Code Review Quality

Defining a *good* code review

Overview

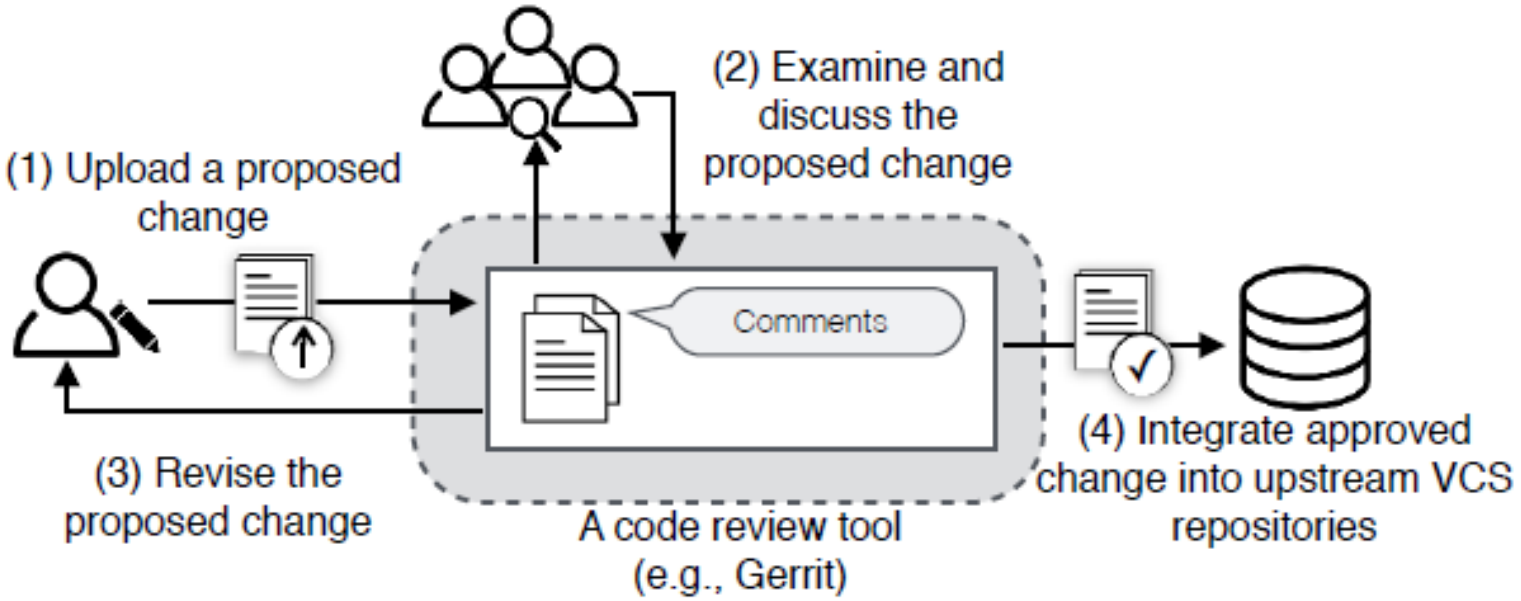
- **Introduction: A modern code review process**
  - The importance of reviews for QA
  - Strength and weaknesses
- **Tools to make (better) reviews**
  - VCS: Git - Keeping track of changes.
  - Gerrit: State of the art for git VCS.
  - Continuous inspection: SonarQube
- **How to define „good“?**
  - Factors to assess review quality
  - **Case study: How Developers see it**
- **Improve Review Quality**
- **Current research**

What is a modern code review process

INTRODUCTION

Introduction

A modern review process



Introduction

The review value - strengths

- **Coding standards compliance**
  - Code review helps to maintain consistent coding style across the company
  - Reviews find high level breaches of design rules and consistency issues.
- **Finding subtile bugs and defects early**
  - A person is able to find logical flaws and approach problems, that might be very hard to find with tests.
  - when they are cheap to fix. (Before submitting)
- **Teaching and sharing knowledge**
  - During review team members gain better understanding of the code base and learn from each other
- **Strengthens team cohesion**
  - Review discussions save team members from isolation and bring them closer to each other.

Introduction

The review value - weaknesses

- **Highly depends on reviewer's experience and code familiarity**
  - Study showed that the ‘worst’ reviewer can be ten times less effective than the best reviewer.
- **The defect coverage is hard to asses**
  - They might find a lot of different sorts of issues while **overlook some potentially serious ones.**
- **Has a frustration and conflict potential**
  - “Criticizing my code is criticizing myself” – Attitude
  - Tell and hear something for the X-times might not be fun
  - Can produce high workload and protracted discussions
- **Be considered costly as they consume developer time.**
  - Highly valuable resource

Short description of practical tools

ESSENTIAL CODE REVIEW TOOLS

Review Tools
Git

- Most common VCS
- VCS is essential to do productive code reviews
- Well documented
- Plenty of tools for integration such as
  - Gerrit (JavaEE)
  - Gitlab (Ruby on Rails)
  - Phabricator (PHP)
  - Rietveld (Python)
  - …

Review Tools

Overview

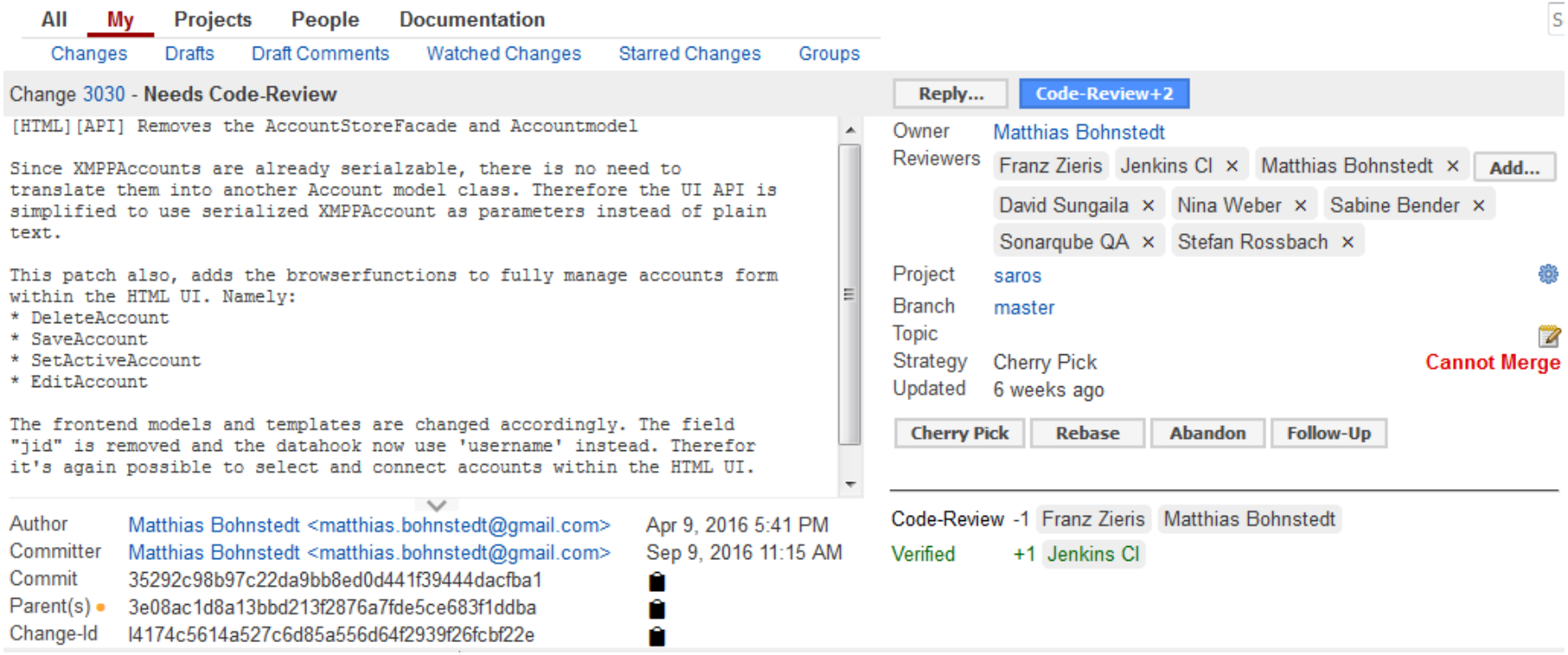
Software	Maintainer	License	VCS supported	Platforms supported	Workflow
Crucible	Atlassian	Proprietary	CVS, Subversion, Git, Mercurial, Perforce	Java	pre- and post-commit
Gerrit	Shawn Pearce	Apache v2	Git	Java EE	pre-commit
GitLab	GitLab Inc.	MIT	Git	Ruby on Rails	pre- and post-commit
Kallithea	kallithea-scm.org	GPL v3	Git, Mercurial	Python	post-commit
Patchwork	Stephen Finucane	GPL v2	VCS-agnostic	Python	mailing list
Phabricator	Phacility	Apache	Git, Subversion, Mercurial	PHP	pre- and post-commit
Review Board	reviewboard.org	MIT	CVS, Subversion, Git, Mercurial, Bazaar, Perforce, ClearCase, Plastic SCM	Python	pre- and post-commit
Rietveld	Guido van Rossum	Apache v2	Git, Subversion, Mercurial, Perforce, CVS	Python	pre-commit
RhodeCode	RhodeCode	AGPL v3	Git, Subversion, Mercurial	Python	pre- and post-commit
Understand	SciTools	Proprietary	Any	Windows, Mac OSX, Linux	pre- and post-commit
Upsource	JetBrains	Proprietary	Git, Mercurial, Perforce, Subversion	Windows, Mac OSX, Linux	post-commit

Review Tools
Gerrit

- Gerrit is a free, web-based team code collaboration tool
- It integrates closely with Git
- Provides overview about current reviews as well as diff and comment tools to do reviews

Review Tools

Gerrit exampel



Review Tools

SonarQube

- Is an open source platform for continuous inspection of code quality.
- Provides static code analyses
- Helps to automatically find common problems and code smells

Review Tools

SonarQube - Examples

```
61 65          public void connectionStateChanged(Connection connection,
62 66              ConnectionState connectionState) {
63 67
64 68              switch (connectionState) {
65 69              case CONNECTED:
```

Sonarqube QA

Major

"switch case" clauses should not have too many lines

Reduce this switch case number of lines from 8 to at most 5, for example by extracting code into methods.

Reply ... Reply 'Done'

```
66 70                  synchronized (StateRenderer.this) {
67    -                    state = new State(new Account(connection.getUser()),
68    -                        connection.getRoster(), connectionState);
   71 +                    XMPPAccount activeAccount = xmppAccountStore
   72 +                        .getActiveAccount();
```

```
71 71          isBusy: {
72 72              deps: ['connectionState'],
73 73              fn: function() {
74 74
75 75                  return this.connectionState == CS.CONNECTING || this.connectionState == CS.DISCONNEC
TING;
```

Sonarqube QA

Major

"===" and "!==" should be used instead of "==" and "!="

Replace "==" with "===".

Reply ... Reply 'Done'

What factors define a code review quality?

DEFINE CODE REVIEW QUALITY

Code Review Quality

Defining “good”

Measurement of quality and review effectiveness is hard, because

- It depends on who to ask. (Patch author, reviewer, customer)
- It depends on what factors are considered and valued the most
- Human factors have a high impact on the process, and are therefore difficult to measure

Code Review Quality

Possible factors to determinate Effectiveness

- Time spend and review rate(i.e., lines of code reviewed per hour)
- Use “defect removal” per hour

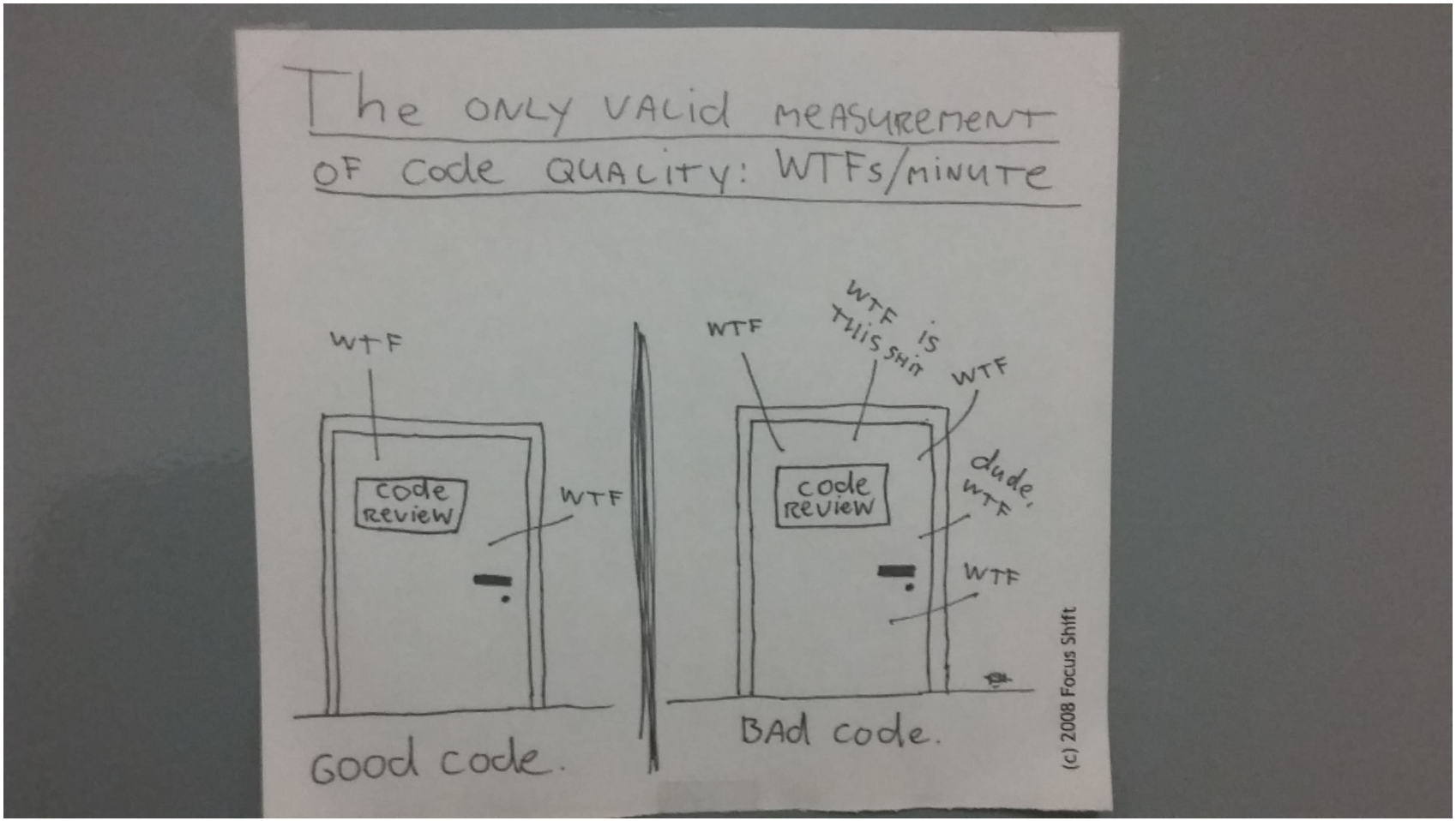
Problem

Both approaches falls short as they don’t consider factors like

- Transfer knowledge
- Resolving non-technical issues
- Improve team cohesion
- ...

Code Review Quality

The only valid measurement…



Case study of experienced Morzial developers

HOW DEVELOPERS SEE IT

Case study
Survey Design

- Participants from Morzilla project.
- Highly experienced developers
- Quantity: 88 response with 938 Quotes (23% response rate)
- Duration over 3 weeks
- Grounded theory methodology
- Open coding process: so categories, quotes, and themes emerged and evolved during the process

Case study

Research Questions

Q1: What factors do developers consider to be influential to review time and decision?

Q2: What factors do developers use to assess code review quality?

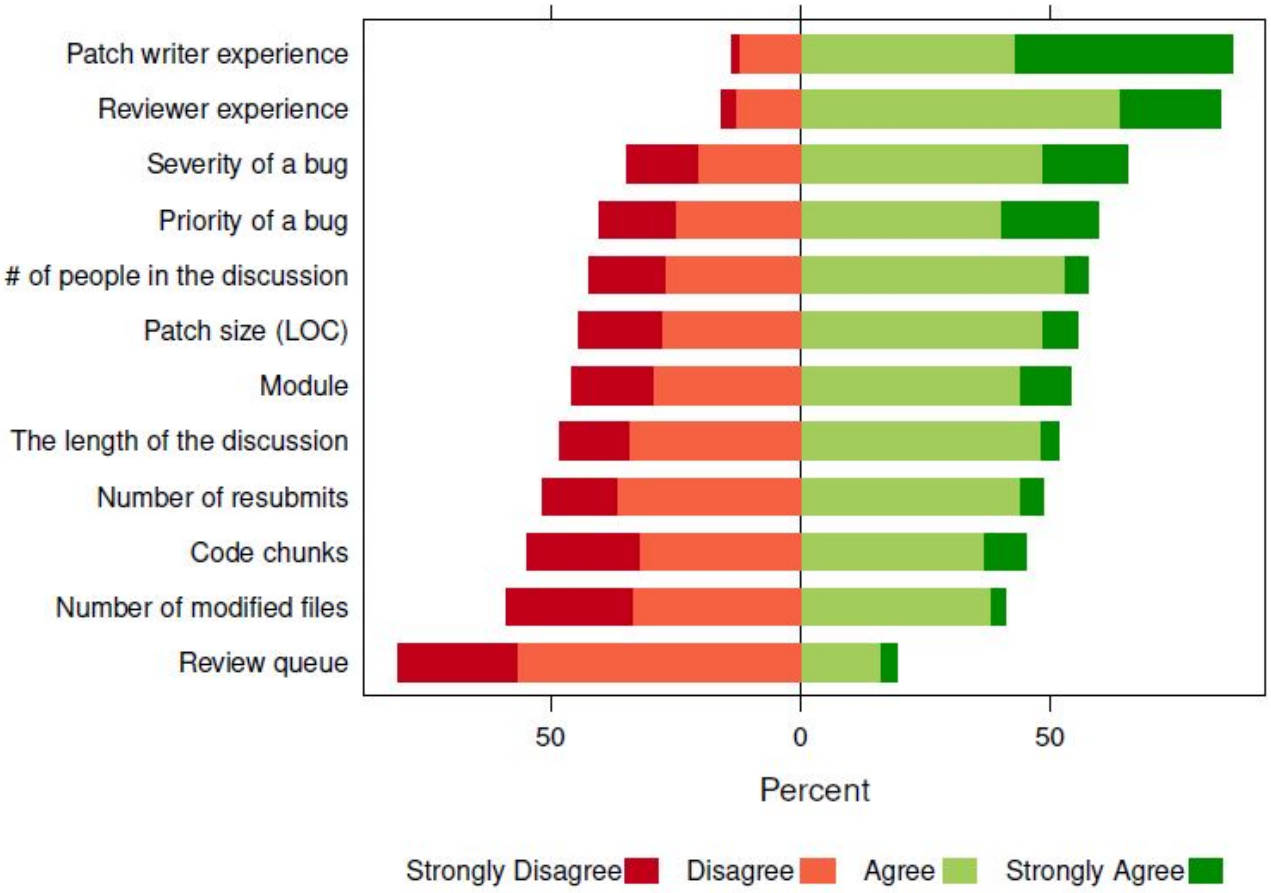
Q3: What challenges do developers face when performing review tasks?

Case study

Q1: Factors for review time

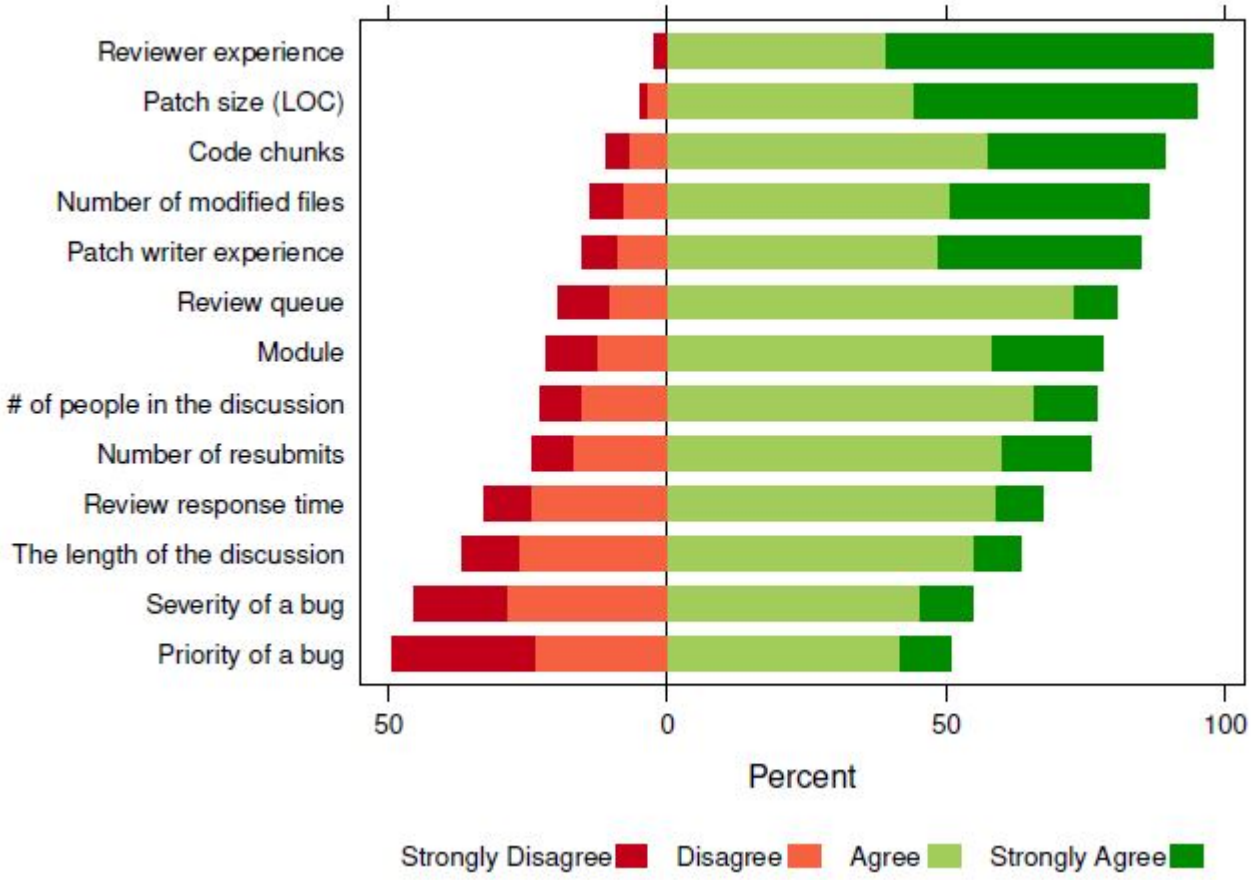
Case study

Q1: Factors for review decision



Case study

Factors that influence code review quality



Case study

Factors do assess code review quality

Can be understood as a combination of

- Patch quality – Patch author responsibility
- Review quality – Reviewer responsibility

Case study

The Reviewer perspective: Factors for patch quality I

Good code quality

- Coding style
  - "The names of things need to be descriptive"
  - *"consistent indentation and style"*
  - Readability, compactness, maintainability
- Presence of meaningful comments

  "I'm looking for a thoughtful summary that instructs me, reviewer, what is going on and what to expect"

Case study

The Reviewer perspective: Factors for patch quality II

Correctness

Does the patch

- *actually implements what it intends to do?*
- *handle all cases?*
- *Introduce any other bugs or ambiguity?*
- *use ´clever´ tricks that other will struggle to understand?*

In a nutshell, a high quality patch usually provides a robust solution for the problem.

Case study

The Reviewer perspective: Factors for patch quality III

Low change complexity

- *If the patch is addressing 3 or 4 different things it is lower quality than 3 or 4 separate patches for the individual issues*
- *Small, focused changes are easier to assess than large ones. If bug rate is proportional to lines of code, quality is inversely proportional to patch size. So, small patches preferred.*

Case study

The Reviewer perspective: Factors for patch quality IV

Tested code is good code

- *Changes that are accompanied by tests are much more likely to be accepted*
- Actual completeness of tests is also important *Thoroughness of tests included in patch.*

Case study

The Reviewer perspective: Factors for patch quality V

Change scope and rationale

- Reviewers first look for the actual appropriateness of changes
- *Does the feature fit in with the product (for patches submitted out of the blue)?*
- *Not all fixes or improvements are a good idea to actually land, even if they're correct*

Case study

The Committer perspective: Define a a well-done code review I

Clear and thorough feedback

- Feedback is not only about code formatting and style
- Actually describes the issue and not only points to it:
  “Worst code ever!” (But I don’t tell you why)

Case study

The Committer perspective: Define a a well-done code review II

Provides constructive advice

- *Points out major correctness issues first, and points our minor issues later*
- Highlighting potential problems ... and how to fix them
- Is delivered via proper communication:
  Good code reviews are dialogues between the reviewer and patch author

Case study

The Committer perspective: Define a a well-done code review III

Is done by the correct reviewer

- Who has the domain knowledge to properly evaluate the change: *Enough domain knowledge is always the first criteria for a well-done code review*

- He provides mentoring and training for patch authors:

  [He] help the author of the patch become a better programmer in the long term

Case study

The Committer perspective: Define a a well-done code review IV

Human Factors play a crucial role

Reviewers needs to

- *be punctual and tactful*
- *supportive, yet strict*
- *be helpful and encouraging, especially when rejecting a patch*
- *establish clear and open-minded communication*

Take away tips for making better code reviews

HOW TO IMPROVE

Improve Review Quality

Questions to ask while doing a Review

- Do I am qualified to review this?
- Does the code solve the described problem?
- Are all relevant cases considered?
- Does it introduce new problems?
- Does it needlessly reinvent the wheel?
  - Does it make use of the codebase?
- Is the solution understandable and reasonable?
  - Does the commit Message provide guidance?
  - Is the code well enough documented?
- Is there a easier (less risky) way to do it?

Improve Review Quality

Tips for Reviewers

- Be precise and provide suggestions
  - Communication is key for motivation
- Don’t fall into the “perfectionist” pit
  - There is a fine line between being thoughtful and perfectionistic
  - Focused on the important aspects and than go into “quality of life” changes
  - A non “perfect” solution might work, and could be worth the risk
- Don’t accept patches because of time pressure or just because you trust the author.
- Be encouraging when ever you can!

Improve Review Quality

Tips for patch authors I

Before submitting

- Provide guideline in the commit message
  - Describe the problem and approach in few words in the commit message.
- Does my code compile and pass all tests
  - Don't waste Reviewers time with unfinished /untested code submits
  - Should I write new tests?
    Test strengthen the confidence of Reviewers in your code!
- Separation of concerns
  - A single patch should address a single problem
  - Keep the patch size low
  - If you have serval "and"; "also", or "furthermore" chances are good that you are mixing different concerns and you should split your patch
- Is my code documented
  - Describe WHY not WHAT.
- Follow the code convention
  - Take the time to ensure code quality

Improve Review Quality

Tips for patch authors II

After submitting

- Invite appropriated Reviewers
  - More isn't better
- Take the feedback
  - Your code isn't your child, it's okay to be criticised
  - Be open minded for suggestion, but know when to "defend" your solution
- Try to clarify (and solve!) addressed problems before submitting a new change set.
  - Been thoroughly saves time and frustration for both reviewer and author

Current Research

- Next Gen-Reviewsystem
  - Navigation, better diffs, IDE quality of live functions
- Redefine Code Ownership
  - Often used in research, should consider Reviews
- Automatic Decomposition of Changesets
  - Create smaller patch chunks might improve Reviews
- Reviewer Recommender
  - Finding the “best” Reviewer.
- Knowledge database to speed up common review tasks
  - Save time and frustration by automatically insert common error descriptions.

Source and literature

1. Oleksii Kononenko, Olga Baysal , Michael W. Godfrey:
   Code Review Quality: How Developers See It
   https://plg.uwaterloo.ca/~migod/papers/2016/icse16.pdf

2. Patanamon Thongtanunam, Shane McIntosh , Ahmed E. Hassan, Hajimu Iida**:**
   Revisiting Code Ownership and its Relationship with Software Quality in the Scope of Modern Code Review
   http://sail.cs.queensu.ca/Downloads/ICSE2016_RevisitingCodeOwnershipAndItsRelationshipWithSoftwareQualityInTheScopeOfModernCode%20Review.pdf

3. **Social aspects of a continuous inspection platform for software source code**
   CHASE '08 Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering p. 85-88

4. L. Hatton. **Testing the value of checklists in code inspections**. IEEE Software, 25(4):82-88, 2008.