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

Code Reviews — Software Engineering II

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 [IEEE 1028-97]

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

<table>
<thead>
<tr>
<th>Should be reviewed</th>
<th>Might not have to be reviewed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parts with complicated algorithms</td>
<td>Trivial parts where no complications are expected</td>
</tr>
<tr>
<td>Critical parts where faults lead to system failure</td>
<td>Parts which won’t break the functionality if faults occur</td>
</tr>
<tr>
<td>Parts using new technologies / environment / ...</td>
<td>Parts which are similar to those previously reviewed</td>
</tr>
<tr>
<td>Parts constructed by inexperienced team members</td>
<td>Reused or redundant parts</td>
</tr>
</tbody>
</table>
Modern Code Reviews

- 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

[Rigby’13]
[Bacchelli’13]
Code Review Goals

Hierarchy of goals
- Building a shared mental model
- Ensuring sane design
- Findings bugs vs. understanding code
Recent Research

- 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

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

Expectations

Ranked Motivations From Developers

Finding defects
Code Improvement
Alternative Solutions
Knowledge Transfer
Team Awareness
Improving Dev Process
Share Code Ownership
Avoid Build Breaks
Track Rationale
Team Assessment

Responses

Empirical study outcomes

Comments in each Category

Code Improvement
Understanding
Social Communication
Defects
External Impact
Testing
Review Tool
Knowledge Transfer
Misc

Percentage of Comments

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

Bacchelli ‘13

Bosu’17
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
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

Size of artifact to review matters
- Semantically coherent changes easier to review than interleaved concerns
Code Review In Industry

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

[Rigby’13]
Code Review Tools

**Gerrit** ([https://www.gerritcodereview.com/](https://www.gerritcodereview.com/))
- Integrated with Github: [http://gerrithub.io](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](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)
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


