Software Reviews

Scalable Software Engineering
WS 2021/22

Enterprise Platform and Integration Concepts

Image courtesy @EthicsInBricks in Twitter: https://twitter.com/EthicsInBricks/status/1430556314556669956 (with permission)
Software Reviews

“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
- Lower cost of fixing defects in review than in the field
Motivations

- **Improve code quality** (e.g. maintainability, readability, uniformity)
- Discuss alternative solutions, **generate ideas** for the future
- **Knowledge transfer** regarding codebase
- Increase sense of **Collective Code Ownership**
- Find **defects**
- Check **compliance** (e.g. legal)

Image by Glen Lipka: http://commadot.com/wtf-per-minute/
Inspections
- Identify software product anomalies
- Since the 1970’s, aka “Fagan Inspection”
- Formal process, can involve hard copies of the code and documents
- Review team checks artifacts independently before, consolidation meeting with developers

### Focus in Reviews

<table>
<thead>
<tr>
<th>Reviewed first</th>
<th>Reviewed later</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementations of complex algorithms</td>
<td>Well-understood problem domains</td>
</tr>
<tr>
<td>Code where faults or exceptions lead to system failure</td>
<td>Code which won’t break the functionality if faults occur</td>
</tr>
<tr>
<td>Parts using new technologies/libraries</td>
<td>Parts similar to those previously reviewed</td>
</tr>
<tr>
<td>Parts constructed by inexperienced team members</td>
<td>Reused and already reviewed parts</td>
</tr>
<tr>
<td>Code that features high code churn</td>
<td>Code with few changes</td>
</tr>
</tbody>
</table>
Change-based Code Reviews

Different Review Approach
- **Lightweight** process
- Size of reviewed code is *(should be)* **small**
- Performed **regularly** and **quickly**, mainly before code enters main branch

Shift in Focus
- From defect finding to **group problem solving**
- Prefer discussion and fixing code over reporting defects

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

Hierarchy of Review Goals
- Build a shared mental model
- Ensure sane design
- Find defects vs. understanding code
Recent Research

- Code review coverage and review participation share a 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

Expectations 4 years later

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

Empirical study outcomes

Comments in each Category

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

[Bacchelli ‘13]

[Bosu’17]
Challenges of Change-based Review

- **Delay** the shipping of implemented features
- Force reviewers to **switch context**
- 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 commit and push continuously
  - Team members see code changes in VCS and can adapt their work

- Chance of unreviewed code in repository
  - Need to/can set restrictions
- Requires branches or similar to work effectively

[https://www.devart.com/review-assistant/learnmore/pre-commit-vs-post-commit.html](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, Google

**Pros:**

- No code enters unreviewed
- Code quality standards met before commit, no 'fixes'
- No repository access for reviews
- Flexible definition of code to review (set of commits, branch, some files)

**Cons:**

- Reviewing all changes takes time
- Another complex system to handle
- Context switch to another system
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


[Rigby’13]
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)
Summary

Software Reviews
- Not a new thing, good reasons to do them (goals & motivation)
- Focus of reviews
- Different types of review techniques
  - Software Inspections
  - Change-based code reviews
- Reviewer assignment & best practices
- Reviews in industry
References


