ECE444: Software Engineering

Inspections, Code Reviews

Shurui Zhou

The Edward S. Rogers Sr. Department
of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

Learning Goals

- Understand different forms of peer reviews with different formality levels.
- Engage in constructive modern code review using a typical commit review system.
- Describe the benefits and properties of good checklists in code review.

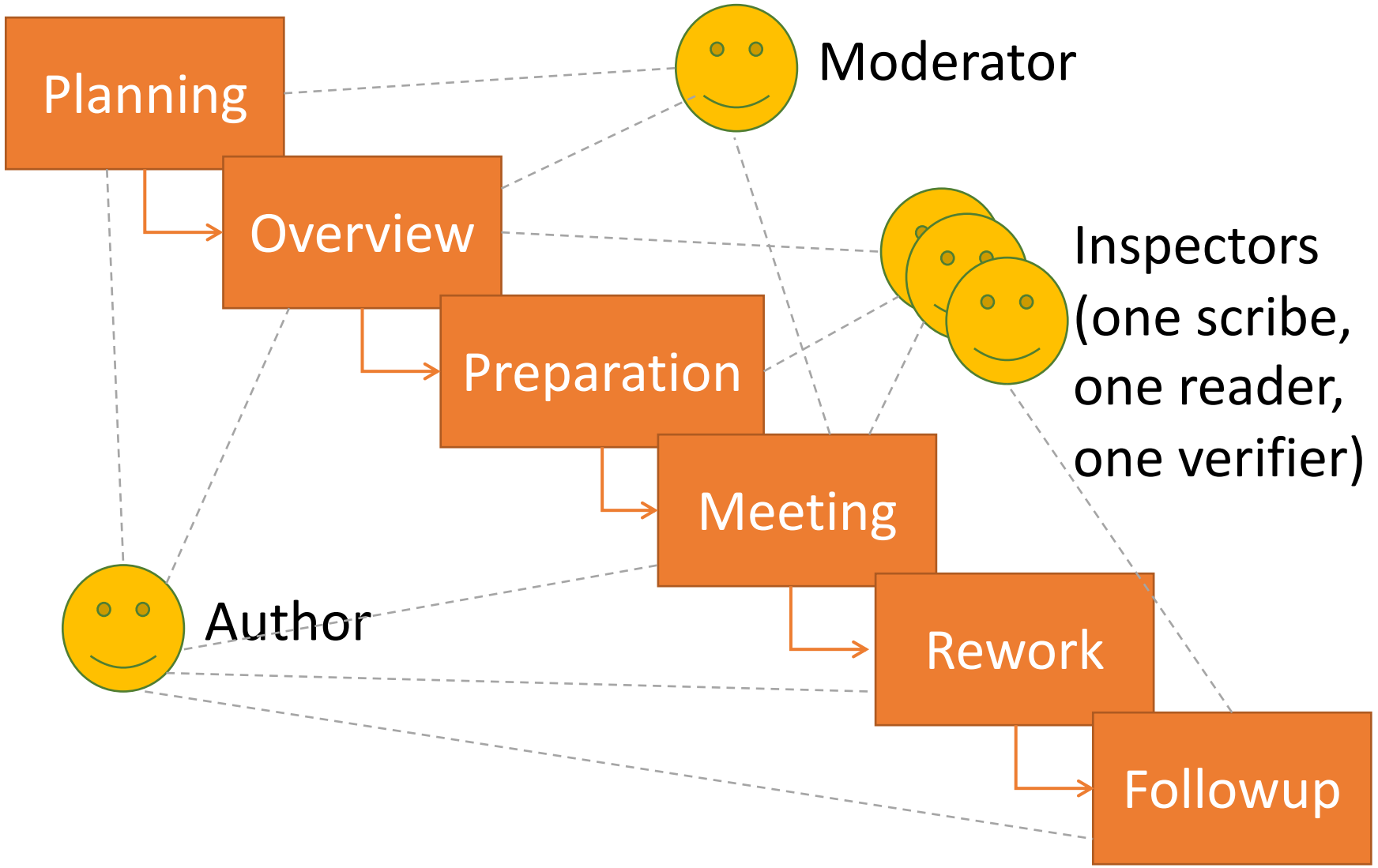
Formal Inspections

Formal Inspections

- Idea popularized in 70s at IBM
- Group of developers meets to formally review code or other artifacts
- Most effective approach to find bugs
  - Typically 60-90% of bugs found with inspections
- Expensive and labor-intensive

Inspection Team and Roles

- Typically 4-5 people (min 3)
- Author
- Inspector(s)
  - Find faults and broader issues
- Reader
  - Presents the code or document at inspection meeting
- Scribe
  - Records results
- Moderator
  - Manages process, facilitates, reports


Planning
Overview
Preparation
Meeting
Rework
Followup
Moderator
Inspectors
(one scribe,
one reader,
one verifier)
Author

Checklists

- Reminder what to look for
- Include issues detected in the past
- Preferably focus on few important items
- Examples:
  - Are all variables initialized before use?
  - Are all variables used?
  - Is the condition of each if/while statement correct?
  - Does each loop terminate?
  - Do function parameters have the right types and appear in the right order?
  - Are linked lists efficiently traversed?
  - Is dynamically allocated memory released?
  - Can unexpected inputs cause corruption?
  - Have all possible error conditions been handled?
  - Are strings correctly sanitized?

Process details

- Authors do not explain or defend the code – not objective
  - Author != moderator, != scribe, !=reader
  - Author should still join the meeting to observe questions and misunderstandings and clarify issues if necessary
- Reader (optional) walks through the code line by line, explaining it
  - Reading the code aloud requires deeper understanding
  - Verbalizes interpretations, thus observing differences in interpretation

Modern Code Review

Closed **v4-dev updated nuspec for content files** #30147

supergibbs wants to merge 5 commits into `twbs:v4-dev` from `supergibbs:v4-dev-updated-nuspec-cont…`

XhmikosR reviewed 5 days ago — View changes

`nuget/bootstrap.nuspec` — Hide resolved

```
...  ...  @@ -2,16 +2,16 @@
 2    2     <package xmlns="http://schemas.microsoft.com/packaging/2011/08/nuspec.
 3    3       <metadata>
 4    4         <id>bootstrap</id>
 5       -     <version>4.4.1</version>
      5     +     <version>4</version>
```

XhmikosR 5 days ago **Member** — +😀 Tip •••

What's the rationale behind not using the full version here?

supergibbs 5 days ago Author Contributor — +😀 Tip •••

I added a comment to the file; the version is pulled from package.json so no need to maintain this

Reply...

`nuget/bootstrap.nuspec` Outdated — Show resolved

XhmikosR commented 5 days ago **Member** — +😀 Tip •••

We should just land one patch in master, cherry pick it and when v5.0.0 is out, we just change the version number in master.

None yet

Milestone

No milestone

Linked issues

Successfully merging this pull request may close these issues.

None yet

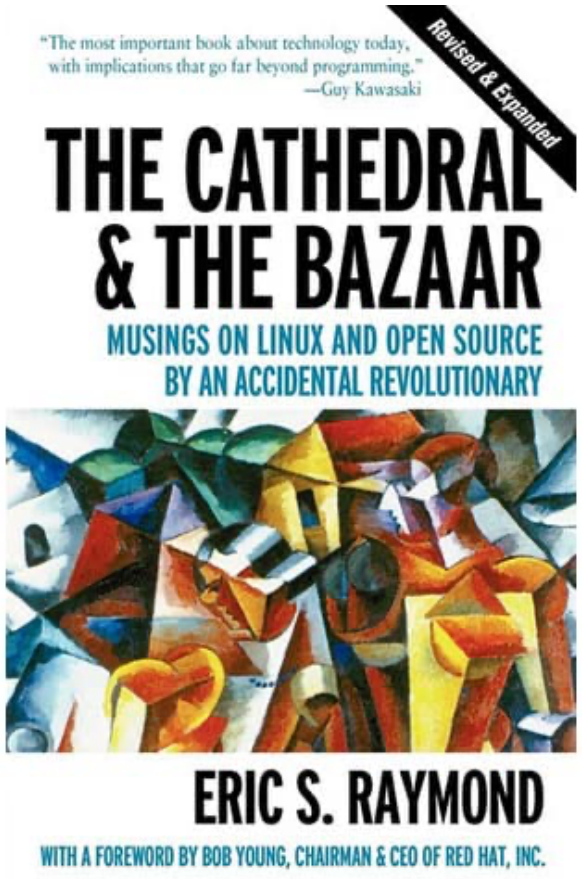
Notifications Customize

Subscribe

You're not receiving notifications from this thread.

2 participants

https://help.github.com/articles/using-pull-requests/

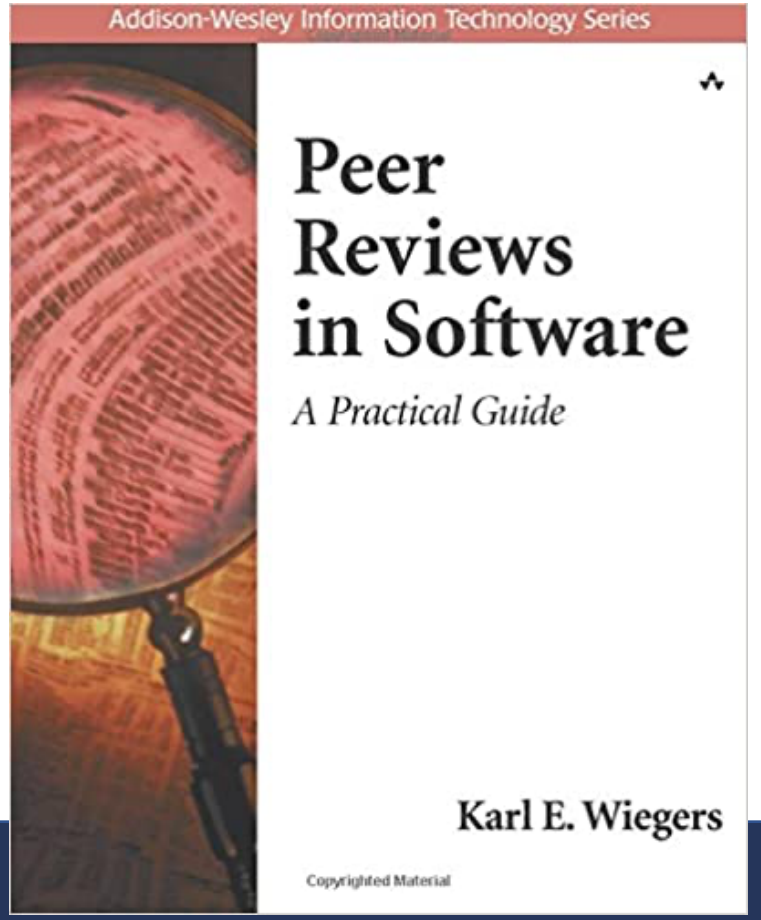


Linus's law: *"Many eyes make all bugs shallow"*

----Standard Refrain in Open Source

"Have peers, rather than customers, find defects"

--- Karl Wiegers



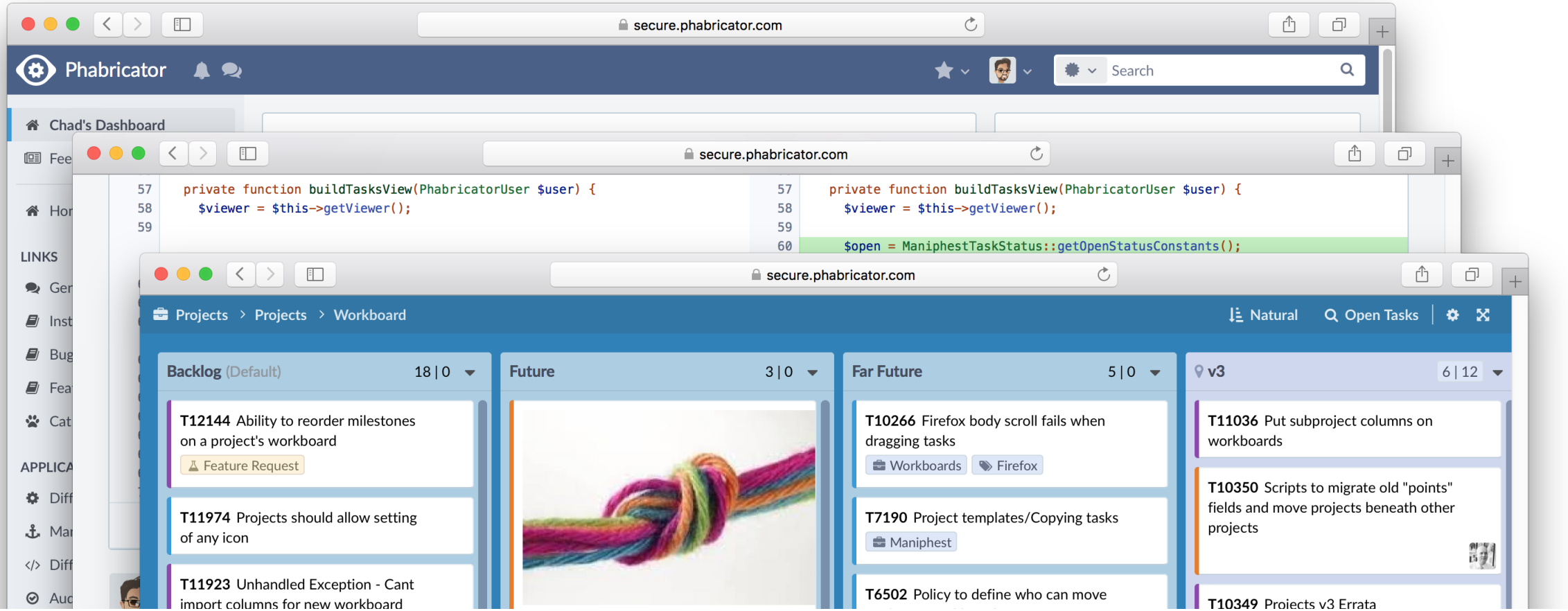
Isn’t testing sufficient?

- Only completed implementations can be tested (esp. scalability, performance)
- Design documents cannot be tested
- Tests don’t check code quality
- Many quality attributes (eg., security, compliance, scalability) are difficult to test

A second pair of eyes

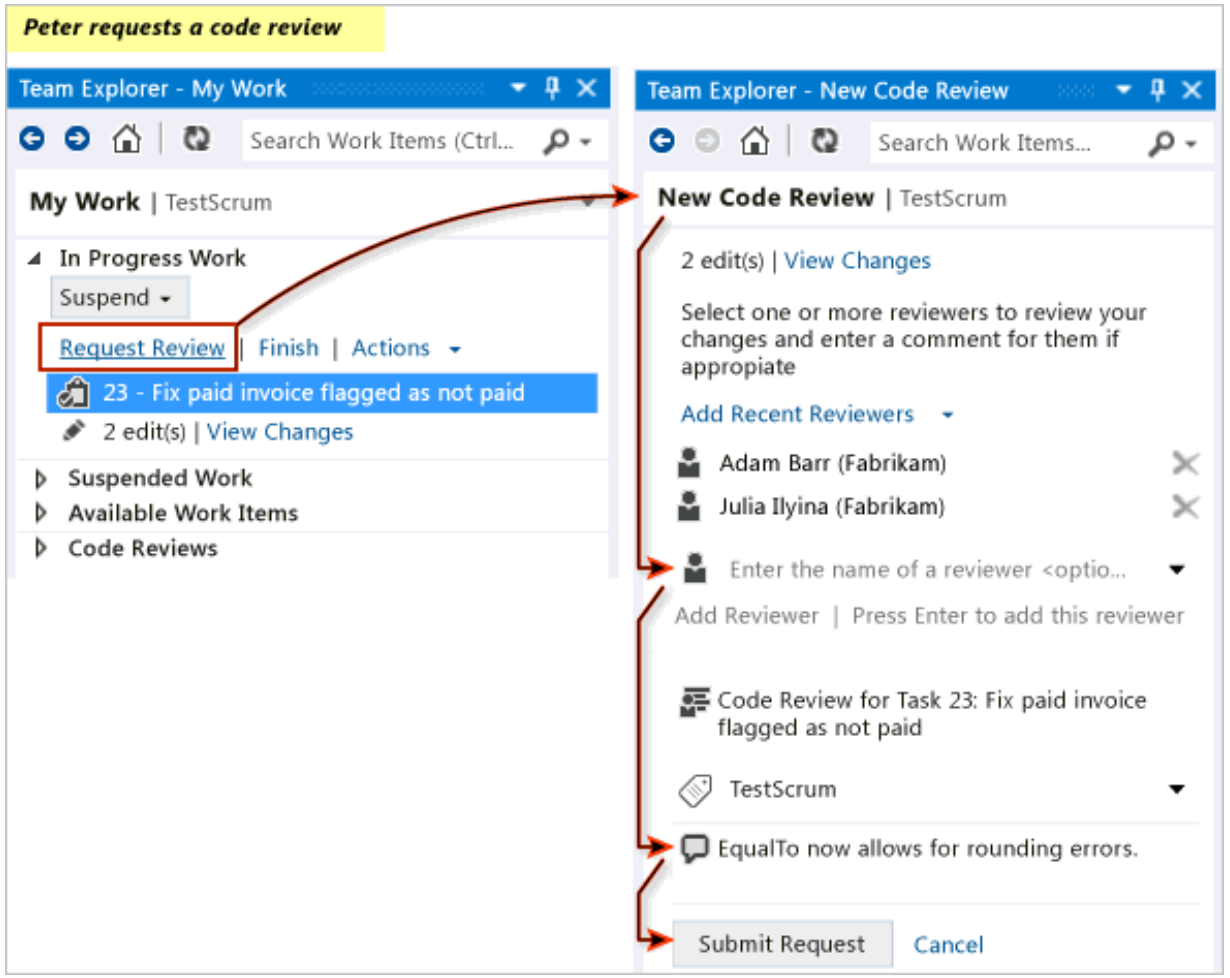
- Different background, different experience
- No preconceived idea of correctness
- Not biased by “what was intended”

facebook



https://github.com/phacility/phabricator

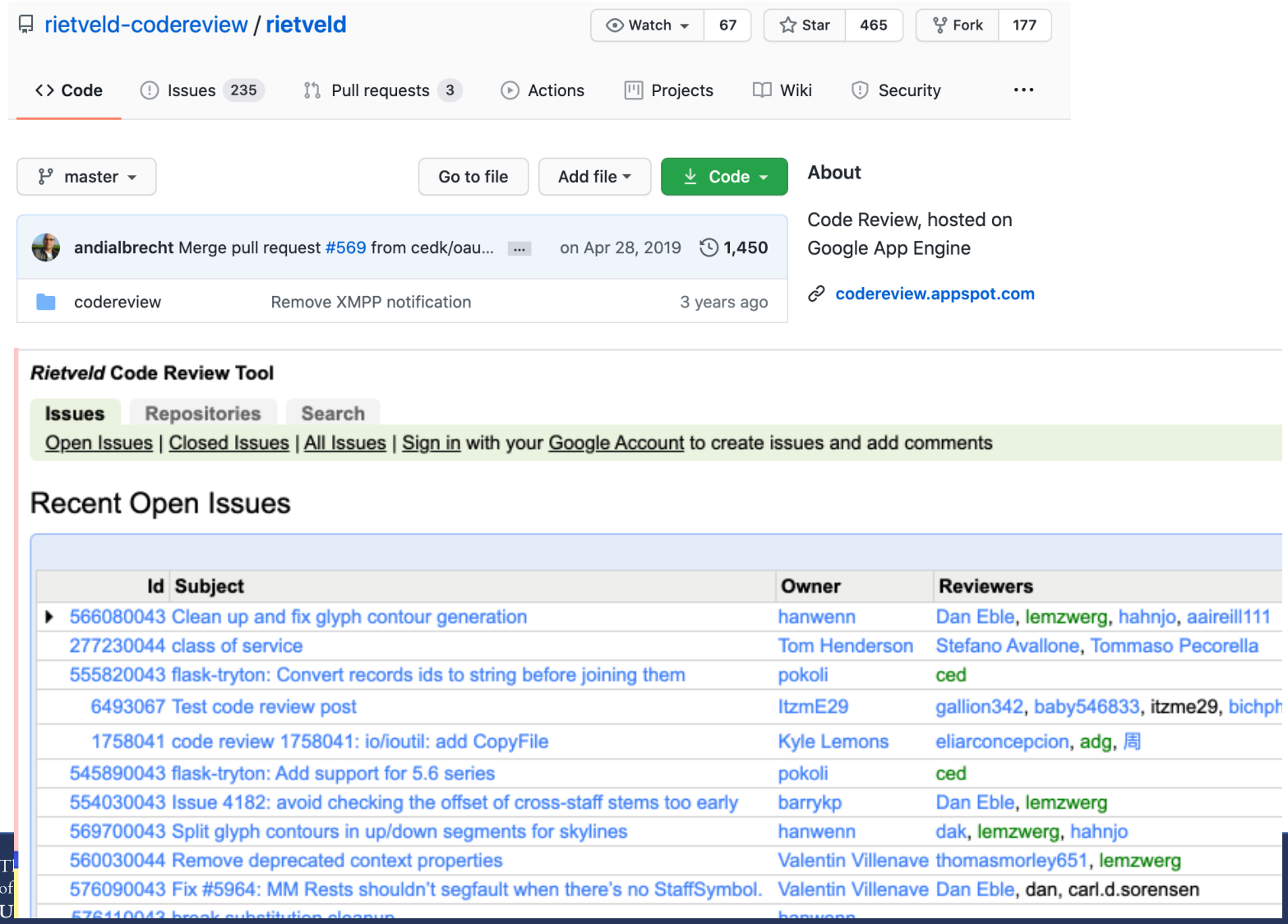
https://www.niallkennedy.com/blog/2006/11/google-mondrian.html
https://www.youtube.com/watch?v=sMql3Di4Kgc



https://docs.microsoft.com/en-us/azure/devops/repos/tfvc/day-life-alm-developer-suspend-work-fix-bug-conduct-code-review?view=azure-devops#request-a-code-review

Microsoft

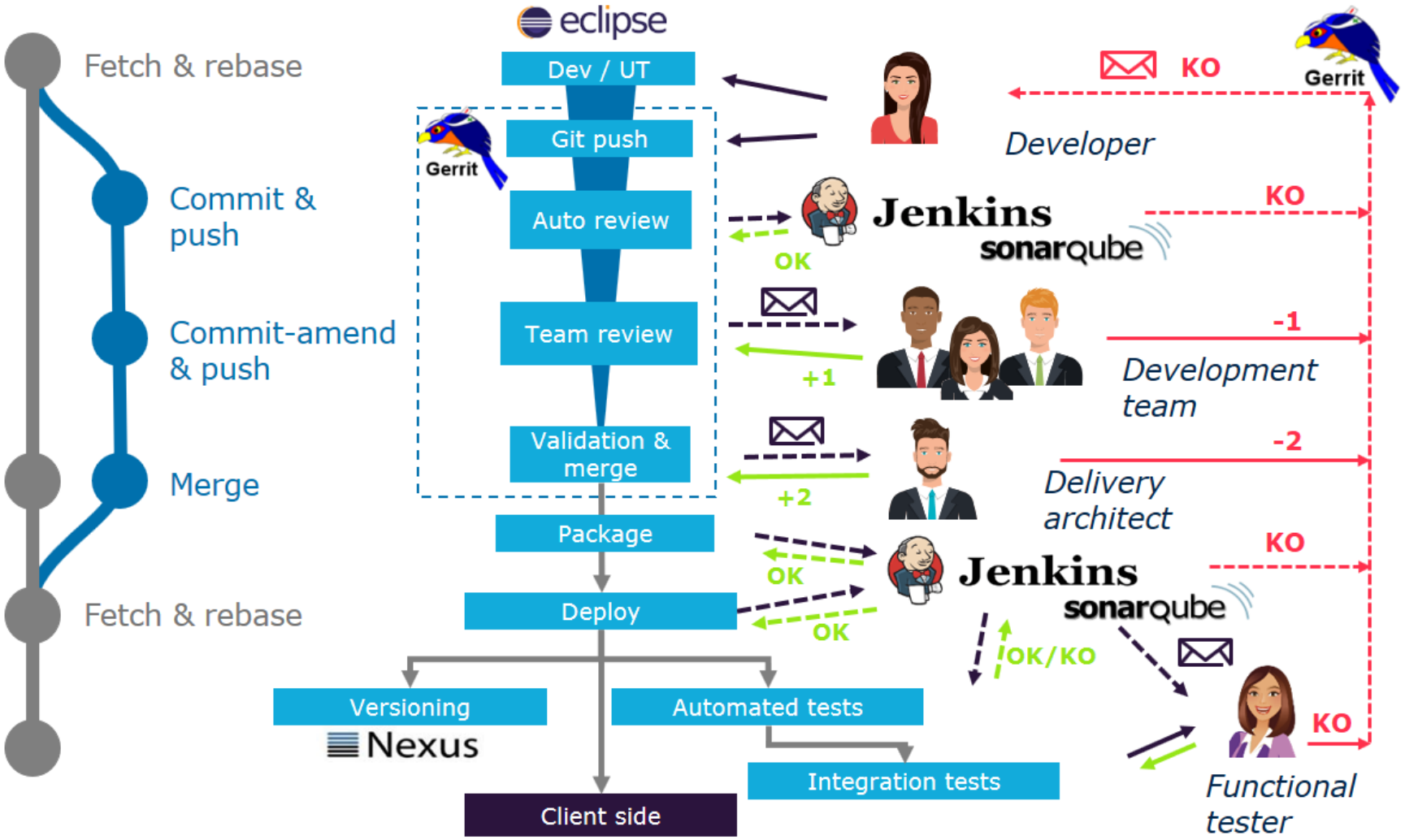
Gerrit Code Review



"As I've learned over the last two years at Google, where I developed a similar tool named Mondrian, proper code review habits can really improve the quality of a code base, and good tools for code review will improve developers' life."

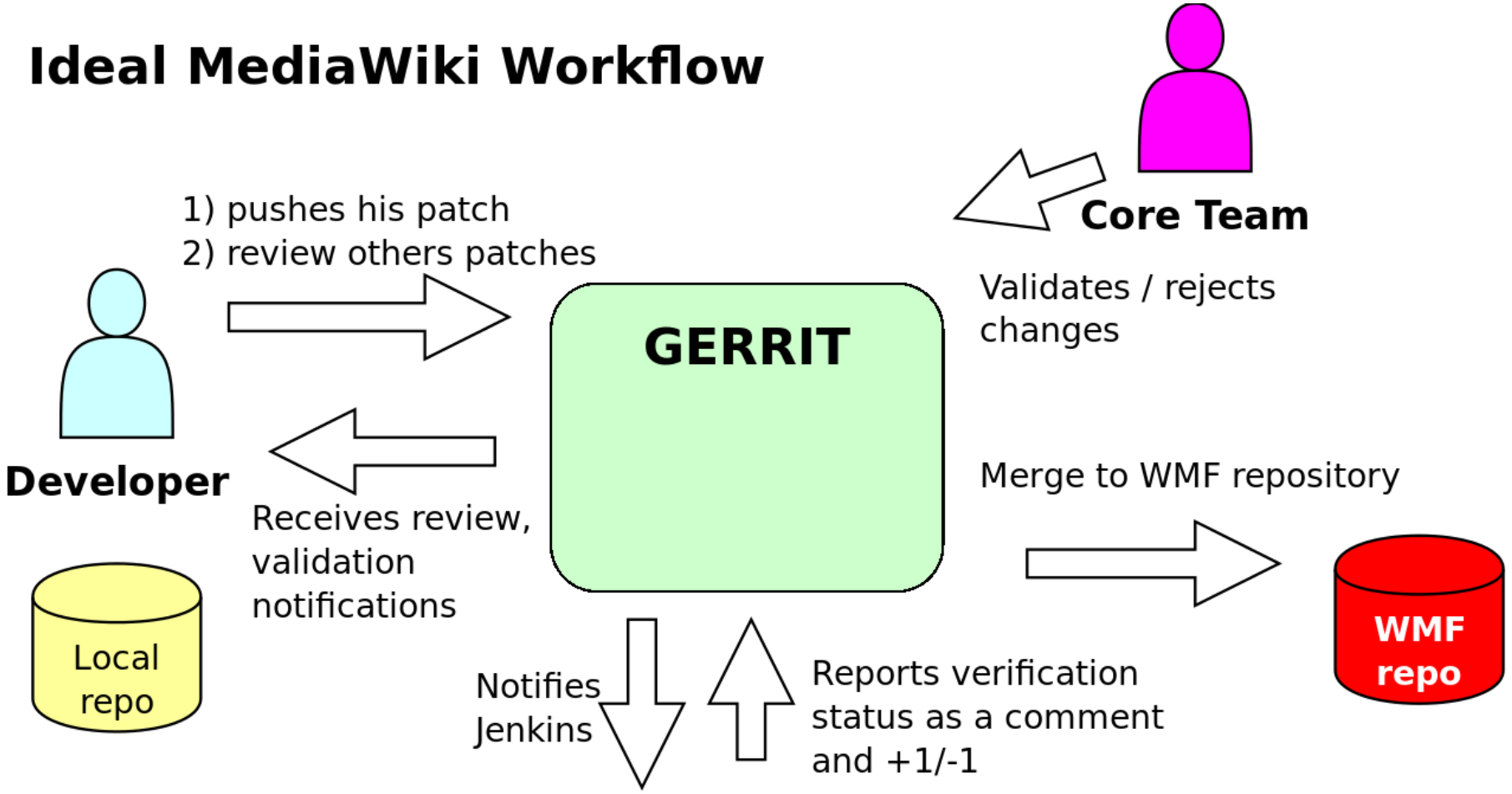
https://github.com/rietveld-codereview/rietveld

Gerrit Code Review



https://bcouetil.gitlab.io/academy/BP-gerrit.html

Ideal MediaWiki Workflow



http://www.mediawiki.org/wiki/Gerrit/Advanced_usage

Process: Checklists!

The Checklist:
https://www.newyorker.com/magazine/2007/12/10/the-checklist

OFFICIAL A.A.F. PILOT'S CHECK LIST

B-17F AND B-17G

For detailed instructions see Pilot's Handbook AN 01-20EF-1 or AN 01-20EG-1 in data case

PILOT

BEFORE STARTING

1. Pilot's Pre-flight — Complete.
2. Form IA, Form F, Weight and Balance — Checked.
3. Controls and Seats — Checked — Checked.
4. Fuel Transfer Valves and Switch — Off.
5. Intercoolers — Cold.
6. Gyros — Uncaged.
7. Fuel Shut-off Switches — Open.
8. Gear Switch — Neutral.
9. Cowl Flaps — Open Right — Open Left — Locked.
10. Turbos — Off.
11. Idle cut-off — Checked.
12. Throttles — Closed.
13. High RPM — Checked.
14. Auto Pilot — Off.
15. De-icers and Anti-icers Wing and Prop. — Off.
16. Cabin heat — Off.
17. Generators — Off.

STARTING ENGINES

1. Fire Guard and Call Clear — Left-Right.
2. Master Switches — On.
3. Battery Switches and Inverters — On and Checked.
4. Parking Brakes — Hydraulic Check-On — Checked.
5. Booster Pumps — Pressure — On and Checked.
6. Carburetor Filters — Open.
7. Fuel Quantity — Gallons per tank.
8. Start Engines
   a. Fire Extinguisher Engine Selector — Checked.
   b. Prime — As Necessary.

CO-PILOT

BEFORE TAKE OFF

1. Tail Wheel — Locked.
2. Gyro — Set.
3. Generators — On.

AFTER TAKE OFF

1. Wheels — Pilot's Signal.
2. Power Reduction.
3. Cowl Flaps.
4. Wheel Check — OK Right. OK Left.

BEFORE LANDING

1. Radio Call Altimeter — Set.
2. Crew Positions — OK.
3. Auto Pilot — Off.
4. Booster Pumps — On.
5. Mixture Controls — Auto Rich.
6. Intercooler — Set.
7. Carburetor Filters — Open.
8. Wing De-icers — Off.
9. Landing Gear
   a. Visual — Down right
      Down left
      Tail wheel Down,
      Antenna In
   b. Light — OK.
   c. Switch Off — Neutral.
10. Hydraulic Pressure — OK. Valve closed.
11. RPM 2100 — Set.
12. Turbos — Set.
13. Flaps $1/3$ — $1/3$ Down

FINAL APPROACH

14. Flaps — Pilot's Signal.
15. High RPM — Pilot's Signal.

Develop checklist for Code Review

Expectations and Outcomes of Modern Code Reviews

Bacchelli, Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013.

Reasons for Code Reviews

- Finding defects
  - both low-level and high-level issues
  - requirements/design/code issues
  - security/performance/… issues
- Code improvement
  - readability, formatting, commenting, consistency, dead code removal, naming
  - enforce to coding standards
- Identifying alternative solutions
- Knowledge transfer
  - learn about API usage, available libraries, best practices, team conventions, system design, "tricks", …
  - "developer education", especially for junior developers

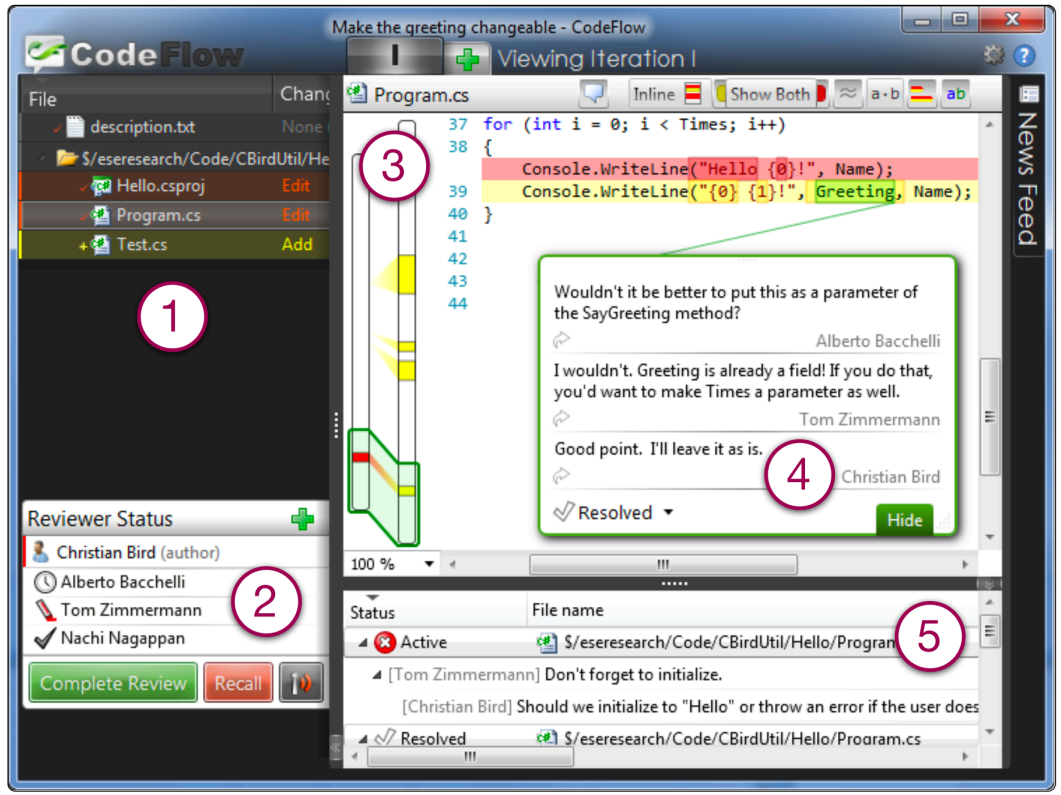
Bacchelli, Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013.

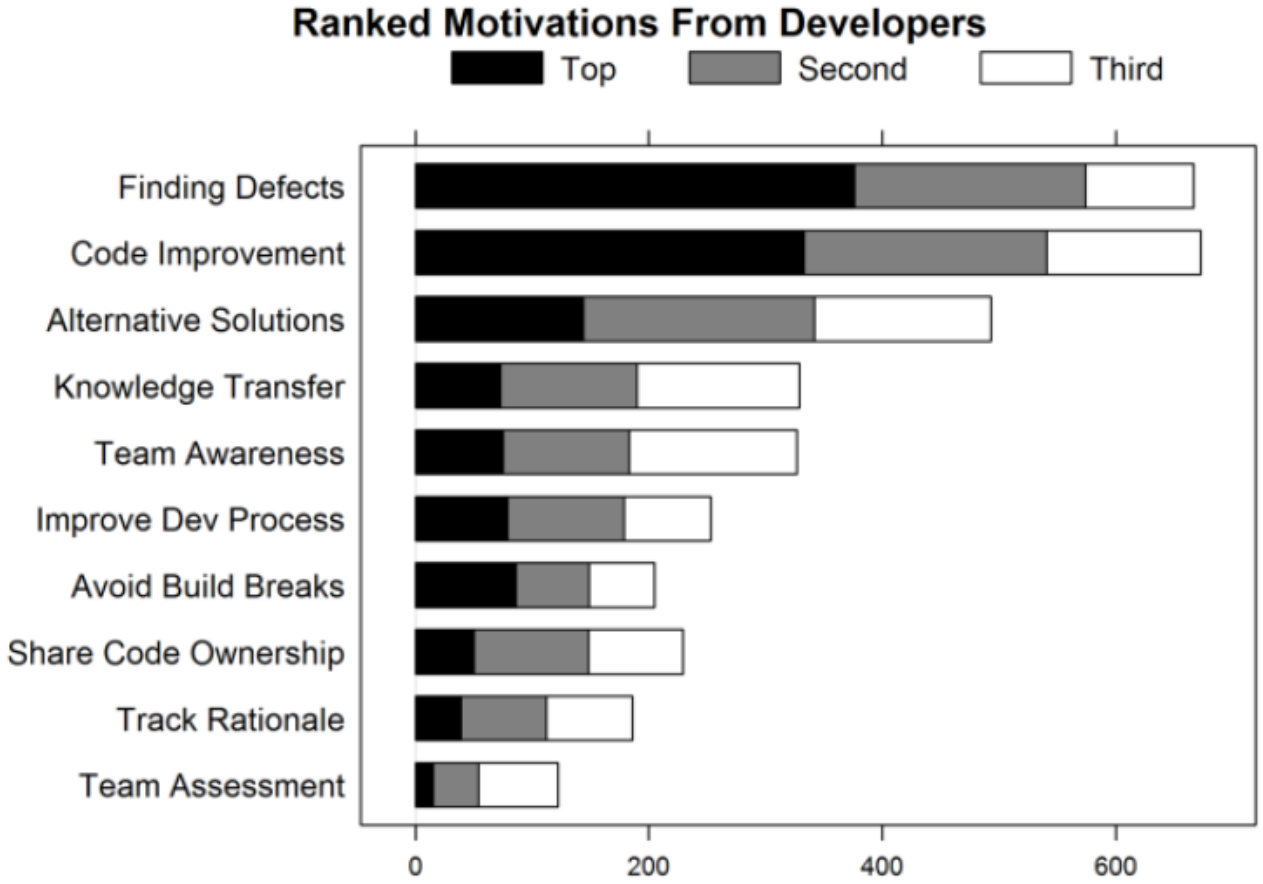
Reasons for Code Reviews (continued)

- Team awareness and transparency
  - let others "double check" changes
  - announce changes to specific developers or entire team ("FYI")
  - general awareness of ongoing changes and new functionality
- Shared code ownership
  - shared understanding of larger part of the code base
  - openness toward critique and changes
  - makes developers "less protective" of their code

Bacchelli, Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013.

Code Review at Microsoft



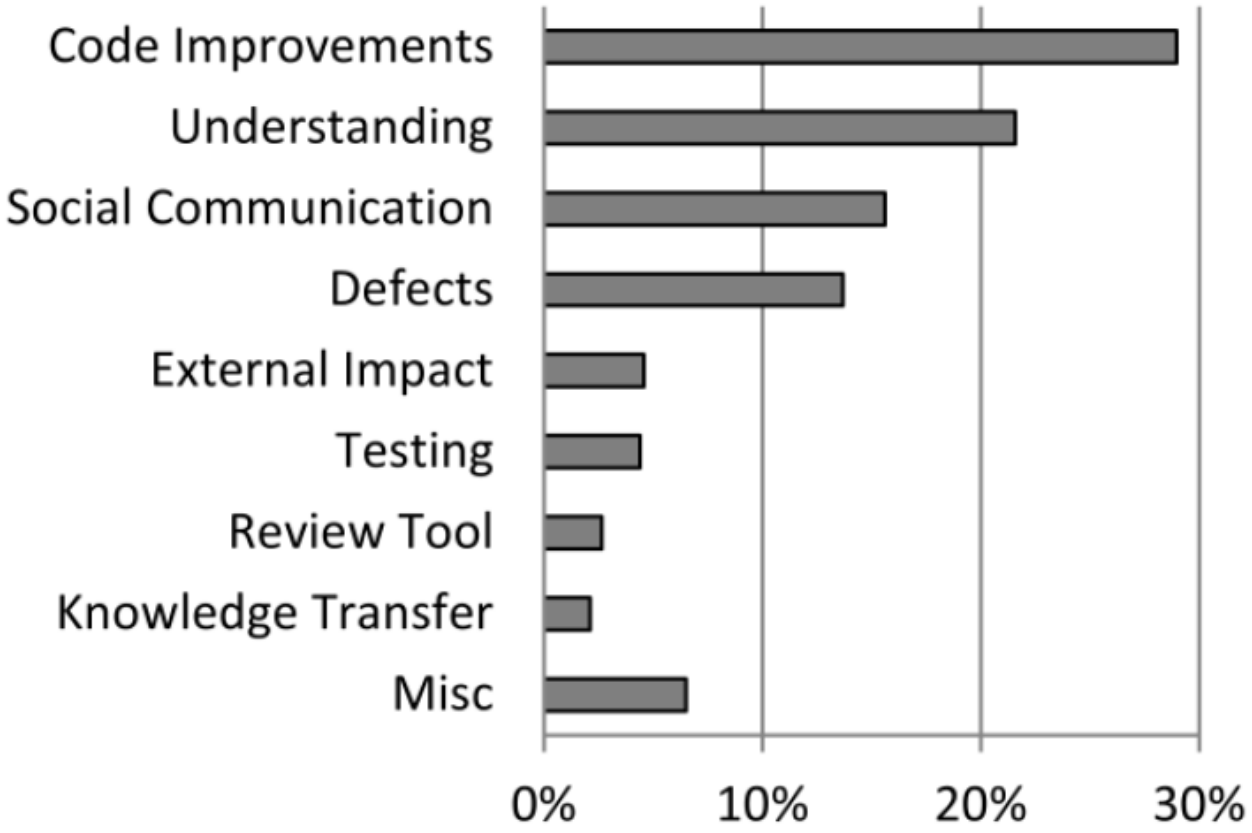


Respo Microsoft

Outcomes
(at Microsoft analyzing 200 reviews with 570 comments)

- Most frequently code improvements (29%)
  - 58 better coding practices
  - 55 removing unused/dead code
  - 52 improving readability
- Defect finding (14%)
  - 65 logical issues (“uncomplicated logical errors, eg., corner cases, common configuration values, operator precedence)
  - 6 high-level issues
  - 5 security issues
  - 3 wrong exception handling
- Knowledge transfer
  - 12 pointers to internal/external documentation etc

Outcomes (Analyzing Reviews)



Mismatch of Expectations and Outcomes

- Low quality of code reviews
  - Reviewers look for easy errors, as formatting issues
  - Miss serious errors
- Understanding is the main challenge
  - Understanding the reason for a change
  - Understanding the code and its context
  - Feedback channels to ask questions often needed
- No quality assurance on the outcome

Code Review at Google

- Introduced to "*force developers to write code that other developers could understand*"
- 3 Found benefits:
  - checking the consistency of style and design
  - ensuring adequate tests
  - improving security by making sure no single developer can commit arbitrary code without oversight

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko and Alberto Bacchelli. 2018. Modern Code Review: A Case Study at Google. International Conference on Software Engineering

google / eng-practices

Watch 621 | Star 15.3k | Fork 1.4k

Code | Issues 3 | Pull requests 3 | Actions | Wiki | Security | Insights

master | 2 branches | 0 tags | Go to file | Add file | Code

ninabikes and Copybara-Service Add missing comma. ... eaff1ef on Aug 4 54 commits

review	Add missing comma.	2 months ago
LICENSE	Internal change	8 months ago
README.md	Internal change	2 months ago
_config.yml	Internal change	13 months ago

README.md

Google Engineering Practices Documentation

Google has many generalized engineering practices that cover all languages and all projects. These documents represent our collective experience of various best practices that we have developed over time. It is possible that open source projects or other organizations would benefit from this knowledge, so we work to make it available publicly when possible.

Currently this contains the following documents:

- Google's Code Review Guidelines, which are actually two separate sets of documents:
  - The Code Reviewer's Guide
  - The Change Author's Guide

About

Google's Engineering Practices documentation

google.github.io/eng-practices/

Readme

View license

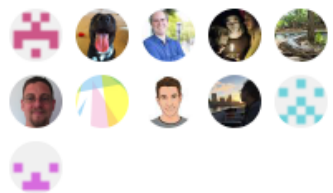
Releases

No releases published

Packages

No packages published

Contributors 11

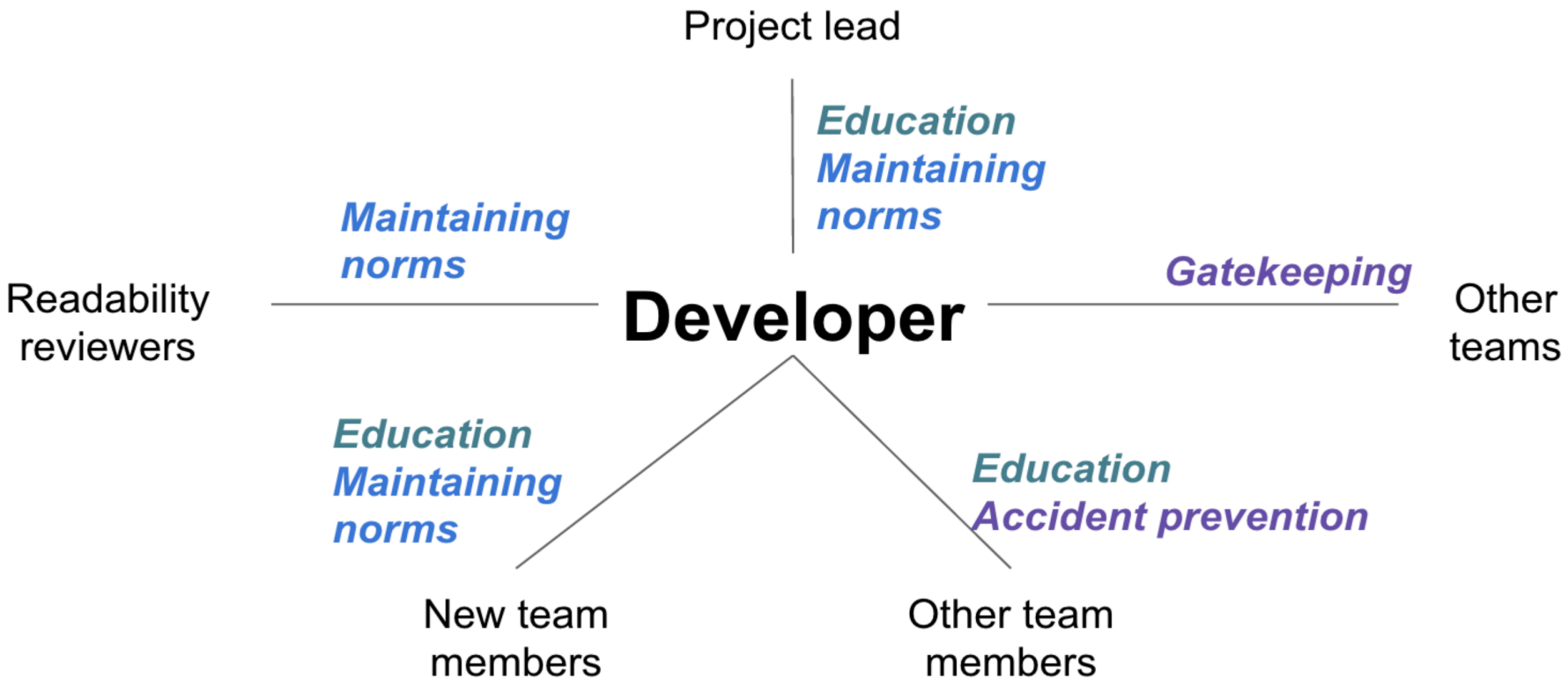


Google's Code Review Policy

- All change lists must be reviewed. Period.
- Any CL can be reviewed by any engineer at Google.
- Each directory has a list of owners. At least one reviewer or the author must be an owner for each file that was touched in the commit. If the author is not in the owners file, the reviewer is expected to pay extra attention to how the code fits in to the overall codebase.
- [… readability review …] If the author does not have readability review, the reviewer is expected to pay extra attention to coding style (both the syntax and the proper use of libraries in that language).
- One can enforce that any CLs to that directory are CC'd to a team mailing list.
- Reviews are conducted either by email, or using a web interface called Mondrian
- In general, the review must have a positive outcome before the change can be submitted (enforced by perforce hooks). However, if the author of the changelist meets the readability and owners checks, they can submit the change TBR, and have a post-hoc review. There is a process which will harass reviewers with very annoying emails if they do not promptly review the change.

source: https://www.quora.com/What-is-Googles-internal-code-review-policy-process, 2010

Reviewing Relationships



How to write code review comments

- Be kind.
- Explain your reasoning.
- Balance giving explicit directions with just pointing out problems and letting the developer decide.
- Encourage developers to simplify code or add code comments instead of just explaining the complexity to you.

*Bad: "Why did **you** use threads here when there's obviously no benefit to be gained from concurrency?"*
Good: "The concurrency model here is adding complexity to the system without any actual performance benefit that I can see. Because there's no performance benefit, it's best for this code to be single-threaded instead of using multiple threads."

https://google.github.io/eng-practices/review/reviewer/comments.html

Summary

- Code reviews effective to identify bugs
- Additional benefits (e.g., knowledge transfer, shared code ownership, awareness)
- Reviews require understanding
- Different review types with different formality
- Formal inspection require planning & social skills, are expensive, but very effective

Further Reading

- Sommerville. Software Engineering. 8th Edition. Addison-Wesley 2007. Chapter 22.2
  - Overview of formal inspections
- Wiegers. Peer Reviews in Software. Addison-Wesley 2002
  - Entire book on formal inspections; how to run them and how to introduce them
- Bacchelli and Bird. "Expectations, outcomes, and challenges of modern code review.“ *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 2013.
  - Detailed studies of modern code reviews at Microsoft
- Oram and Wilson (ed.). Making Software. O’Reilly 2010. Chapter 18
  - Overview of empirical research on formal inspections

styleguide

Google Style Guides

Every major open-source project has its own style guide: a set of conventions (sometimes arbitrary) about how to write code for that project. It is much easier to understand a large codebase when all the code in it is in a consistent style.

"Style" covers a lot of ground, from "use camelCase for variable names" to "never use global variables" to "never use exceptions." This project (google/styleguide) links to the style guidelines we use for Google code. If you are modifying a project that originated at Google, you may be pointed to this page to see the style guides that apply to that project.

This project holds the C++ Style Guide, C# Style Guide, Swift Style Guide, Objective-C Style Guide, Java Style Guide, Python Style Guide, R Style Guide, Shell Style Guide, HTML/CSS Style Guide, JavaScript Style Guide, AngularJS Style Guide, Common Lisp Style Guide, and Vimscript Style Guide. This project also contains cpplint, a tool to assist with style guide compliance, and google-c-style.el, an Emacs settings file for Google style.

https://google.github.io/styleguide/

Further Reading 2

Awesome Code Review awesome

> A curated list of tools, articles, books, and any other resource related to code review

Code review is the systematic examination (sometimes referred to as peer review) of computer source code.

Contents

- Academic Papers
- Articles
- Books
- Talks and Podcasts
- Tools

- https://github.com/joho/awesome-code-review