

45426: Teste e Qualidade de Software

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

Ilídio Oliveira

v2025-04-01

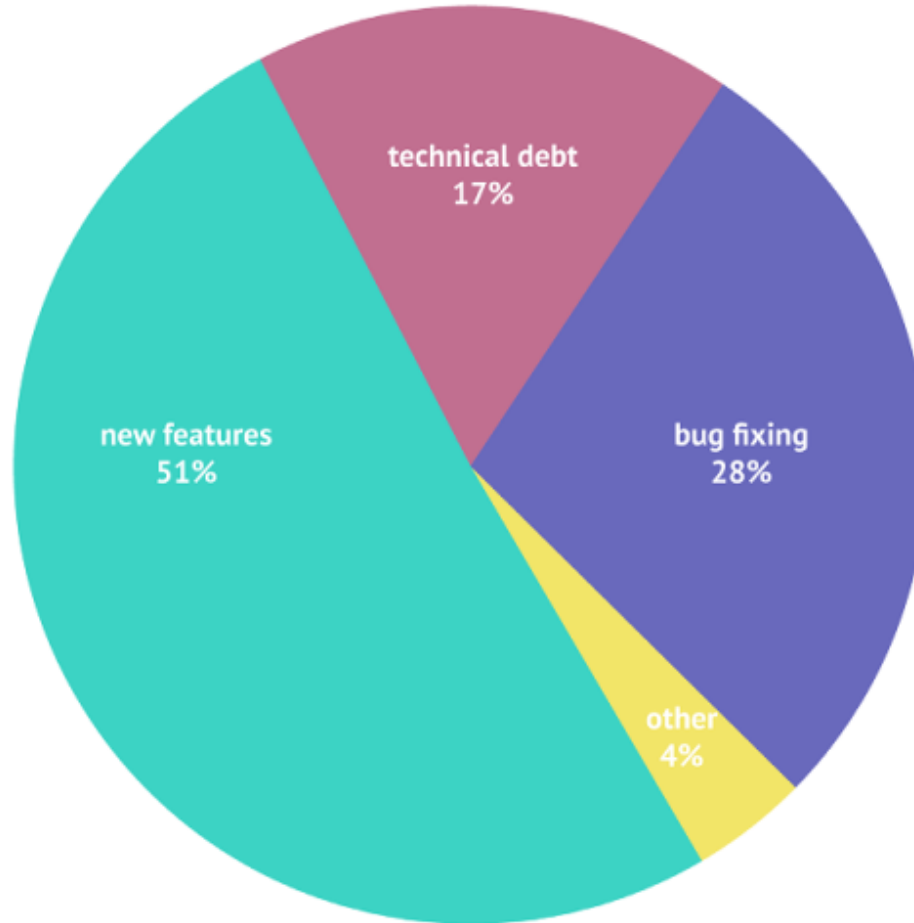
# Learning objectives

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

# Survey among +600 developers

## Time spent on

- new features
- technical debt
- bug fixing
- other



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

# Clean Code

A Handbook of Agile Software Craftsmanship

Foreword by James O. Coplien

Robert C. Martin

## 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

*AI “Co-piloting”*

...

Culture for clean/readable code

Developer/style guidelines

Definition of Done

Patterns (reuse)

...

# Code review practice

Code review is careful, systematic study of source code by people who are not the original author of the code.

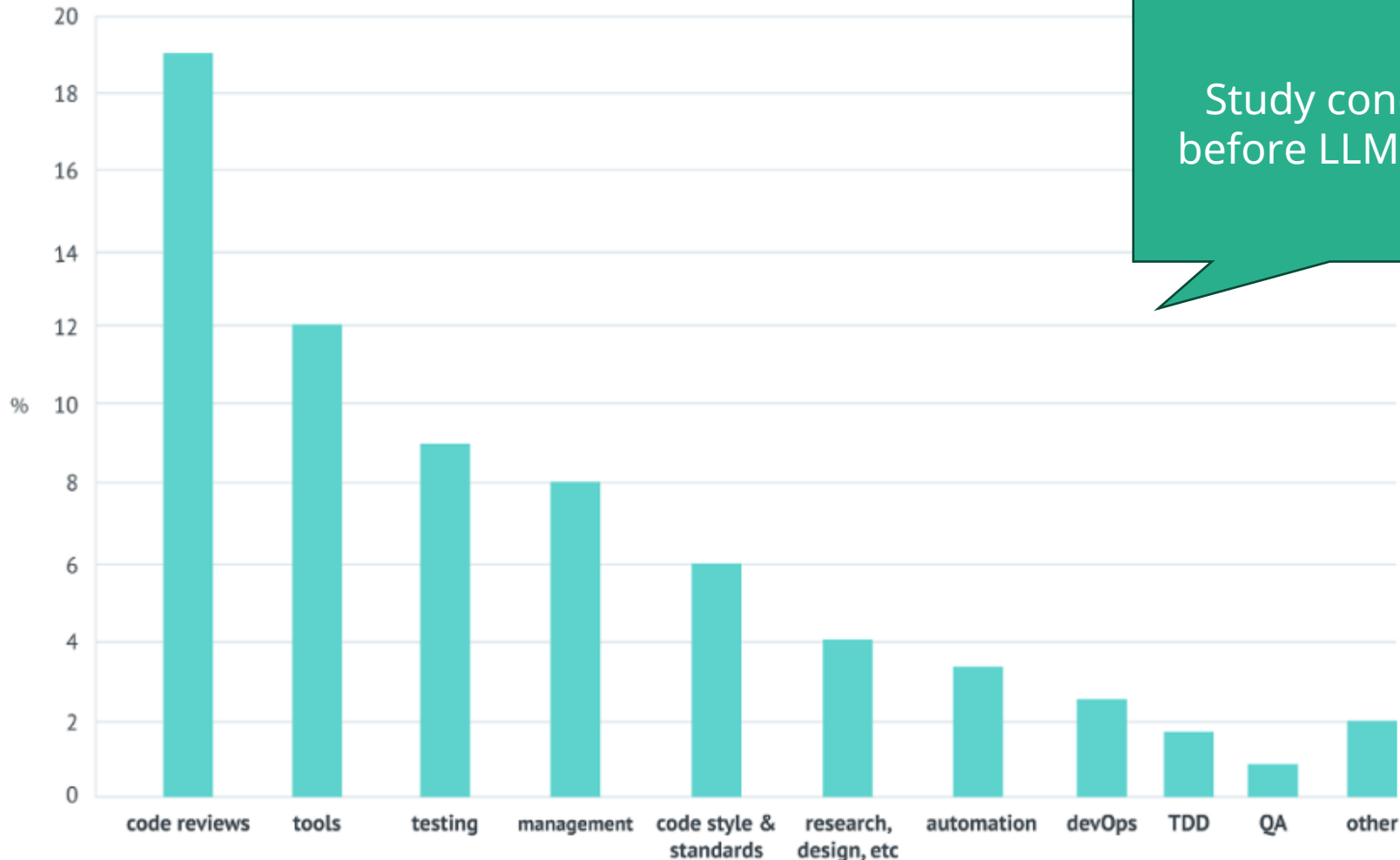
It's analogous to proofreading a term paper.

<https://ocw.mit.edu/ans7870/6/6.005/s16/classes/04-code-review/>

Code review really has two purposes:

- **Improving the code.** Finding bugs, anticipating possible bugs, checking the clarity of the code, and checking for consistency with the project's style standards.
- **Improving the programmer.** Code review is an important way that programmers learn and teach each other, about new language features, changes in the design of the project or its coding standards, and new techniques. In open source projects, particularly, much conversation happens in the context of code reviews.

## 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.

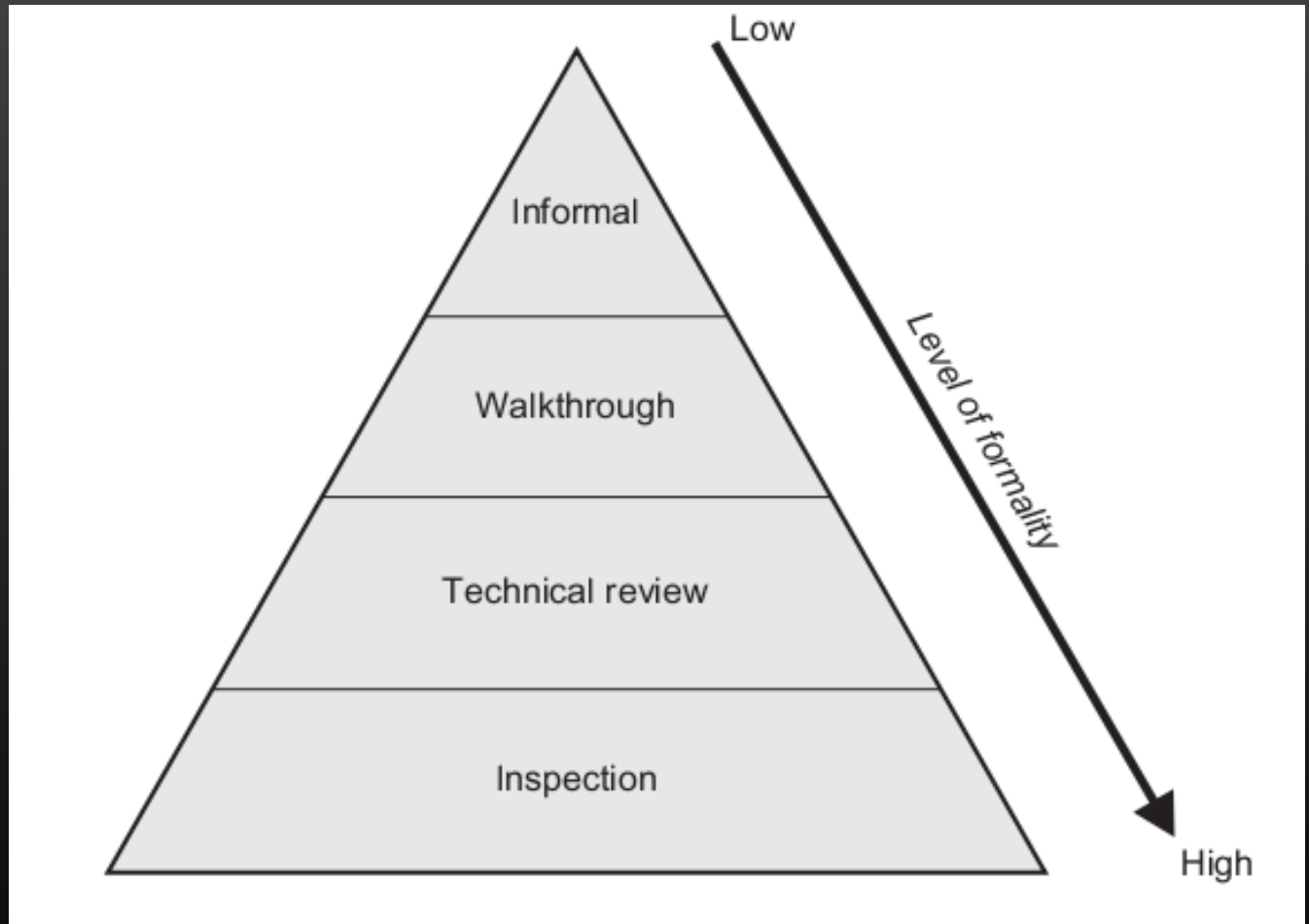
# Issues (likely to address) in a code review

- **Verify feature/story 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)**
- **In-house policies compliance**

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



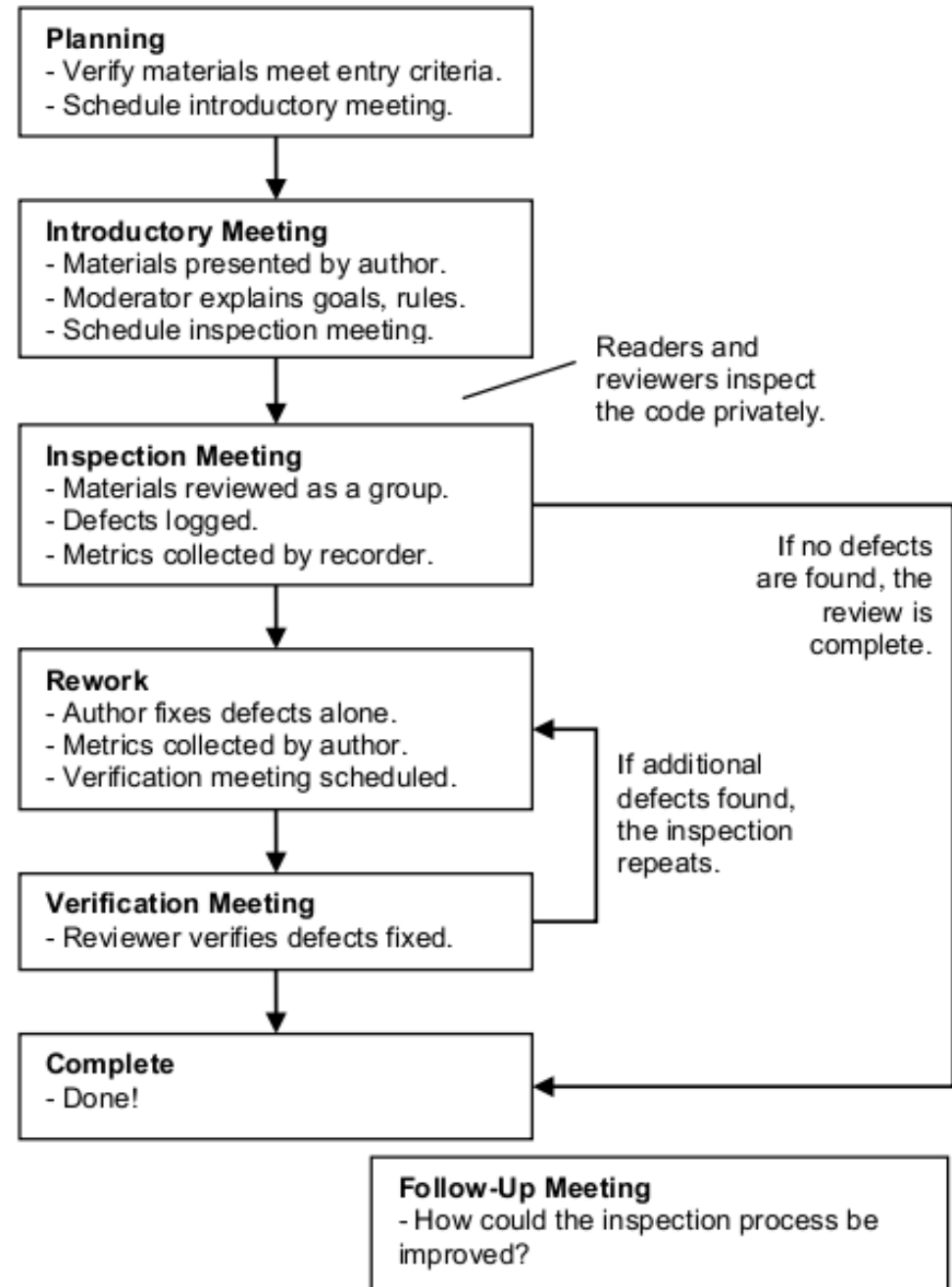
# A review process can have very different levels of formality (Informal to Formal Tech Review)



More info:

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

## A Typical Formal Inspection Process



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.



junior developers  
should care about  
reviews

# 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.

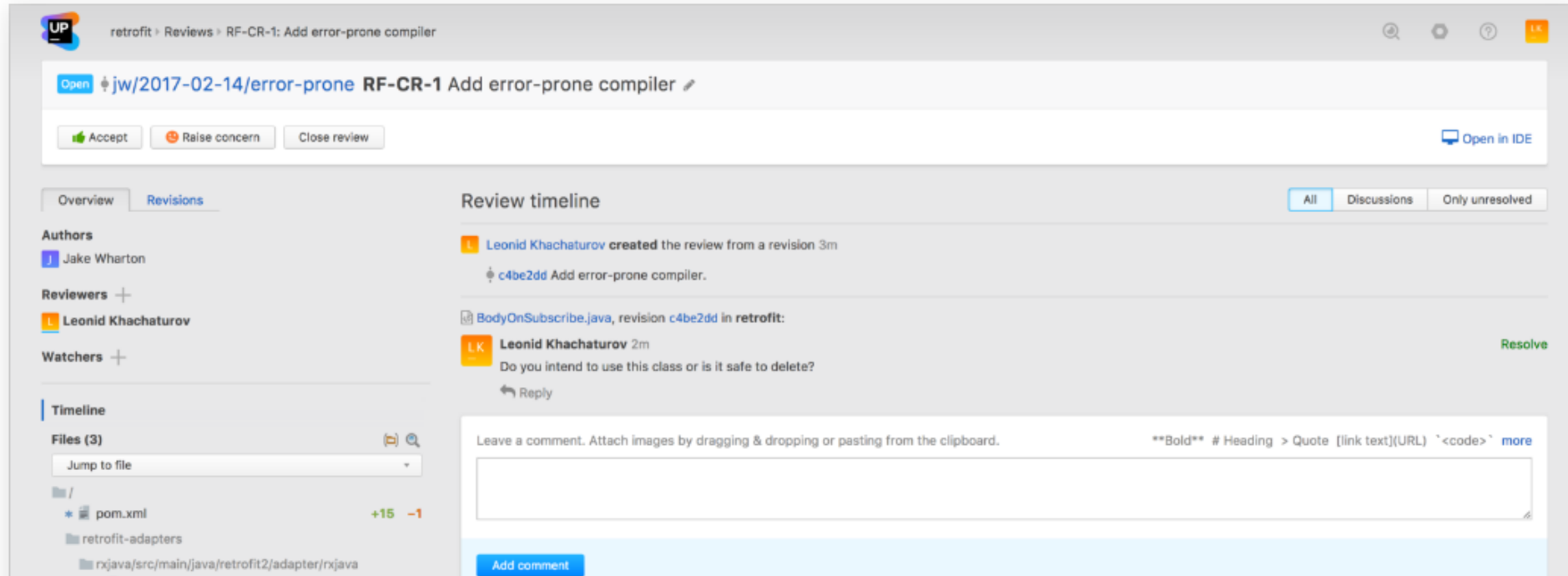
Collect changes, support discussions, visualize diffs,...

E.g.: [Collaborator](#) , [Guerit](#),  
[Upsource](#) , [Crucible](#)

# Efficient Code Review

Performing ad-hoc code reviews provides an opportunity to improve code quality, enhance team collaboration, and learn from each other.

As Upsource does not impose any strict workflow, you can fit it into your preferred process: create a code review for a recent commit, for an entire branch, or review a GitHub pull request.



The screenshot shows the Upsource interface for a code review. At the top, the breadcrumb is 'retrofit > Reviews > RF-CR-1: Add error-prone compiler'. Below this is a bar with an 'Open' button, a link to 'jw/2017-02-14/error-prone', and the review title 'RF-CR-1 Add error-prone compiler'. Action buttons include 'Accept', 'Raise concern', and 'Close review', along with an 'Open in IDE' link.

On the left sidebar, there are sections for 'Authors' (Jake Wharton), 'Reviewers' (Leonid Khachaturov), and 'Watchers'. Below these is a 'Timeline' section with a 'Files (3)' list: 'pom.xml' (+15, -1), 'retrofit-adapters', and 'rxjava/src/main/java/retrofit2/adapters/rxjava'.

The main area is titled 'Review timeline' with tabs for 'All', 'Discussions', and 'Only unresolved'. It shows a timeline entry where Leonid Khachaturov created the review from a revision 3m ago, with a commit link 'c4be2dd' and the message 'Add error-prone compiler.' Below this is a discussion entry from Leonid Khachaturov asking 'Do you intend to use this class or is it safe to delete?' with a 'Reply' button.

At the bottom, there is a comment box with a placeholder 'Leave a comment. Attach images by dragging & dropping or pasting from the clipboard.' and a 'Add comment' button. A rich text toolbar is visible above the box.

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

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

# Pair programming must be done right to be effective and productive

Pairs are short-lived

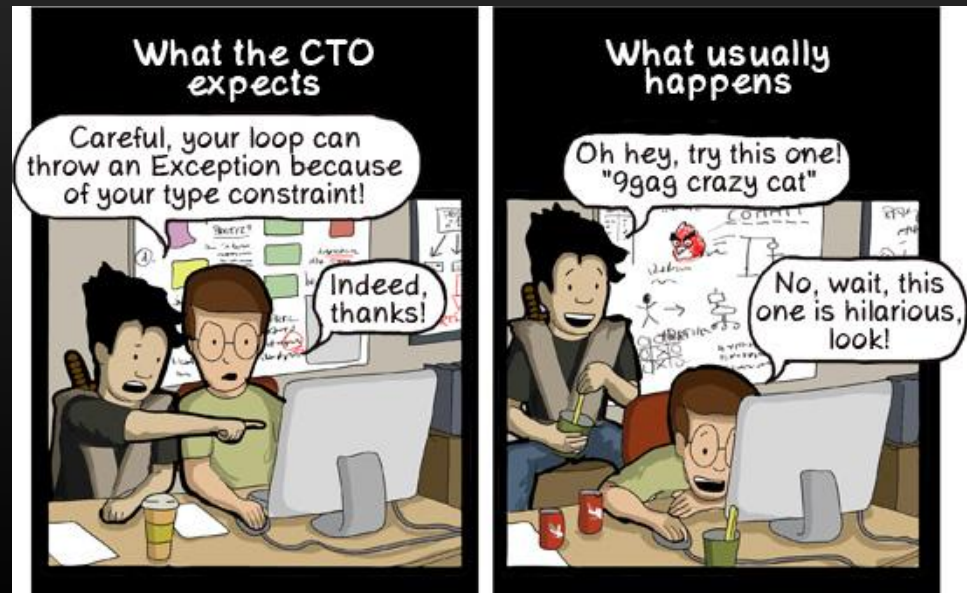
“Half of the time”, one is working on his own tasks (and then swap)

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/>



CommitStrip

# 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

## Gerrit-style

Specific workflow

Fetch, push to staging branch, vote (score)

<https://www.gerritcodereview.com/>

## Pull request (in Git)

Review an unmerged branch before merge

Different merge strategies

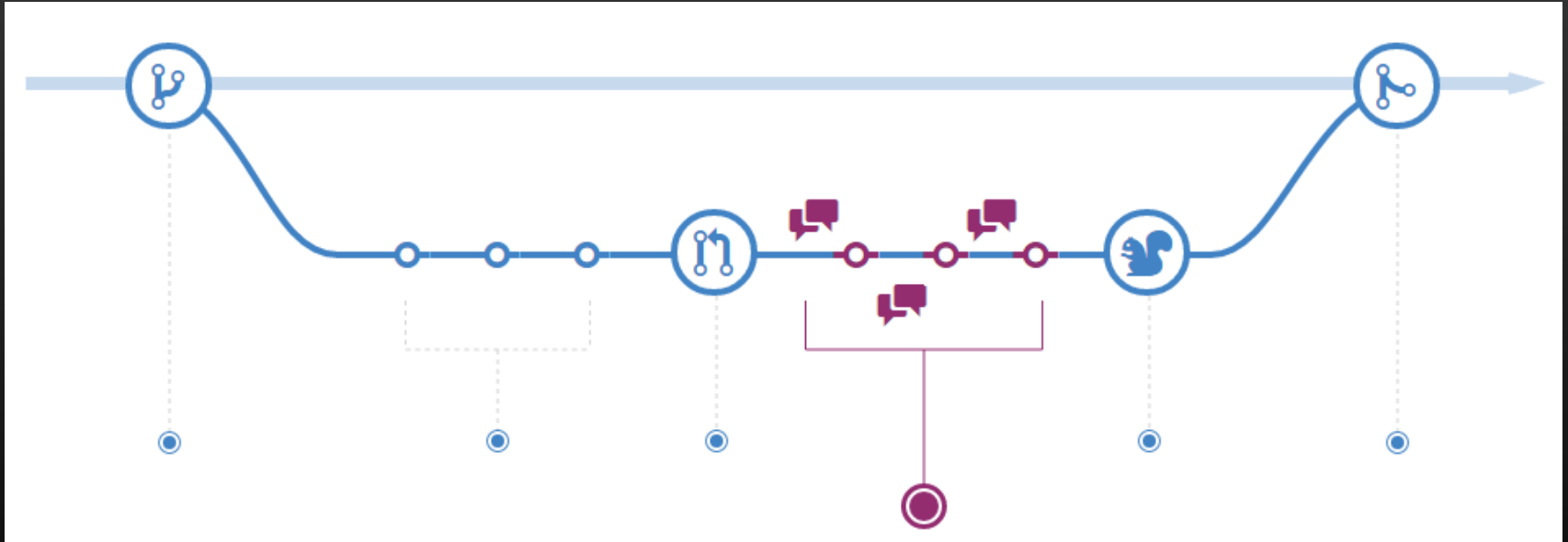
Check whether build passes



# Understanding the GitHub Flow

🕒 5 minute read

📄 Download PDF version



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



# 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.

The screenshot displays a GitHub pull request titled "Add author section #8". At the top, it indicates that "emilyistoofunky" merged 1 commit into the master branch from the "add-authors" branch a minute ago. Below this, there are tabs for "Conversation" (0), "Commits" (1), and "Files changed" (1). The "Conversation" tab is active, showing a comment from "emilyistoofunky" stating: "I added a section where we can list project authors to the README file. @octo-org/octo-team, please review and let me know what you think!". Below the comment is a commit diff for "Add author section" (8ffc474). Another comment from "octocat" says: "Looks great to me!". At the bottom of the pull request, it shows that "emilyistoofunky" merged commit 7453107 into master a minute ago, with a "Revert" button. Below the pull request, a code diff for the file "app/assets/javascripts/github/pages/diffs/linkable-line-number.coffee" is shown. The diff highlights changes to line numbers and CSS background colors. The code includes a function to highlight and scroll to lines in the current location hash.

```
115 115 lineE.  
116 116  
117 117 if li  
118 - $(`###{anchorPrefix}LC#{lineRange[0]}`).css 'background-color', "#ffc"  
118 + $(`###{anchorPrefix}LC#{lineRange[0]}`).css 'background-color', "#f8eec7"  
119 119  
120 120 else if lineRange.length > 1  
121 121 i = lineRange[0]  
122 122 while i <= lineRange[1]  
123 - $(`###{anchorPrefix}LC#{i}`).css 'background-color', "#ffc"  
123 + $(`###{anchorPrefix}LC#{i}`).css 'background-color', "#f8eec7"  
124 124 i++  
125 125  
126 126 # Highlight and scroll to the lines in the current location hash.
```

24 app/assets/javascripts/github/pages/diffs/linkable-line-number.coffee

# Code review “etiquette” (“a set of rules about behavior 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**

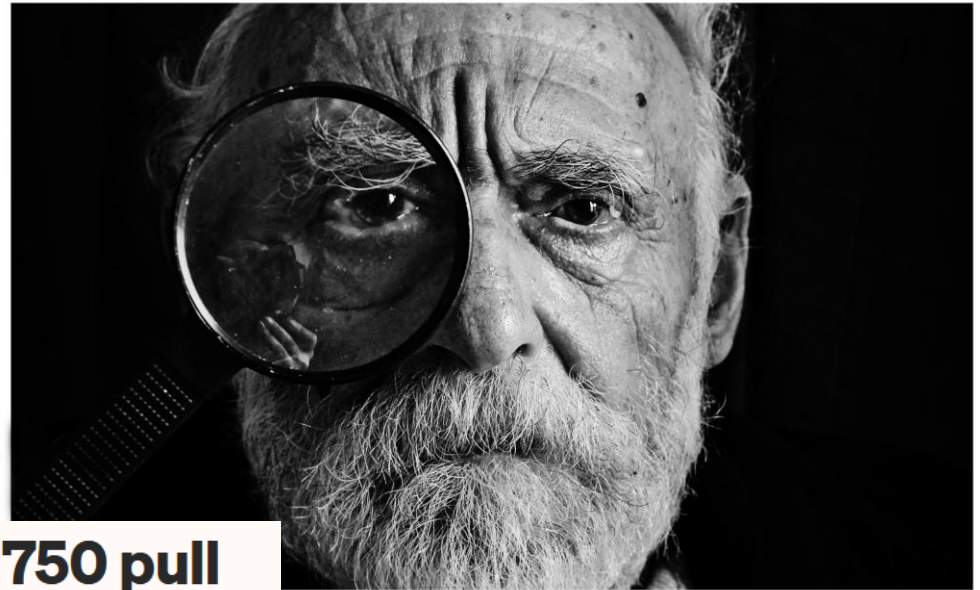
The optimal amount of time to spend on code reviews is between 0.5 to 1 day per week per developer.

## **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.



## **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.*

# Using GitHub Copilot code review

Learn how to request a code review from GitHub Copilot.

## Who can use this feature?

📄 See the table below.

Visual Studio Code

**Web browser**

### Note

- GitHub Copilot code review is in public preview and subject to change.
- The [GitHub Pre-release License Terms](#) apply to your use of this product.

## Receiving access

If you get a Copilot subscription from an organization, you will only be able to participate in the public preview on the GitHub website if an owner of your organization has enabled **Copilot in GitHub.com > Opt in to preview features** in the organization policies. See [Managing policies for Copilot in your organization](#).

If an enterprise owner has explicitly enabled or disabled preview features, organizations in the enterprise will not be able to change the policy. See [Managing policies and features for Copilot in your enterprise](#).

<https://docs.github.com/en/copilot/using-github-copilot/code-review/using-copilot-code-review>

# Configuring coding guidelines for GitHub Copilot code review

Learn how to customize Copilot code review with custom coding guidelines.

## Note

The custom coding guidelines feature is only available as part of a subscription to GitHub Copilot Enterprise, and is currently only available to selected participants in the public preview of Copilot code review.

## About coding guidelines

You can customize Copilot code review with custom coding guidelines written in natural language. For more information on Copilot code review, see [Using GitHub Copilot code review](#).

With coding guidelines, Copilot can give feedback based on your organization's specific coding style and best practices.

Because Copilot code review is powered by a large language model, it can help with enforcing coding guidelines that are not covered by your linter or static analysis tool.

Coding guidelines are configured at the repository level. You can create and enable up to 6 coding guidelines per repository.

## Note

- Coding guidelines only work with languages supported by Copilot code review. For a list of supported languages, see [Using GitHub Copilot code review](#).
- Coding guidelines only apply to code reviews carried out by Copilot. The guidelines do not

<https://docs.github.com/en/copilot/using-github-copilot/code-review/configuring-coding-guidelines>

# 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 changed and why.

**Use the imperative mood:** as if you are giving a command or instruction, e.g.: "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.

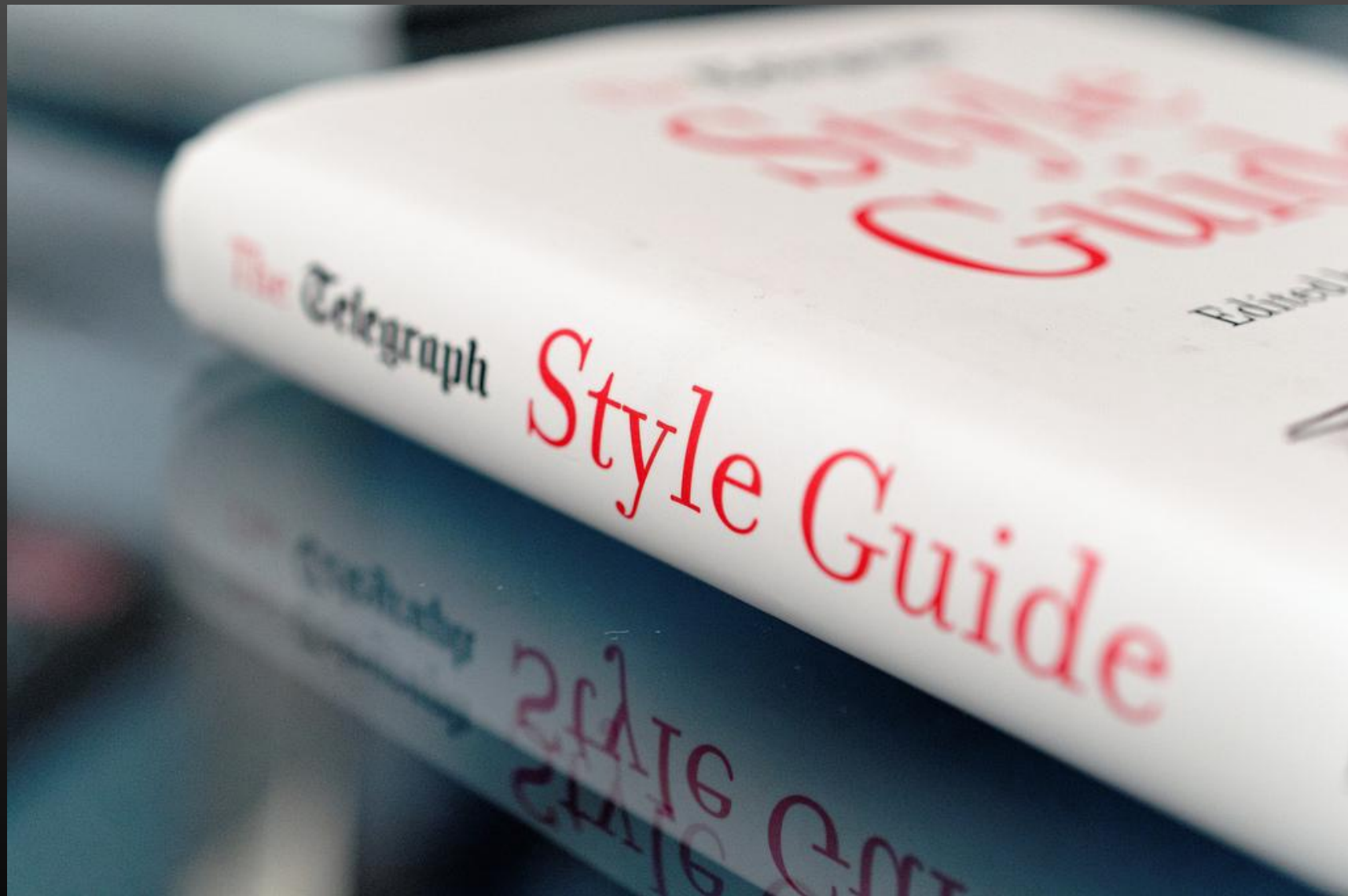
**Include the context of the change:** Explain why the change is necessary. This helps other developers understand why a particular modification was made.

**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... Be specific about what bug was fixed or what part of the code was updated.

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





# Code style improves readability

## Major references

[Google](#) coding styles

## Code style for projects:

[Android open-source project](#) (Good source for Java developers)

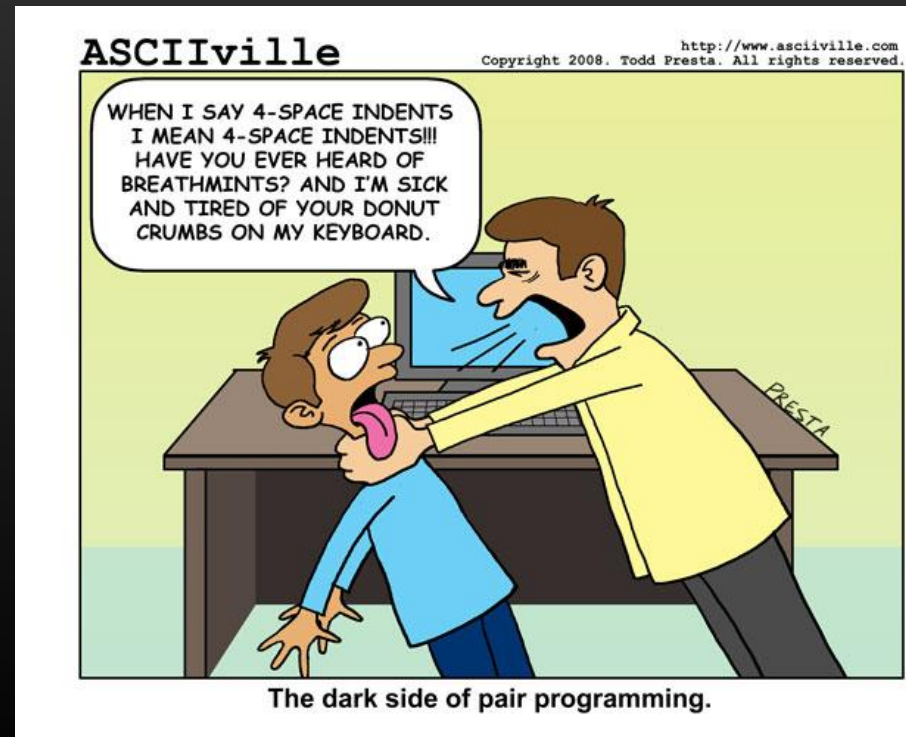
Code style for [Chromium open source](#) (after [Google C++ style](#))

## Java

Original [conventions for Java](#)

## Linux

[Kernel coding style](#)



# 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=1MsETFPD3IOC&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.