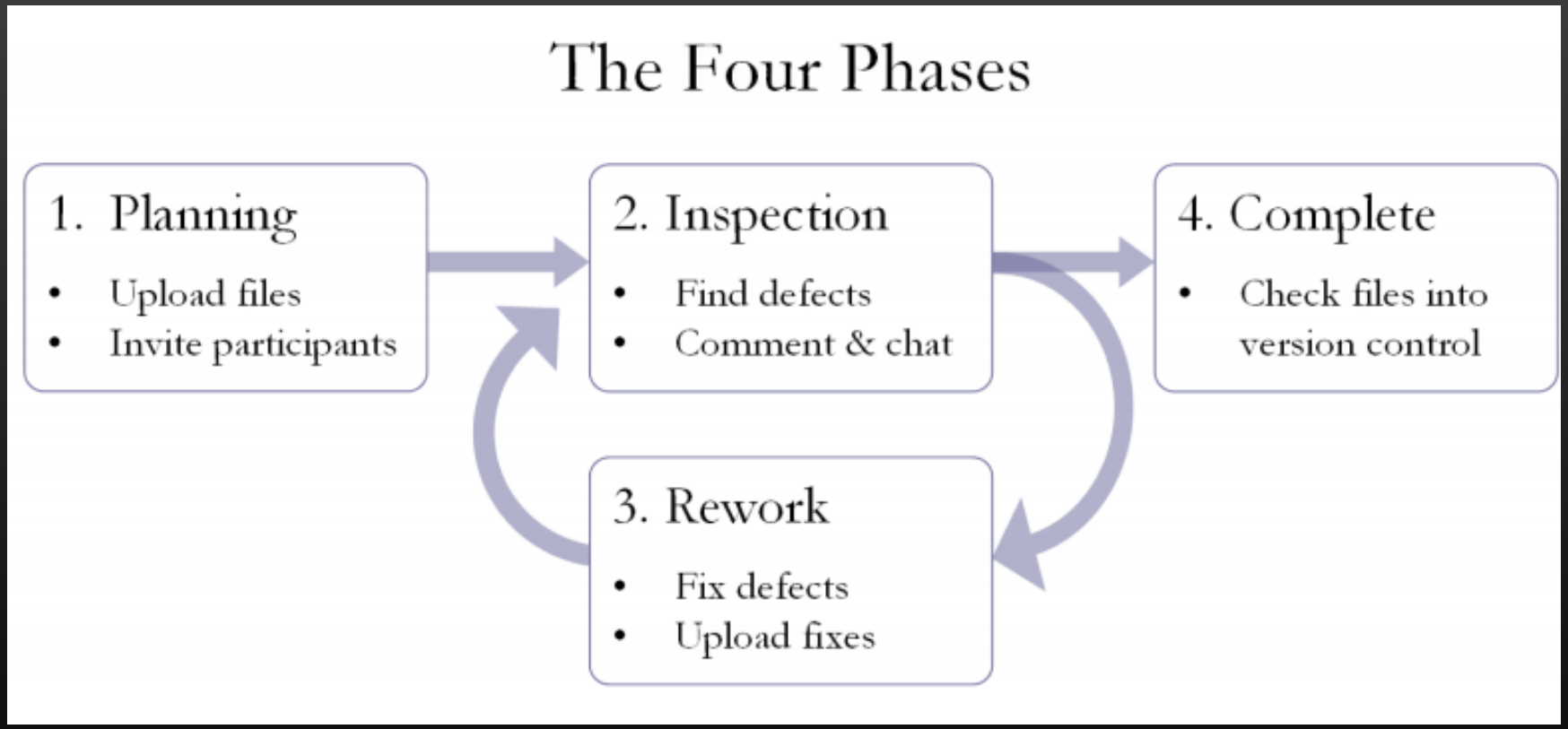
https://www.pluralsight.com/blog/software-development/code-review-checklist

A review process can have very different levels of formality (Informal to Formal Tech Review)

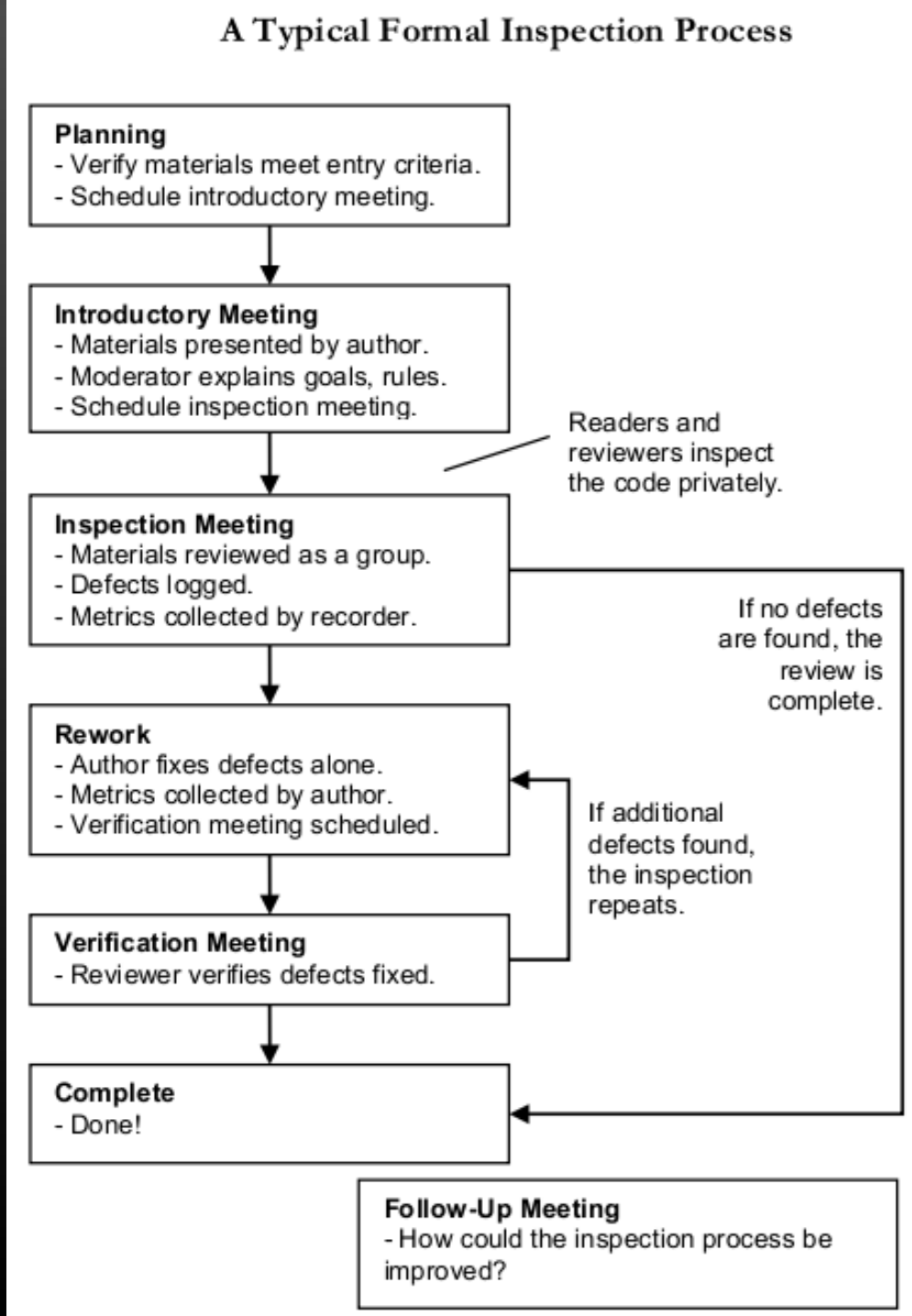


More info:
Wiegers, K. E. (2002). Seven truths about peer reviews. Cutter IT Journal, 15(7), 31-37.

The code review lifecycle



See also: Cohen's book on code review



In: P. Farrell-Vinay, Manage Software Testing. Taylor & Francis, 2008.

Formal technical review [R. Pressman]

Objectives

- to uncover errors in function, logic, or implementation for any representation of the software;
- to verify that the software under review meets its requirements;
- to ensure that the software has been represented according to predefined standards;
- to achieve software that is developed in a uniform manner;
- to make projects more manageable.

FTR serves as a training ground

junior engineers → observe different approaches to software analysis, design, and implementation.

Lightweight techniques for code review

Over-the-shoulder

a developer stands over the author's shoulder as the latter walks through the code changes.

Email pass-around

The author (or SCM system) emails code to reviewers

Pair Programming

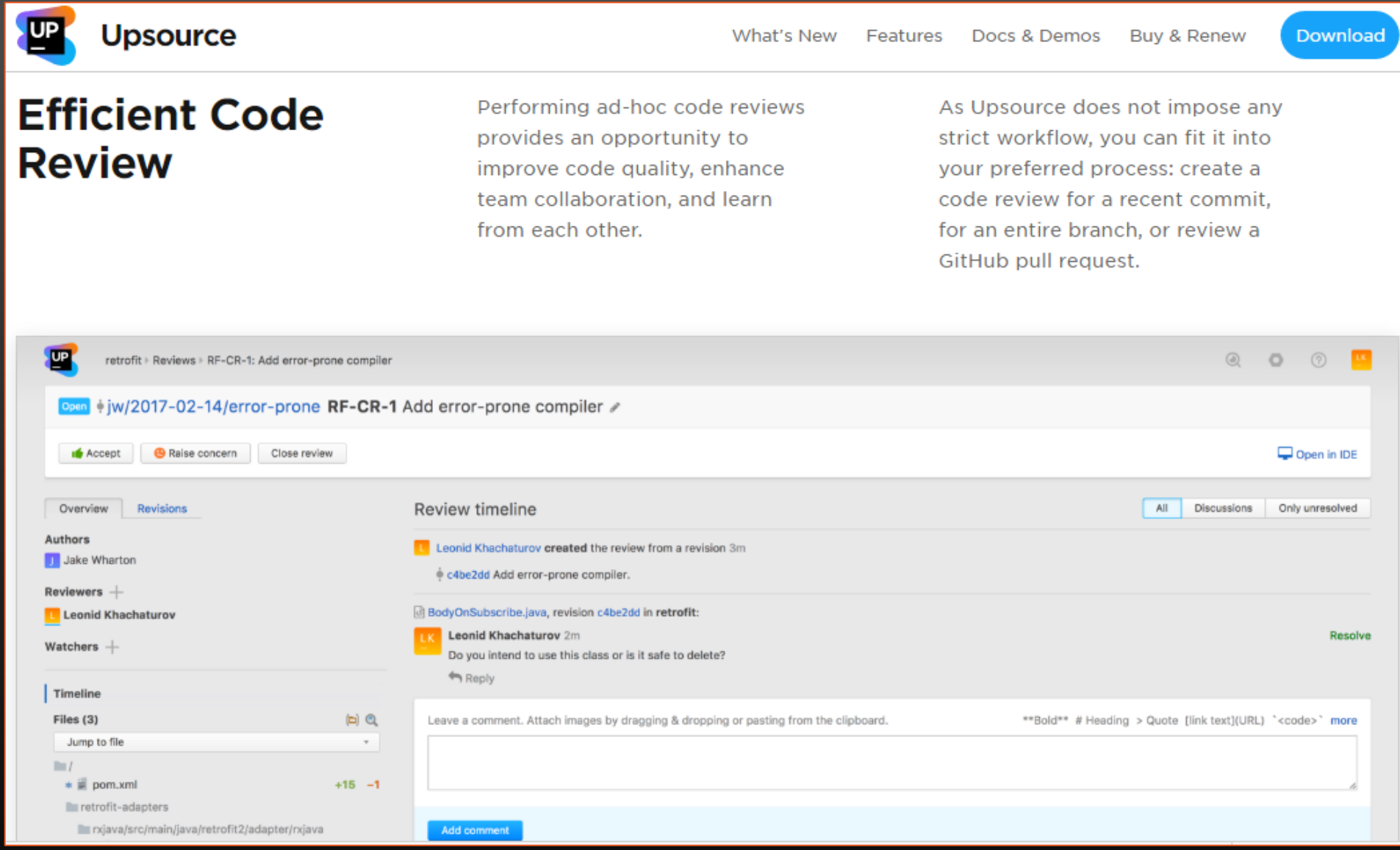
Two authors develop code together at the same workstation.

Tool-assisted reviews

Authors and reviewers use specialized tools designed for peer code review.

Collect changes, support discussions, visualize diffs,…

E.g.: Collaborator , Guerrit, Upsource



http://www.jetbrains.com/upsource/

https://www.atlassian.com/software/crucible

https://smartbear.com/product/collaborator/overview/

Pair programming must be done right to be effective and productive

Pairs are short-lived

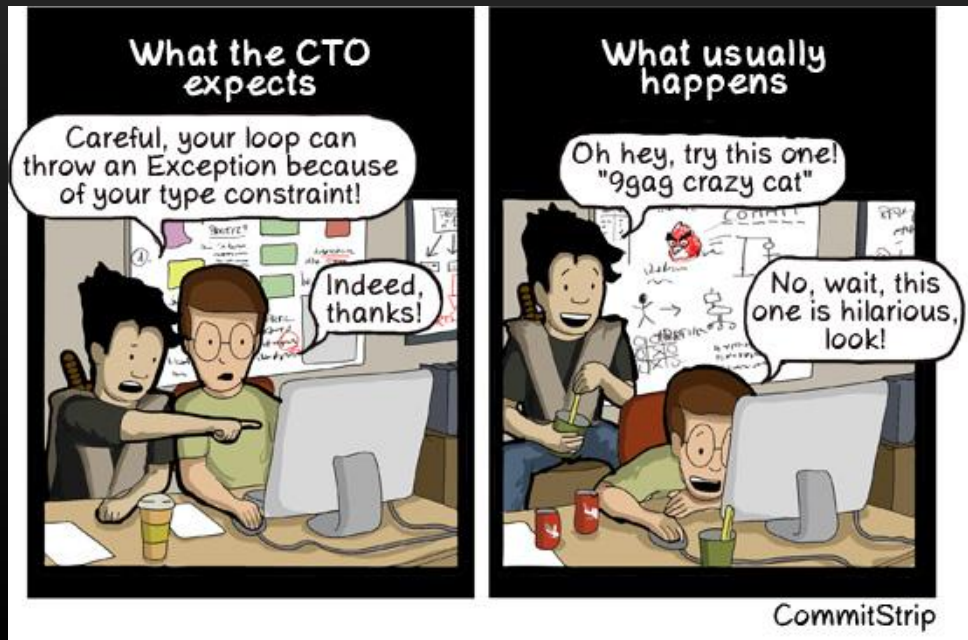
"Half of the time", one is working on his own tasks (and then swap)

You can't check in production code that you have written on your own.

Excellent way to train a new team member in the existing code

Newbies should pair most often with team members with more seniority…

https://developer.atlassian.com/blog/2015/05/try-pair-programming/



Styles of code reviews

Pre-commit review

E.g.: discuss the changes with email, authorized maintainers will commit

Not integrated in the history; only one author for a feature/patch

Post-commit review

Diffs (added and removed lines)

Review a single commit or a group of commits

Guerrit-style

https://www.gerritcodereview.com/

Specific workflow

Fetch, push to staging branch, vote (score)

Pull request (in Git)

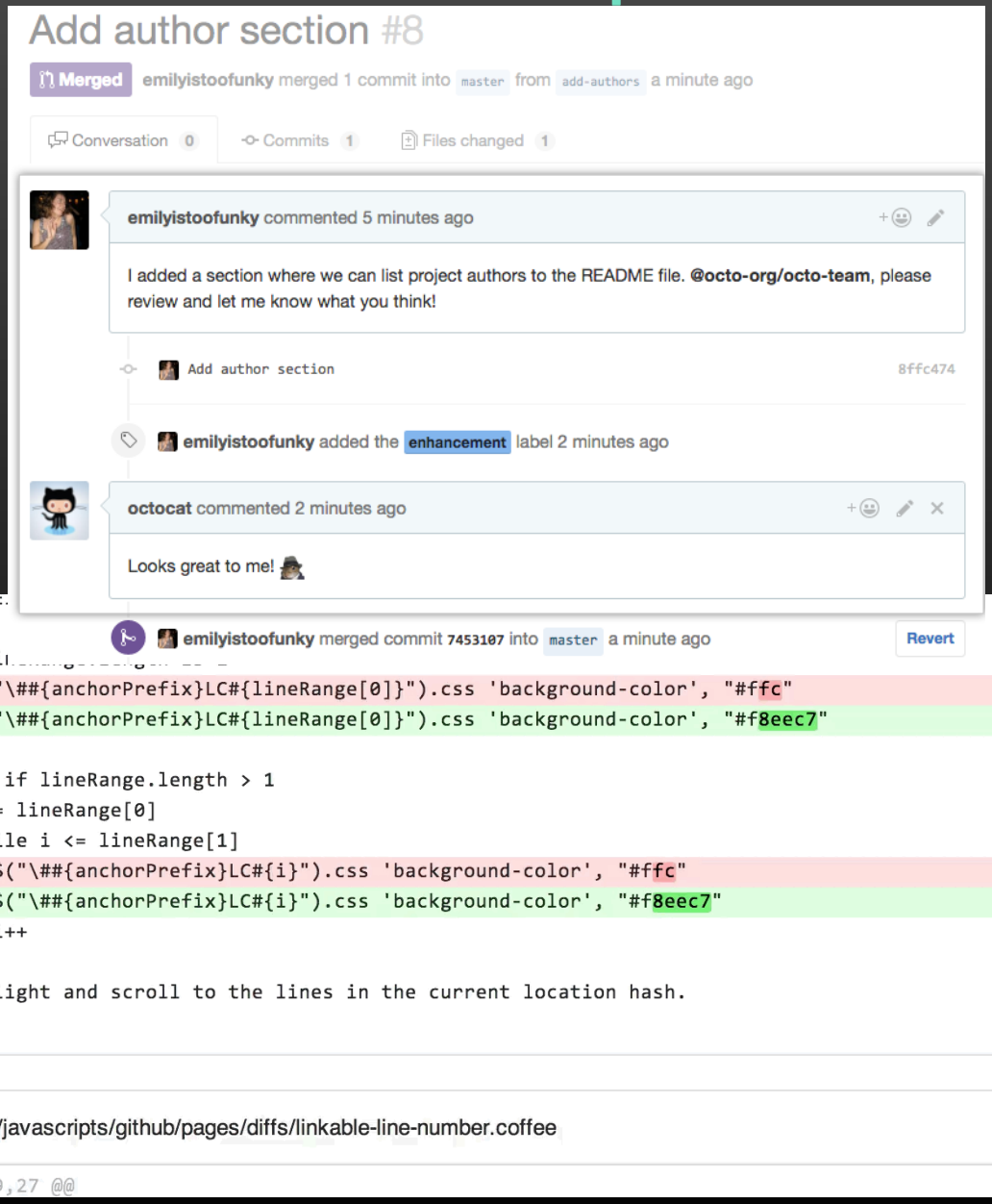
Review an unmerged branch before merge

Different merge strategies

Check whether build passes

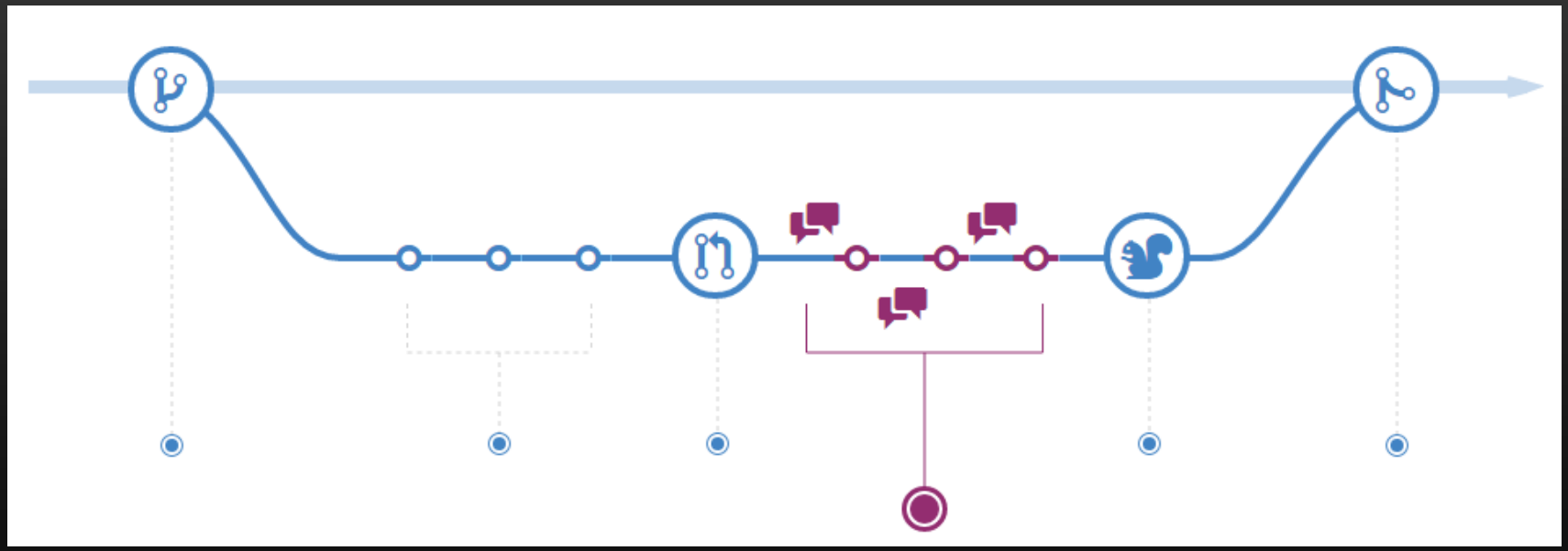
Integrate code review in the workflow with pull requests

All merge requests (ak.a. pull requests) […], whether written by a team member or a volunteer contributor, must go through a code review process to ensure the code is effective, understandable, and maintainable.



Understanding the GitHub Flow

5 minute read Download PDF version



https://docs.github.com/en/get-started/quickstart/github-flow

In detail: http://scottchacon.com/2011/08/31/github-flow.html

Code review "etiquette" ("a set of rules about behaviour for people in a particular profession/social situations")

https://github.com/thoughtbot/guides/tree/master/code-review

Code Review

A guide for reviewing code and having your code reviewed. Watch a presentation that covers this material from Derek Prior at RailsConf 2015.

Everyone

- Accept that many programming decisions are opinions. Discuss tradeoffs, which you prefer, and reach a resolution quickly.
- Ask good questions; don't make demands. ("What do you think about naming this `:user_id`?")
- Good questions avoid judgment and avoid assumptions about the author's perspective.
- Ask for clarification. ("I didn't understand. Can you clarify?")
- Avoid selective ownership of code. ("mine", "not mine", "yours")
- Avoid using terms that could be seen as referring to personal traits. ("dumb", "stupid"). Assume everyone is intelligent and well-meaning.
- Be explicit. Remember people don't always understand your intentions online.
- Be humble. ("I'm not sure - let's look it up.")
- Don't use hyperbole. ("always", "never", "endlessly", "nothing")
- Don't use sarcasm.
- Keep it real. If emoji, animated gifs, or humor aren't you, don't force them. If they are, use them with aplomb.
- Talk synchronously (e.g. chat, screensharing, in person) if there are too many "I didn't understand" or "Alternative solution:" comments. Post a follow-up comment summarizing the discussion.

Having Your Code Reviewed

- Be grateful for the reviewer's suggestions. ("Good call. I'll make that change.")
- A common axiom is "Don't take it personally. The review is of the code, not you." We used to include this, but now prefer to

RULE 1

Do the code reviews before deployment. Your team will end up, on average, spending 7 percentage points% more of its time on building new features compared with those who do after, and 10 percentage points% more than those who don't do code reviews at all.

RULE 2

Make sure all your developers get to review code. This will improve the feeling of empowerment, facilitate knowledge transfer, and improve developer satisfaction and productivity.

RULE 3

The optimal amount of time to spend on code reviews is between 0.5 to 1 day per week per developer.

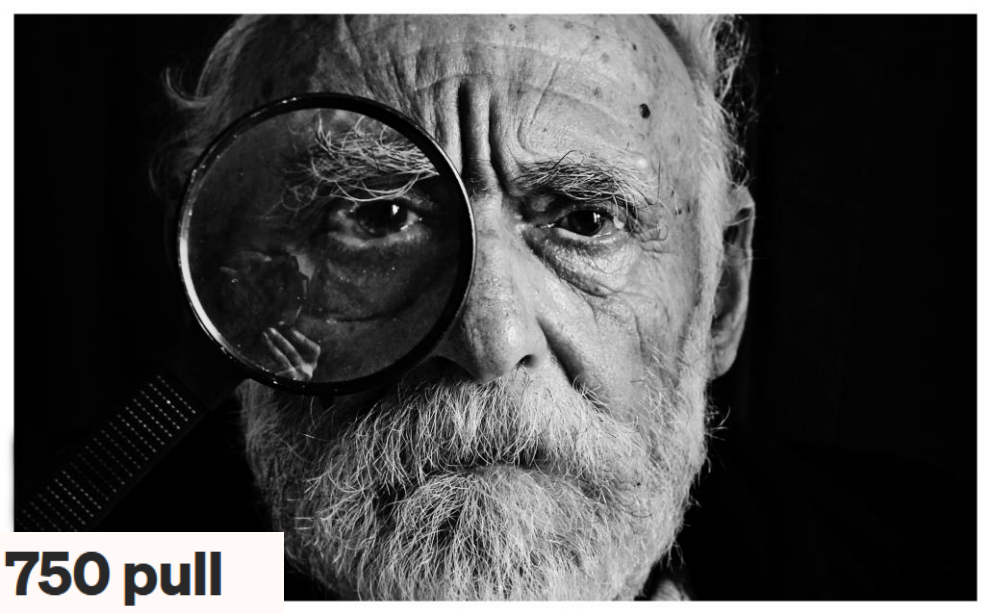
RULE 4

Make code reviews blocking, that is, don't deploy before they have been carried out.

RULE 5

Be strict and thorough when reviewing code. Your code quality and velocity will thank you.

https://www.codacy.com/ebooks/guide-to-code-reviews



I've code reviewed over 750 pull requests at Amazon. Here's my exact thought process.

Curtis Einsmann Nov 10, 2020 · 4 min read

https://curtiseinsmann.medium.com/ive-code-reviewed-over-750-pull-requests-at-amazon-here-s-my-exact-thought-process-cec7c942a3a4

AI-assisted

Prompt: *Write a code review for the code below. Keep a focus on the maintainability of the code, potential security issues, and performance flaws. {code}*

Prompt: *I added a new search feature to the application that filters results based on user input. Can you help me write a PR description for this?*

Prompt: *For this program, suggest unit tests to correctly calculate the total amount, check the typical and edge cases, such as zero and negative bill amounts and high tip values. Also check for invalid inputs. For the unit tests, you can have console logs.*

Commit messages

Make the commit message concise but informative: The commit message should be short and to the point, but also provide enough information for others (and future you) to understand what changes were made and why.

Use the imperative mood: Start your commit messages in the imperative mood, as if you are giving a command or instruction. For example, use “add book search" instead of "Added feature" or "Adds feature".

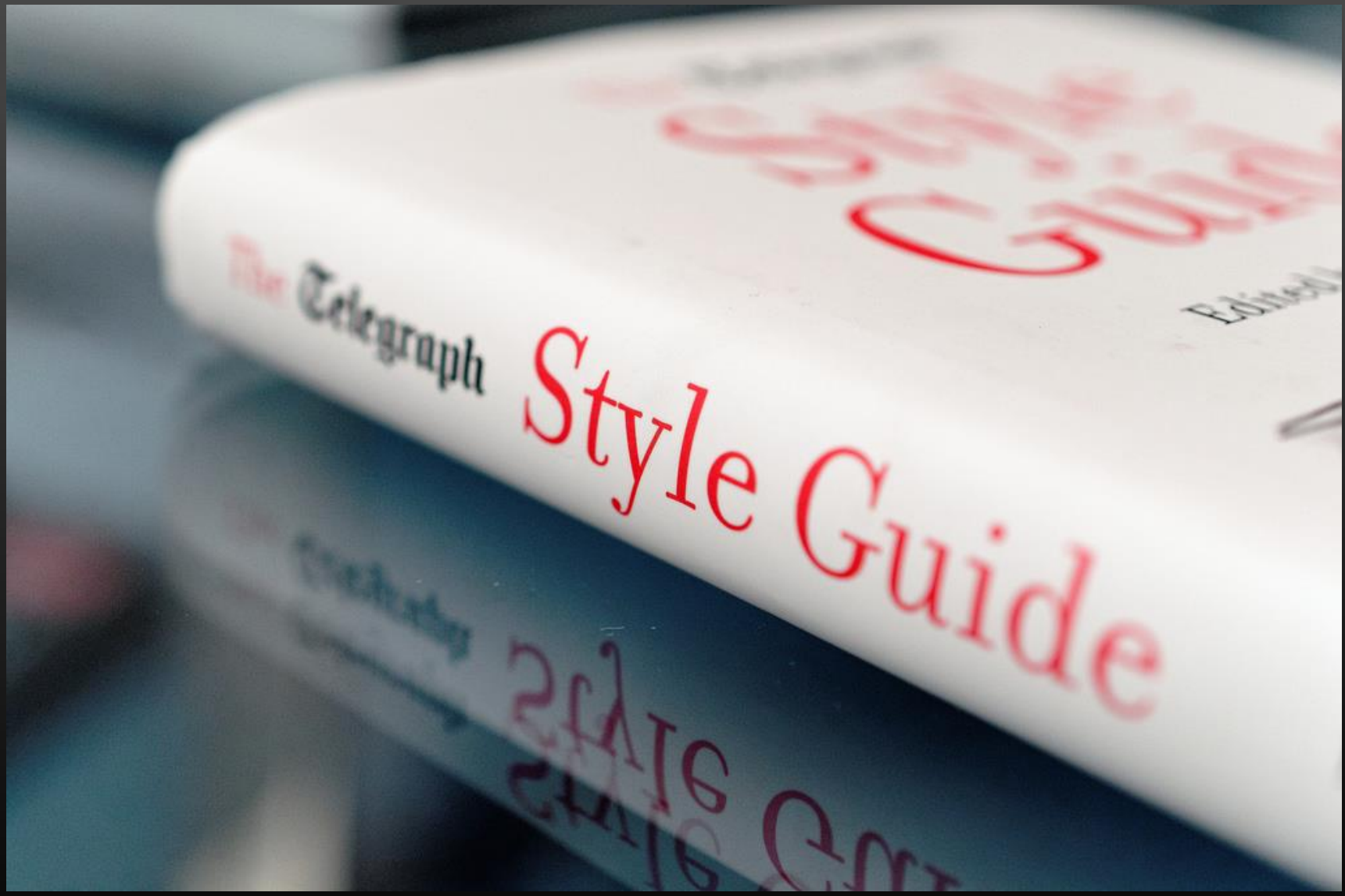
First line should be a summary: The first line of the commit message should be a summary of the changes, followed by a blank line, then a more detailed explanation if necessary. The first line should be limited to 50 characters and the following lines should be wrapped at 72 characters.

Include the context of the change: Explain why the change is necessary. This helps other developers understand why a particular modification was made.

Reference related work: If your commit is related to a bug, issue, or story, include its ID in the commit message. This helps create a connection between your commit and the related work.

Avoid vague messages: Commit messages like "fix bug" or "update code" are not helpful because they don't provide enough context. Be specific about what bug was fixed or what part of the code was updated.

https://github.com/joelparkerhenderson/git-commit-message


The Telegraph
Style Guide

Code style **improves readability**

Major references

Google coding styles

Mozilla Coding Style

Code style for projects:

Android open-source project (Good source for Java developers)

Code style for Chromium open source (after Google C++ style)

Java

Original conventions for Java

Linux

Kernel coding style

The dark side of pair programming.

References

Cohen. 2012. Best Kept Secrets of Code Review, SmartBear contributed book .

Bloch, Joshua. 2008. Effective Java. 2nd ed. Addison-Wesley Professional. http://books.google.pt/books?id=ka2VUBqHiWkC.

Fowler, Martin. 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional. http://books.google.com/books?id=1MsETFPD3I0C&pgis=1

Martin, Robert C. 2008. Clean Code: A Handbook of Agile Software Craftsmanship (Google eBook). Pearson Education. http://books.google.com/books?id=_i6bDeoCQzsC&pgis=1.

R. Pressman, “Software Engineering: A Practitioner’s Approach,” Jan. 2009.