Software Reviews

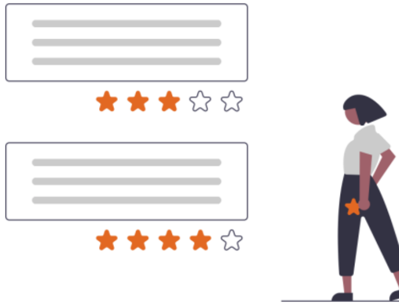
Software Engineering II
WS 2020/21

Enterprise Platform and Integration Concepts

Image by *Chris Isherwood* from flickr: https://www.flickr.com/photos/isherwoodchris/6807654905/ (CC BY-SA 2.0)

Review Meetings

> “a **software** product is [**examined** by] project personnel, managers, users, customers, user representatives, or other interested parties **for comment or approval**”
>
> —IEEE1028



Principles

- Generate comments on software
- Several sets of eyes check
- Emphasis on **people over tools**

Software Reviews

Motivation

- Improve code
- Discuss alternative solutions
- Transfer knowledge
- Find defects

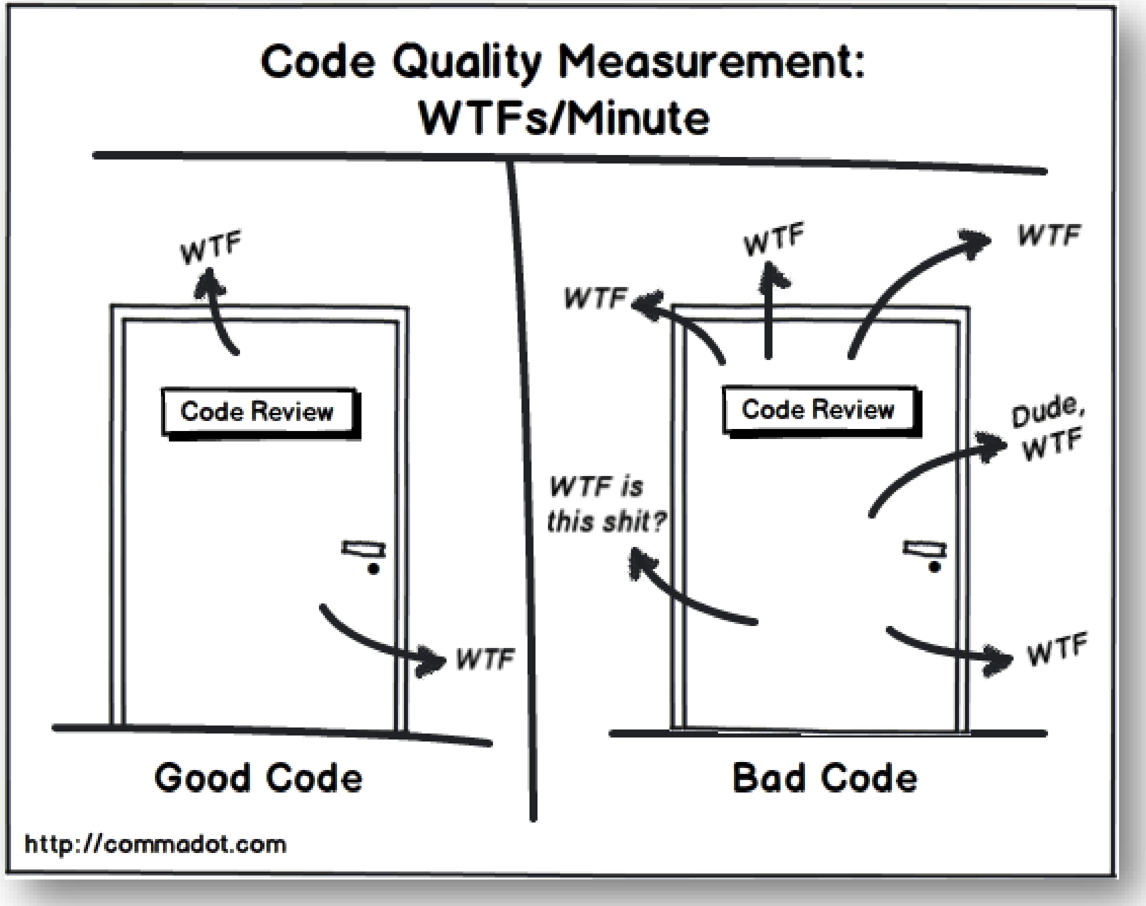
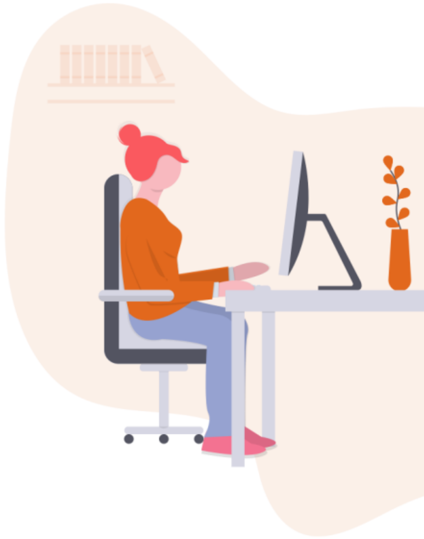


Image by Glen Lipka: http://commadot.com/wtf-per-minute/

Involved Roles

Manager

- Assessment is an important task for manager
- Possible lack of deep technical understanding
- Assessment of product vs. assessment of person
- Outsider in review process
- **Support with resources** (time, staff, rooms, …)



Developer

- Should not justify but only explain their results
- **No boss** should take part at review

Review Team

Team lead

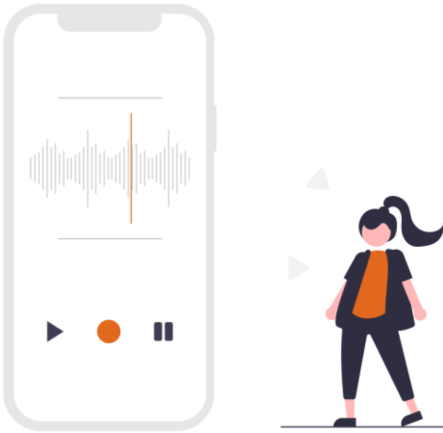
- Responsible for quality of review & moderation
- Technical, personal and administrative competence

Reviewer

- Study the material before first meeting
- Don’t try to achieve personal targets!
- Give positive *and* negative comments on review artifacts

Recorder

- Any reviewer, can rotate even in review meeting
- Protocol as basis for final review document



Tasks of Review Team

Deliver a good review

- "Don't shoot the messenger"
- Identify issues, but **don't try to solve them**

Clear assessments of artifacts

- Accepted, partly accepted, needs corrections, rejected
- Acceptance only possible if no participant speaks against it

➡ Artifact issues should be **identified and documented**

Types of Reviews [IEEE1028-97]

Management Review

- Monitor progress and status of plans, confirm requirements
- **Evaluate effectiveness** of management approaches / corrective actions

Technical Review

- Evaluate entire software on suitability for intended use
- Provide evidence whether software product **meets specifications**

Types of Reviews [IEEE1028-97]

Inspections

- Identify software product anomalies, invented at IBM in the 1970's
- **Formal process**, can involve hard copies of the code and documents
- Review team members check important artifacts independently, consolidation meeting with developers
- Preparation time for team members, shorter meetings

Walk-through

- Evaluate software, focus on **educating an audience**
- Organized by developer for reviewing own work
- Bigger audience can participate, little preparation for team members

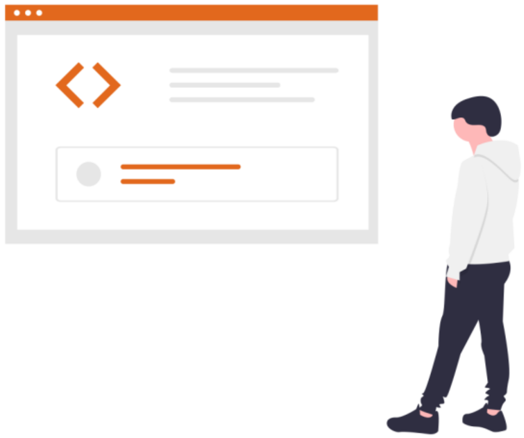
Artifacts to Review

Should be reviewed	Might not have to be reviewed
Parts with complicated algorithms	Trivial parts where no complications are expected
Critical parts where faults lead to system failure	Parts which won’t break the functionality if faults occur
Parts using new technologies / environment / …	Parts which are similar to those previously reviewed
Parts constructed by inexperienced team members	Reused or redundant parts

Modern Code Reviews

[Rigby’13]
[Bacchelli‘13]

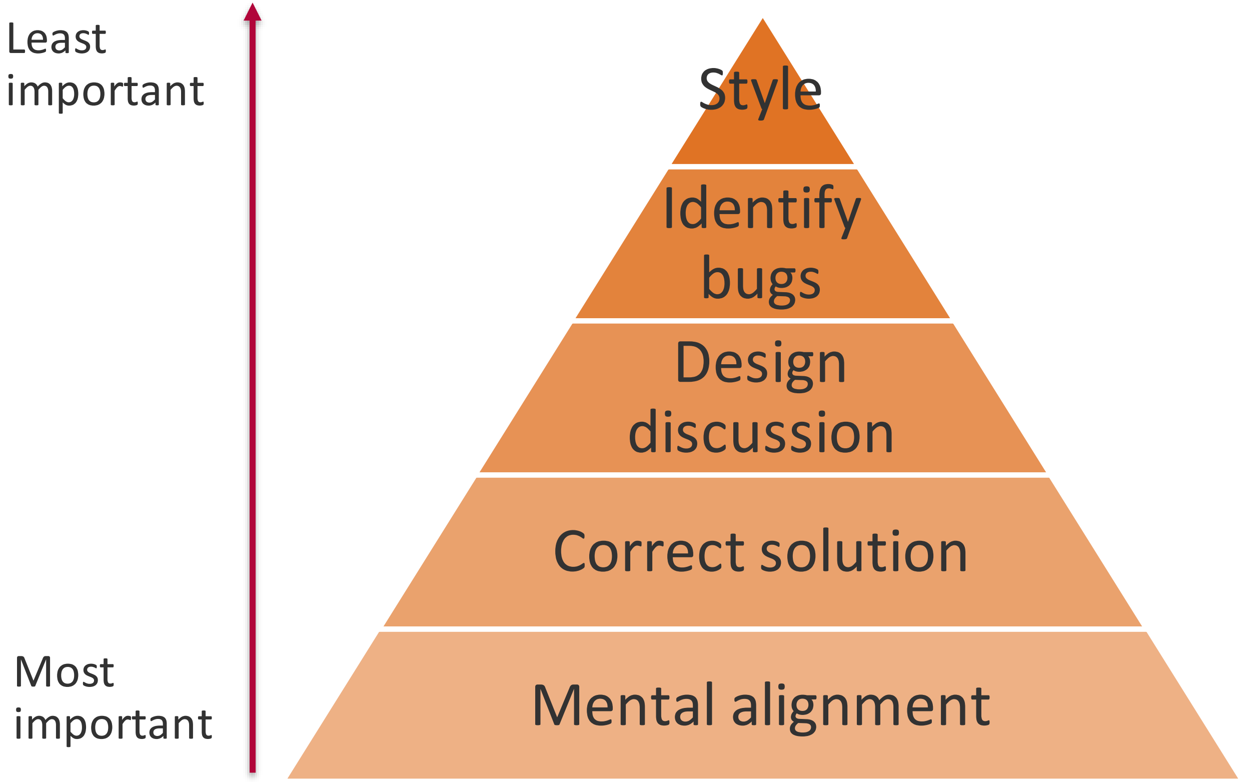
- Follows more **lightweight**, **flexible** process
- Change sizes are **smaller**
- Performed **regularly** and **quickly**, mainly just before code committed to main branch



Shift in Focus

- From defect finding to group problem solving activity
- Prefer discussion and fixing code over reporting defects

Code Review Goals



Hierarchy of goals

- Building a shared mental model
- Ensuring sane design
- Findings bugs vs. understanding code

http://blakesmith.me/2015/02/09/code-review-essentials-for-software-teams.html

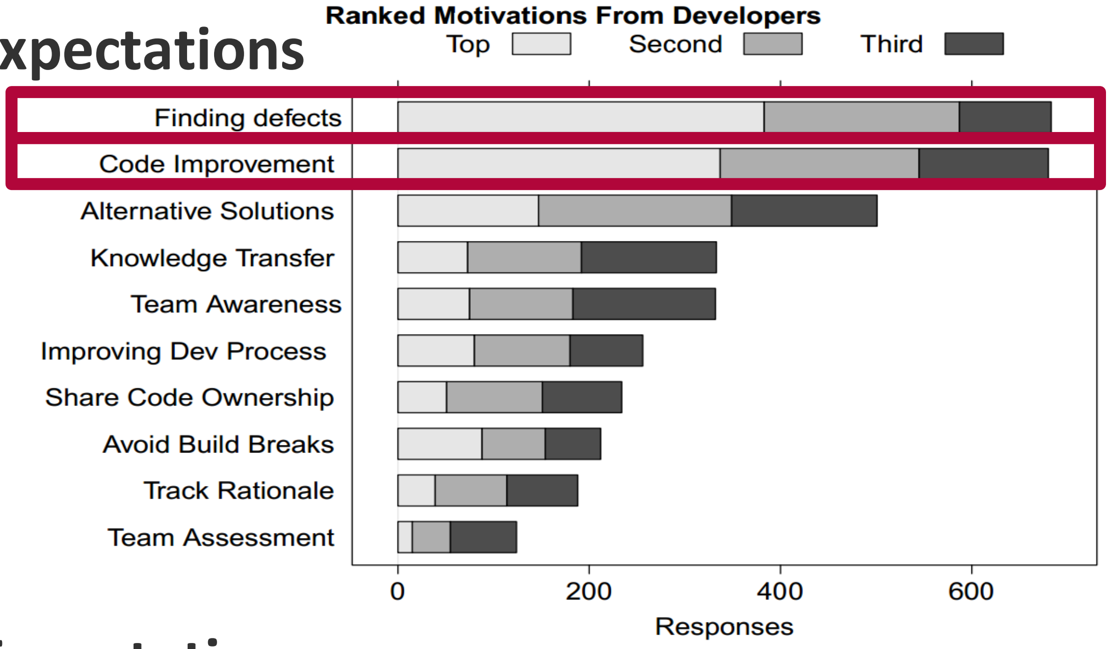
Recent Research

[Bosu’17]
[McIntosh’14]
[Bacchelli ‘13]

- Code review coverage and review participation share **significant link with software quality**
- Most comments concern code improvements, understandability, social communication
- Only ~15% of comments indicate possible defects
- Developers spend approximately five hours per week (10-15% of their time) in code reviews

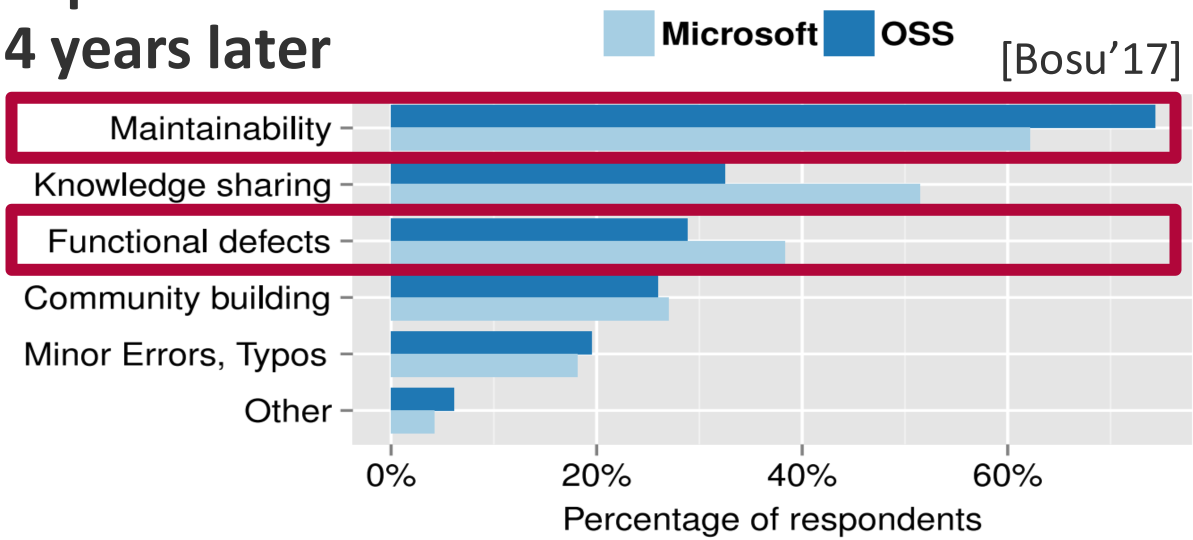
Recent Research

Expectations



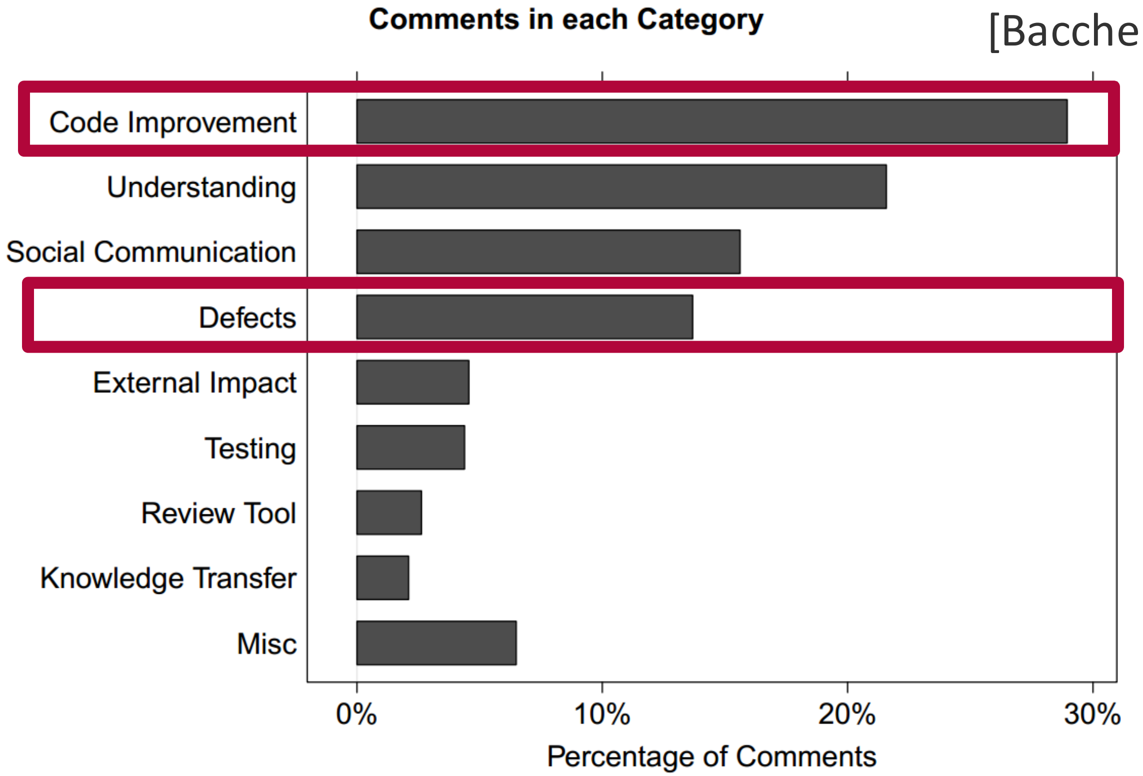
Expectations 4 years later

[Bosu'17]



Empirical study outcomes

[Bacchelli '13]



Maintainability and code improvements identified as most important aspects of modern code reviews

Challenges of the Review Process

- **Delay** the use of implemented features
- Forces the reviewers to **switch context** away from their current work
- Offer little feedback for **legacy code**

- **Overloading** (too many files), developers create large patches

- **Overcrowding** (too many reviewers), assigning too many reviewers may lower review quality

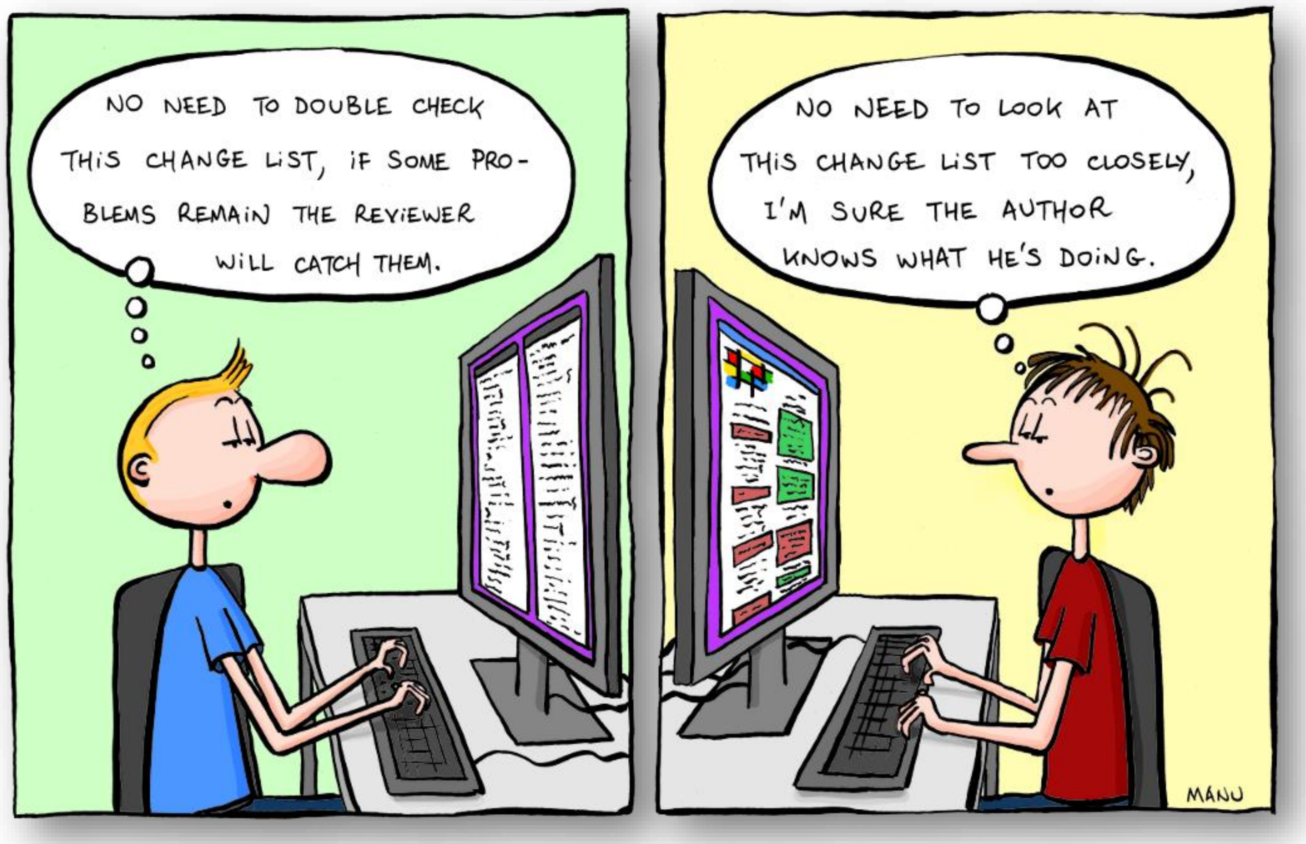

NO NEED TO DOUBLE CHECK THIS CHANGE LIST, IF SOME PROBLEMS REMAIN THE REVIEWER WILL CATCH THEM.
NO NEED TO LOOK AT THIS CHANGE LIST TOO CLOSELY, I'M SURE THE AUTHOR KNOWS WHAT HE'S DOING.
MANU

Image: https://devops.com/dark-side-infrastructure-code/

Post-commit Code Review

- Review **after committing to VCS** (pull requests are one! way of doing this)
- Used by most projects on GitHub and BitBucket

+

- Developers can commit continuously
- Other team members see code changes in VCS and can adapt their work
- Flexible definition of the code to be reviewed (set of commits, whole branch, some files)

−

- Chance of unreviewed code in main repository
  - □ Need to / can set restrictions
- Requires branches or similar to work effectively
- May take a while for developers to come back to the code and improvement ideas

https://www.devart.com/review-assistant/learnmore/pre-commit-vs-post-commit.html

Pre-commit Code Review

- Review **before committing** to version control system (e.g. using mailing lists / Gerrit, Crucible tools)
- Used by e.g. Linux Kernel, Git, Google

+

- No code enters unreviewed
- Code quality standards met before commit, no 'fixes'
- No repository access needed for reviews
- Other developers definitely not affected by bugs in reviewed code

–

- Reviewing all code takes time
  - Deciding what needs a review takes time too
- Possibly another complex system to handle
  - Might not want to work on submitted code until review done (e.g. mailing list)

Reviewer Assignment

Usually, **two reviewers** find optimal number of defects

Reviewer candidates

- People who contributed changes (find defects)
- New developers (transfer knowledge)
- Team members with a small review queue
- Reviewers with different fields of expertise
- Let reviewers know what they should look out for

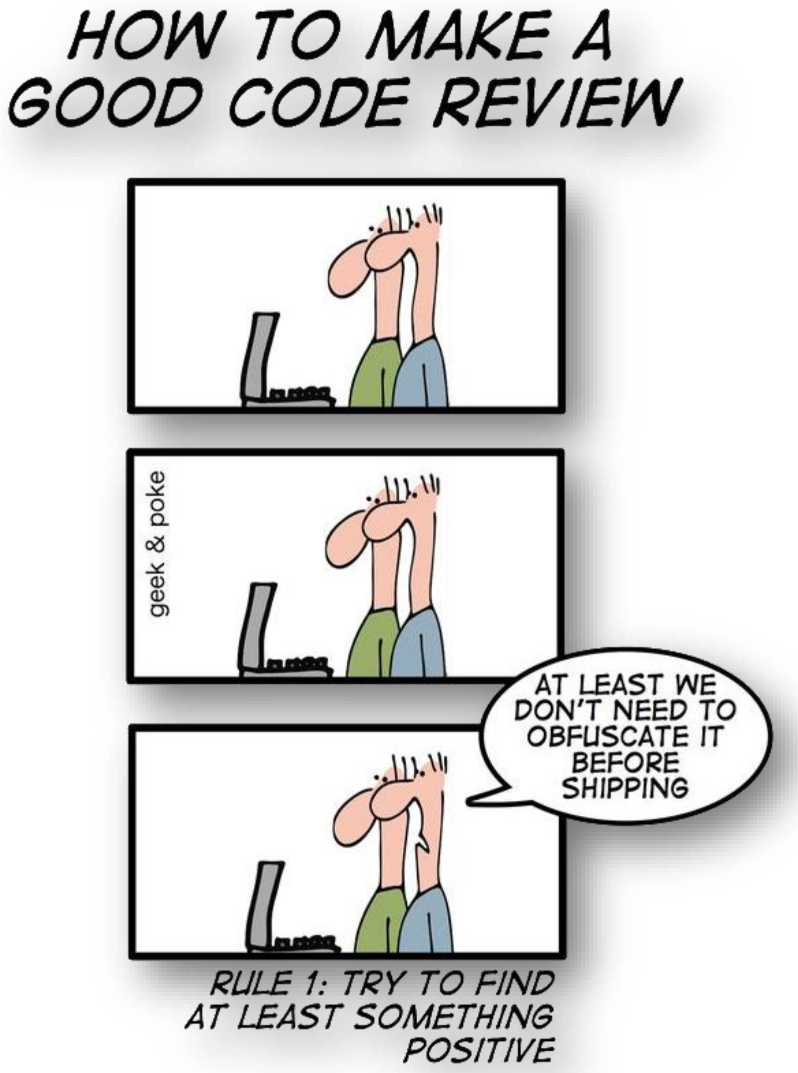
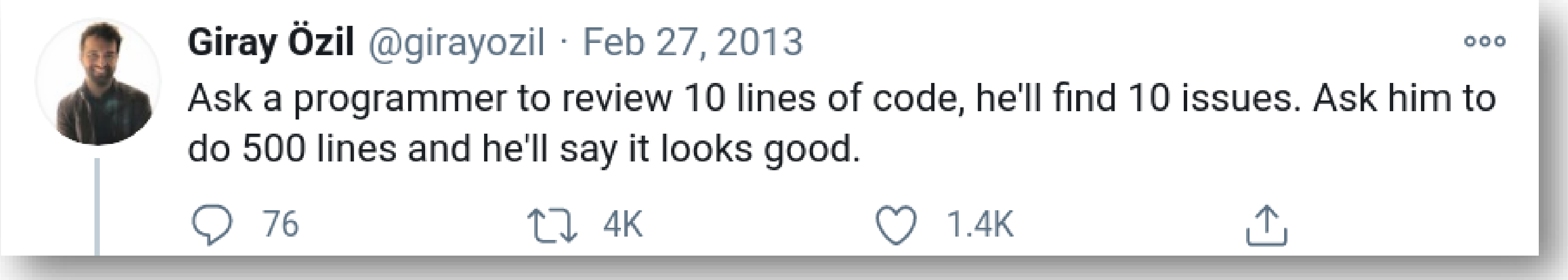

HOW TO MAKE A GOOD CODE REVIEW
geek & poke
AT LEAST WE DON'T NEED TO OBFUSCATE IT BEFORE SHIPPING
RULE 1: TRY TO FIND AT LEAST SOMETHING POSITIVE

Image: http://geek-and-poke.com/geekandpoke/2010/11/1/how-to-make-a-good-code-review.html

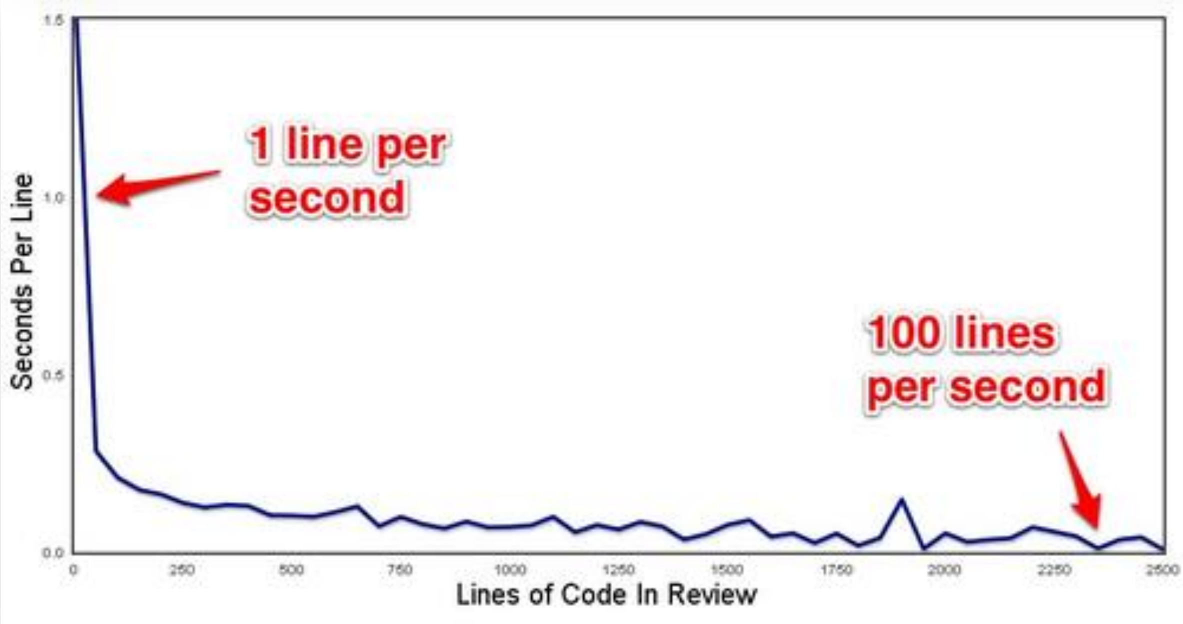
[Rigby'13]

Review Content



Giray Özil @girayozil · Feb 27, 2013

Ask a programmer to review 10 lines of code, he'll find 10 issues. Ask him to do 500 lines and he'll say it looks good.



- Size of artifact to review matters
- Semantically coherent changes easier to review than interleaved concerns

Images: http://atlassianblog.wpengine.com/developer/assets_c/2011/07/mt-perloc-thumb-500x263-7290.png
https://twitter.com/girayozil/status/306836785739210752?lang=en

Code Review In Industry

[Rigby'13]

Microsoft

- Median completion times: 14.7h (Bing), 18.9h (Office), 19.8h (SQL Server)
- Median number of reviewers: 3-4
- Developers spend 4-6 hours per week on reviews

Google

- Mandatory review of every change
- Median completion times: 15.7h (Chrome), 20.8h (Android)
- Median patch size: 78 lines (Chrome), 44 lines (Android)
- Median number of reviewers: 2

Code Review Tools

Gerrit (https://www.gerritcodereview.com/)

- Integrated with Github: http://gerrithub.io
- Used by, e.g., Chromium, Eclipse, Qt, Typo3, Wikimedia, etc.
- Plug-ins available (e.g. EGerrit for Eclipse)

FishEye (https://www.atlassian.com/software/fisheye/overview)

- Visualize, Review, and organize code changes

GitHub Pull Requests

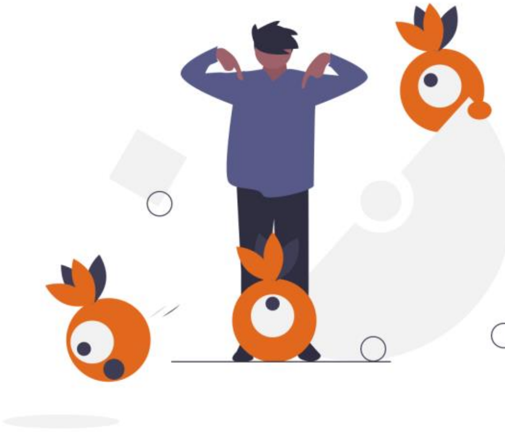
- Branches with comments and checks

Software Review Helpers

- Testing checks functionality via dynamic analysis
- Code reviews manually check code **quality** via static analysis

Automated static analysis (linters)

- Code coverage (e.g. SimpleCov https://github.com/simplecov-ruby/simplecov)
- Coding conventions (e.g. RuboCop, https://github.com/rubocop-hq/rubocop)
- Code smells (e.g. reek https://github.com/troessner/reek)



Summary

- Reviews are not a new thing, good reasons to do them
- Different types of review techniques
  - Management Review
  - Technical Review
  - Inspection
  - Walk-through
  - Modern / contemporary code reviews
- Method to find faults and improvement opportunities early in the process

References

[Bosu’17] Bosu, Amiangshu, et al. "Process Aspects and Social Dynamics of Contemporary Code Review: Insights from Open Source Development and Industrial Practice at Microsoft." *TSE* 43.1 (2017): 56-75.

[McIntosh’14] McIntosh, Shane, et al. "The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects." *MSR’14.*

[Rigby’13] Rigby, Peter C., and Christian Bird. "Convergent contemporary software peer review practices." *FSE’13*.

[Bacchelli‘13] Bacchelli, Alberto, and Christian Bird. "Expectations, outcomes, and challenges of modern code review." *ICSE’*13.

[Feitelson‘13] Feitelson, Dror G., Eitan Frachtenberg, and Kent L. Beck. "Development and deployment at facebook." *IEEE Internet Computing* 17.4 (2013): 8-17.