Code Review

Adrianna Keeney & Sofia Lizotte

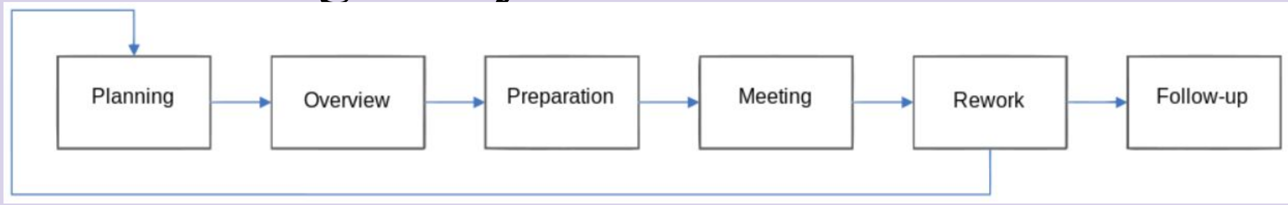
Introduction

How Code Reviews Started - Fagan-Style

https://en.wikipedia.org/wiki/Fagan_inspection

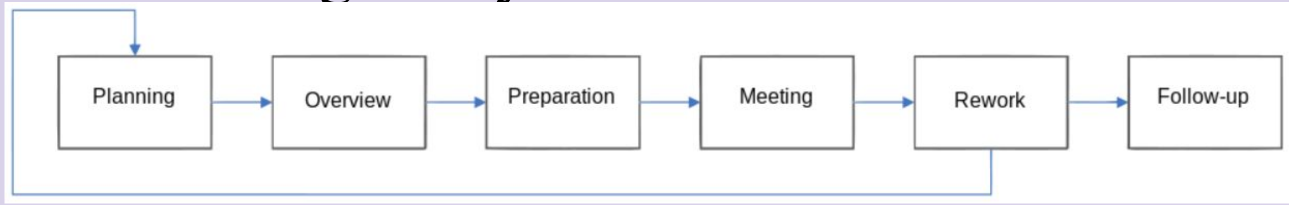
- **Fagan inspection** : process of trying to find defect in documents during various phases of software development
    - Certain activity with pre-specified entry and exit criteria (think unit tests)
    - Group review method
- Goal : check for defects and fix them earlier in the software development process and iteratively following updates; benefit is having a higher quality product in the end
- Different Roles:
    - Author/Designer/Coder: the person who wrote the document
    - Reader: paraphrases the document
    - Reviewers: reviews the document from a testing standpoint
    - Moderator: responsible for the inspection sessions, functions as a type of coach
- Document Types:
    - Low-level document: Current implementation not meeting final requirements goal
    - High-level document: Final requirements goal; Finished implementation

How Code Reviews Started - Fagan-Style



- Planning:
  - Prepping materials
  - Arranging of participants
  - Arranging of meeting place
- Overview:
  - Group education of participants on the materials under review
  - Assignment of roles
- Preparation:
  - Participants review the item to be inspected and supporting material to prepare for the meeting noting any questions or possible defects
  - Participants prepare their roles
- Inspection Meeting:
  - Actual finding of defect(s)

How Code Reviews Started - Fagan-Style



- Rework:
  - The step in software inspection in which the defect(s) found during the inspection meeting are resolved by the author, designer, or programmer. On the basis of the list of defect(s) the low-level document is corrected until the requirements in the high-level document are met
- Follow-Up:
  - All defects found in the inspection meeting should be corrected. The moderator is responsible for verifying that this is done. They should verify that all defects are fixed and no new defects are inserted while trying to fix the initial defects. It is crucial all defects are corrected, as the cost of fixing them in a later phase can be 10-100 times higher compared to current costs.

Modern Code Reviews

https://rethought.se/research/modern-code-reviews/#:~:text=Modern%20Code%20Review%20is%20a,the%20feedback%20and%20submits%20again.

‘Quality assurance practice where a code change is reviewed by 1 or more reviewers. The review feedback is sent to the author who then modifies the code based on the feedback and submits again. This process continues until the open source community or software organizations’ requirements are met…’

Google Case Study

https://dl.acm.org/doi/pdf/10.1145/3183519.3183525

- Modern: informal (contrasting the Fagan-style), tool-based, asynchronous, focused on reviewing code changes
- Flow: Creating, Previewing, Commenting, Addressing Feedback, and Approving (‘LGTM’) ‘Looks Good To Me’

The Difference

Fagan

- Heavyweight approach
- Formal inspection
- Over-the-Shoulder
- Email pass around
- Pair-Programming
- Review Supported by a Tool

Modern

- Lightweight approach: avoiding mainly protocols, meeting, guidelines, or other boring tasks such as a checklist
- Different expectations related to readability and maintainability
- Education (learning/teaching about something new in the code), maintaining norms (company preferences), gatekeeping (maintaining software architecture check points, design choices, and software architect tread-off), and accidental prevention (defect detection and other quality issues)

Code Review Checklist

https://github.com/mgreiler/code-review-checklist

QR for following along

https://github.com/sophie177/321Presentation

How to Review Code: Pull Requests

Although these studies used specific tools, an alternative way to perform code reviews is to use pull requests:

- Create a branch for the feature/bug you are working on
- Once you are done, commit your work and push it to your repository
- Create a pull request from your branch
- Describe what the work is about
- Select reviewers
- Address the reviewers comments until the code is accepted
- The reviewer/verifier can then merge the code to the master

Other options:

- Simple Commenting on code

Characteristics of Useful Code Reviews

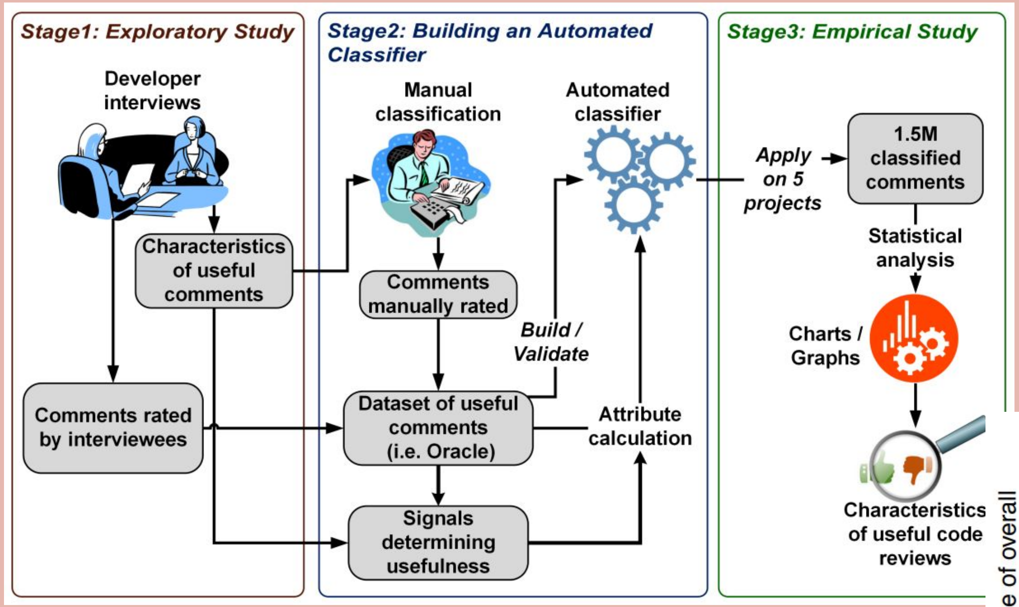
https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bosu2015useful.pdf

- Jan. 2015 : 50,000+ Microsoft employees practiced tool-based code review, also known as contemporary code reviewing : lightweight, informal, tool-based
- Survey recorded developers average 6 hours per week preparing code for review or reviewing others’ code
- Main building blocks of code reviews:
    - Comments that reviewers add as feedback
        - Bugs, improved ideas for solving..etc.
    - Suggestions for change for author to address
- Goal:
    - Code change is free from defects
    - Follows team conventions
    - Solves a problem in a reasonable way
    - High quality code

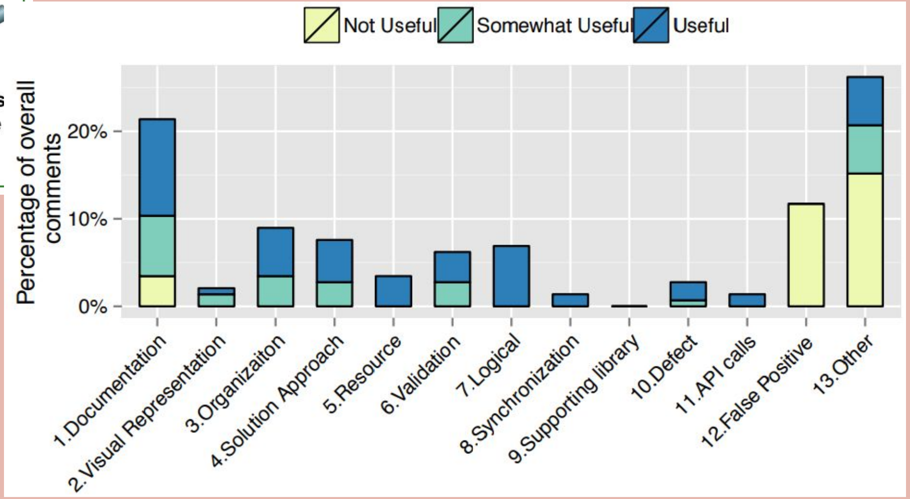
Characteristics of Useful Code Reviews

- Code Review Tools: **CodeFlow**, Gerrit, Phabricator, ReviewBoard…etc.
  - These tools work similarly: author submits change for review, reviewers are notified and can examine the code change through the tool
  - Each updated change for feedback is referred to as an *iteration*
- Research:
  - 7 devs from 4 different projects based on varying levels of development and code-review experience
  - Semi-structured individual 3-phased 30 minute interviews
  - Phase 1: 5 min.; job role, experience, what role comments play during review
  - Phase 2: 20 min.; shown 20-25 randomly selected review comments on author’s code; interviewee (author) rated comments based on usefulness and category
  - Phase 3: 5 min; give other types of of useful comments not covered

Characteristics of Useful Code Reviews


Stage1: Exploratory Study
Developer interviews
Characteristics of useful comments
Comments rated by interviewees
Stage2: Building an Automated Classifier
Manual classification
Automated classifier
Comments manually rated
Build / Validate
Dataset of useful comments (i.e. Oracle)
Attribute calculation
Signals determining usefulness
Stage3: Empirical Study
Apply on 5 projects
1.5M classified comments
Statistical analysis
Charts / Graphs
Characteristics of useful code reviews

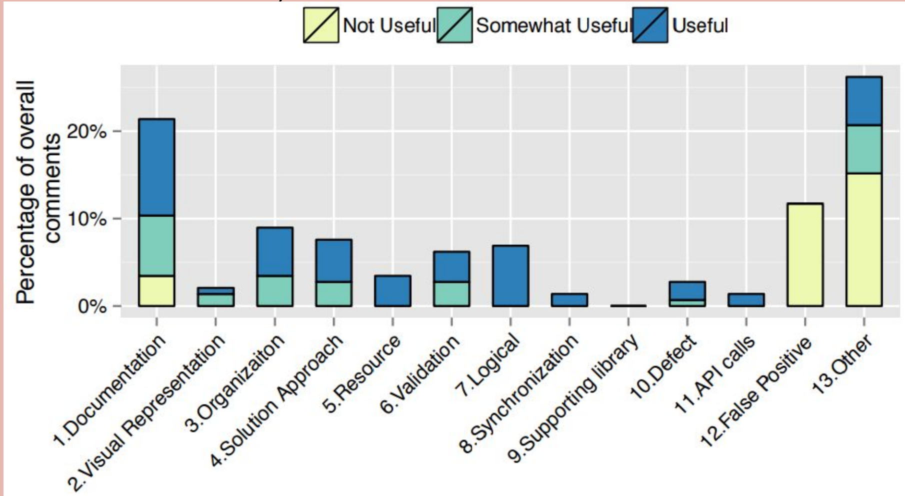
1) Rate the given comment on a 3-point scale:
   1: *not useful*
   2: *somewhat useful*
   3: *useful*
2) Briefly explain why they selected that rating
3) Assign to category



- The 7 interviewees rated and categorized 145 review comments in total

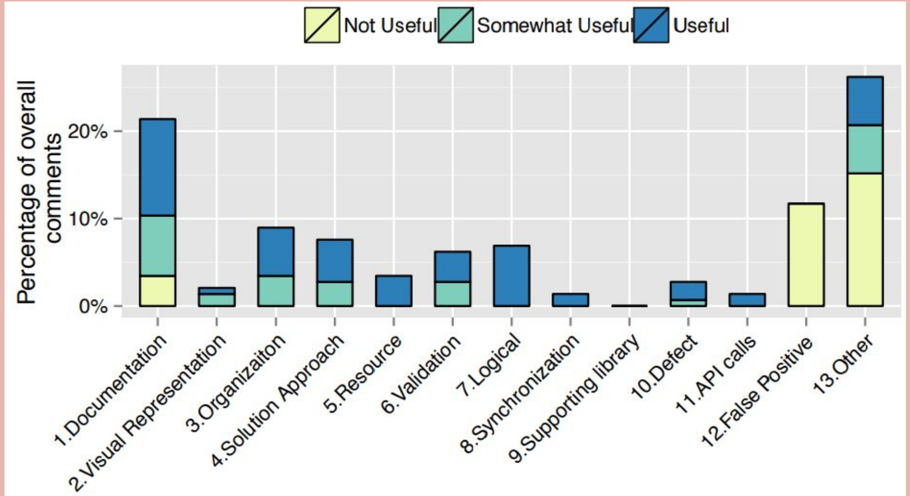
Characteristics of Useful Code Reviews

- Almost 69% of comments were either *useful* or *somewhat useful*
- Most comments identifying functional defects were rated as *useful* (fig. 5-11 on x-axis)
- More than 60% of the *somewhat useful* comments belong to the first 4 categories:
  - Documentation in code
  - Visual representation of the code (blank lines, indentation…etc.)
  - Organization of the code (how functionality is divided into methods)
  - Solution approach
- These 4 are in a class termed 'evolvability defects' represent issues that affect future development effort rather than runtime behavior



Characteristics of Useful Code Reviews

- Most (73%) fell into one of predefined categories labeled categories
- Most of the *not useful* comments are either false positives (when a reviewer incorrectly indicates a problem in the code) or don’t fall into a predefined category
  - These included:
    - Questions from reviewers (not useful)
    - Praise for implementation (not useful)
    - Suggestions for alternate output or error messages (useful)
    - Discussions on design or new features (useful)



Characteristics of Useful Code Reviews

- *Useful* Comments:
  - Identification of any functional issues
  - Validation issues or alternate scenarios where the current implementation may fail
  - Code reviews are very useful for the new members to learn the project design, constraints, available tools, and APIs
- *Somewhat Useful* Comments:
  - "Nit-picking issues" (indentation, comments, style, identifier naming, and typos); these were rated between *useful* and *somewhat useful*; <u>these were seen as non-essential, but the identification and resolution of these types of issues helped long-term project maintenance</u>
  - Sometimes review feedback / questions help the authors to think about an alternate implementation or a way to refactor the code to make it more comprehensible, even if the current implementation is correct
- *Not Useful* Comments:
  - Asking questions to understand the implementation; possibly useful for knowledge dissemination, but do not improve the code
  - Praisal of code segments; may help in building positive impressions and encourage good coding
  - Pointing out work that needs to occur in the future, but not during the current development cycle

Characteristics of Useful Code Reviews

Final Result Takeaways:

- Results suggested review effectiveness decreases with the number of files in the change set. Therefore, recommendation is submitting smaller and incremental changes whenever possible
- Special care should be taken when non-code files, such as configuration or build files, are included in reviews, as these elicit less useful feedback

‘The primary goal of code reviews is most often to improve the quality of software by identifying defects, identify better approaches for a source code change, or help to improve the maintainability of code’

- Secondary benefits: knowledge dissemination and team awareness
    - Although beneficial, most developers see these types as less useful

https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/bosu2015useful.pdf

Expectations, Outcomes, & Challenges

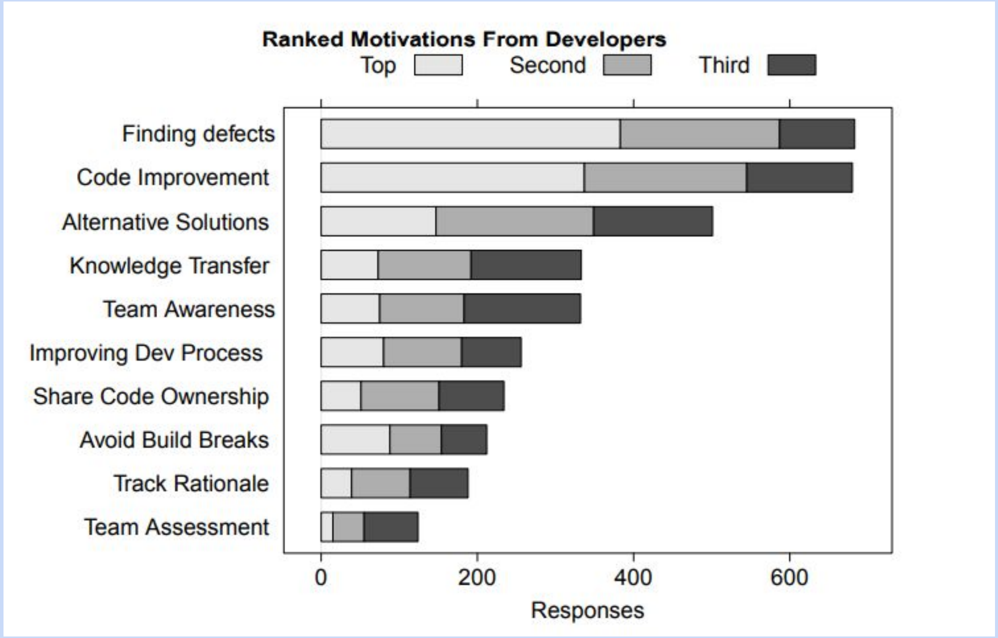
“Defect related comments comprise a small proportion and mainly cover small logical low-level issues. On the other hand, code review additionally provides a wide spectrum of benefits to software teams, such as knowledge transfer, team awareness, and improved solutions to problems.”

RQ1. What are the expectations of a code review?

RQ2. What are the outcomes of code reviews?

RQ3. What are the challenges of code reviews?

What are the expectations of a code review ?
(Are they consistent between managers, developers, and testers?)



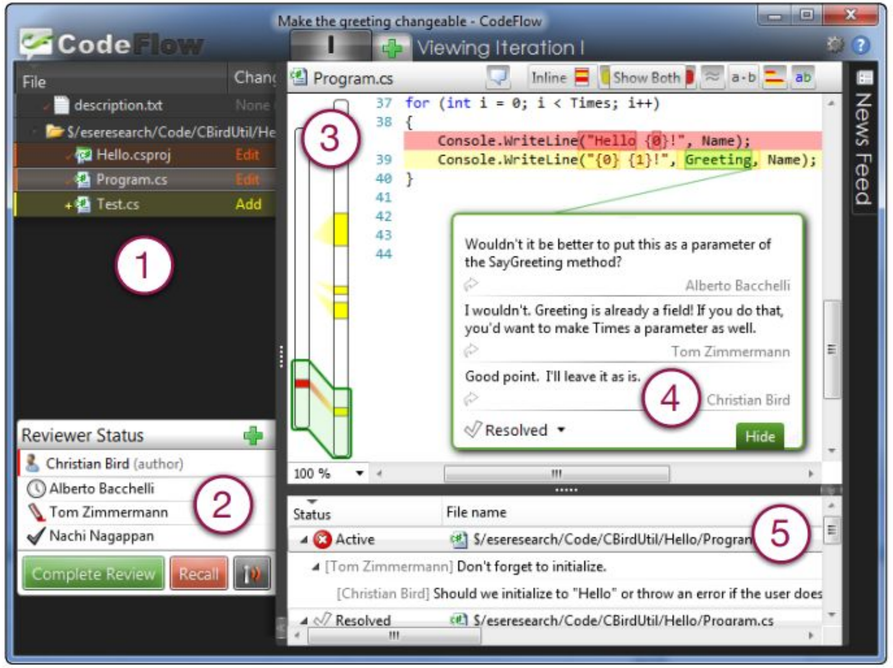
Formatting (syntax) != finding defects!

Both programmers and managers share motivations, with Managers placing more emphasis on "Share Code Ownership" and "Team Assessment".

Sharing of code ownership correlates with the average programmer having a broader understanding of the project and encourages 'understudies' of those who would otherwise specialize in a niche functionality.

" …(Managers want to) … delete any 'rigid sense of ownership' that might develop over chunks of code"

Outcomes



Make the greeting changeable - CodeFlow

File	Change
description.txt	None
$/eseresearch/Code/CBirdUtil/He	
Hello.csproj	Edit
Program.cs	Edit
Test.cs	Add

Program.cs

```
for (int i = 0; i < Times; i++)
{
    Console.WriteLine("Hello {0}!", Name);
    Console.WriteLine("{0} {1}!", Greeting, Name);
}
```

Wouldn't it be better to put this as a parameter of the SayGreeting method? — Alberto Bacchelli

I wouldn't. Greeting is already a field! If you do that, you'd want to make Times a parameter as well. — Tom Zimmermann

Good point. I'll leave it as is. — Christian Bird

Resolved

Reviewer Status
- Christian Bird (author)
- Alberto Bacchelli
- Tom Zimmermann
- Nachi Nagappan

Status	File name
Active	$/eseresearch/Code/CBirdUtil/Hello/Progran
[Tom Zimmermann] Don't forget to initialize.	
[Christian Bird] Should we initialize to "Hello" or throw an error if the user does	
Resolved	$/eseresearch/Code/CBirdUtil/Hello/Program.cs

Fig. 1. CodeFlow, the main code review tool used by developers at Microsoft.

1. **Code Improvements**
   30% of comments, split three ways evenly between: (i) best code practices, (ii) optimizing redundant or inefficient logic, and (iii) improving readability.

2. **Defect Finding**
   14% of comments, the majority on logical issues with a minority identifying security concerns and exception handling.

3. **Knowledge Transfer**
   Hard to quantify: often included in comments identifying a logic issue, reviewers provided links to resources or clarifying language intended to improve understanding of the change.

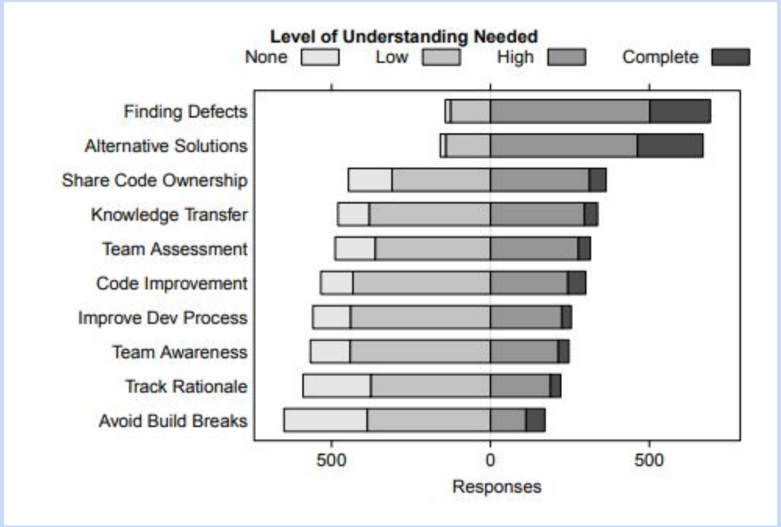
Overcoming the Challenges

- Quantity != Quality. The average code review was full of low-quality suggestions which addressed syntax and readability, not logic.

Possible Solution: Allow those file owners to review their own code occasionally.

- Communication: In order for the reviewee to understand changes, up to 50% of the time they would seek in-person conversations.

Solution: Code review cannot be limited to tools.



Remember, formatting != finding defects!

The biggest challenge faced is understanding, double-tested by comparing reviews by those familiar or unfamiliar with a file:

“The main difference with file owner comments is that they are substantially deeper, more detailed and insightful.”

What Makes Code Reviews Confusing ?

Code reviews can delay incorporation of a change into the code base, slowing down the development process. This delay is partly caused by reviewers not understanding/being uncertain or confused about the intention, behavior, or effect of a code change.

RQ1. What are the reasons for confusion in code review?

RQ2. What are the impacts of confusion in code reviews?

RQ3. How do developers cope with confusion during code reviews?

What are the Major Causes of Confusion in Code Reviews ?

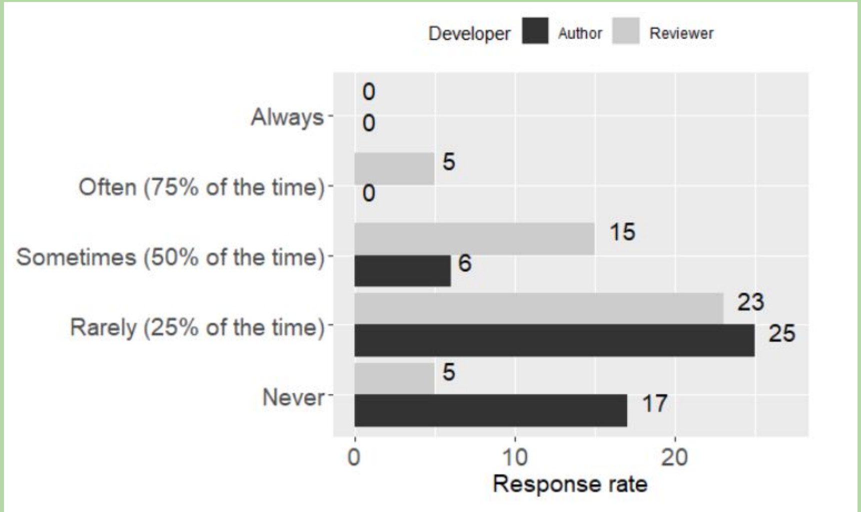


Fig. 3. Frequency of confusion for developers and reviewers.

Disagreement among the developers is the prevalent topic, e.g., "[...] If actual change has a big difference from my expectation, I am confused.".

The second most cited reason is the ***misunderstanding*** of the message's intention, e.g., "Sometimes I don't understand general meaning (need to read several times to understand what person means)" .

Six reasons are related to the link between the developer and the artifact. The most popular one is the ***lack of familiarity with existing code***, e.g., "Lack of knowledge about the code that's being modified."

Followed by the ***lack of programming skills***, e.g., "sometimes I'm confused because I'm missing some programming", and the ***lack of understanding of the problem***, e.g., "I'm embarrassed to admit it, but I still don't understand this bug.".

What *not* to do…

1. **Blindly approve changes**

Interestingly, blind approval is also a strategy developers use to cope with confusion, i.e., it is not just an impact, e.g., "assume the best, (of the change)" (R34).

2. **Abandon the project - or, abandon documentation**

Early detection of confusion [17] becomes crucial for preventing contributors' burnout and loss of productivity [48]. Such situations might even cause the abandonment of the project, which is undesirable as it eventually leads to undesired turnover; since **developers focusing on documentation are more susceptible to turnover than those working on code** [49], project abandonment is likely to exacerbate lack of documentation further increasing confusion within the project

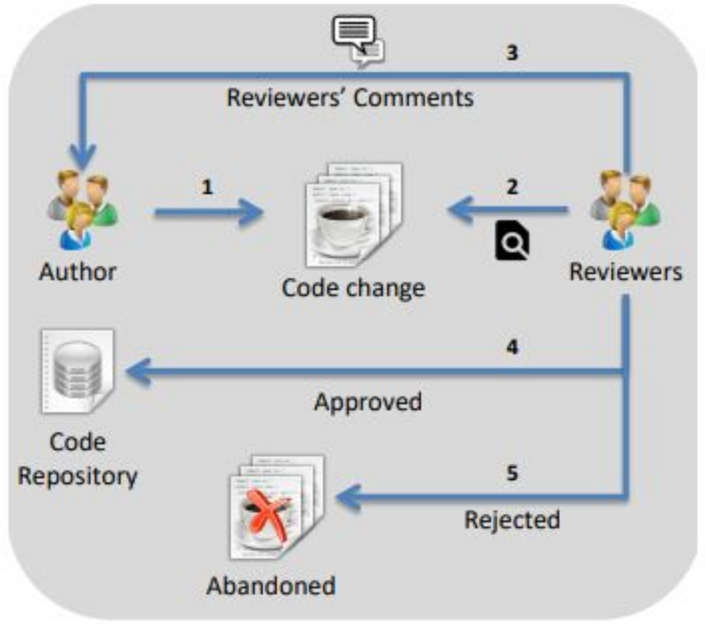


Fig. 1. Code review process.

Resolving Confusion - Better Alternatives

Asking a question on StackOverflow be like
That's a stupid question
Downvotes
Read the documentation
Here's what you did wrong, this snippet should help you
You should stop using that framework
That one guy who actually read the question
Me

1. ***improve the organization*** of work, such as making clearer commit messages
2. ***spending more time*** and delaying the code review in order to commit better work
3. ***make the code change smaller***, e.g., "Also I ask large changes to be broken into smaller pieces", and ***asking for clarification*** - asking the reviewer to explain unclear points.
4. Being ***open to critical review comments***, and ***providing polite criticism***. (Not responding like certain StackOverflow users!)
5. ***Early admittance of confusion*** (for the author) and ***explaining your reasoning*** step-by-step (for the reviewer)

Introductory Examples of Code Review

A great example of C# code review

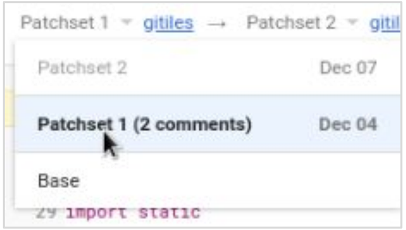
https://dzone.com/articles/code-review-c-code

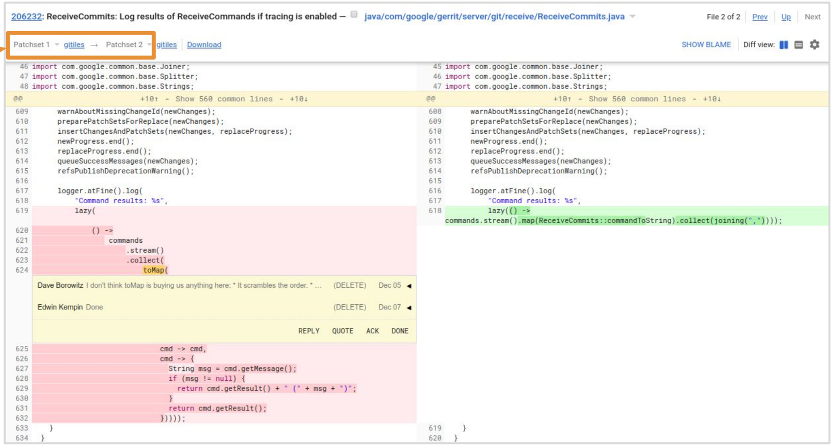
Couchbase:

https://review.couchbase.org/c/couchbase-java-client/+/21805

Comparing Patch Sets

Patch Set Selector:





206232: ReceiveCommits: Log results of ReceiveCommands if tracing is enabled —
java/com/google/gerrit/server/git/receive/ReceiveCommits.java
File 2 of 2
Prev
Up
Next
Patchset 1
gitiles
→
Patchset 2
gitiles
Download
SHOW BLAME
Diff view:

```
import com.google.common.base.Joiner;
import com.google.common.base.Splitter;
import com.google.common.base.Strings;
@@                +10↑ - Show 560 common lines - +10↓
    warnAboutMissingChangeId(newChanges);
    preparePatchSetsForReplace(newChanges);
    insertChangesAndPatchSets(newChanges, replaceProgress);
    newProgress.end();
    replaceProgress.end();
    queueSuccessMessages(newChanges);
    refsPublishDeprecationWarning();

    logger.atFine().log(
        "Command results: %s",
        lazy(
            () ->
                commands
                    .stream()
                    .collect(
                        toMap(
```

Dave Borowitz I don't think toMap is buying us anything here: * It scrambles the order. * ... (DELETE) Dec 05

Edwin Kempin Done (DELETE) Dec 07

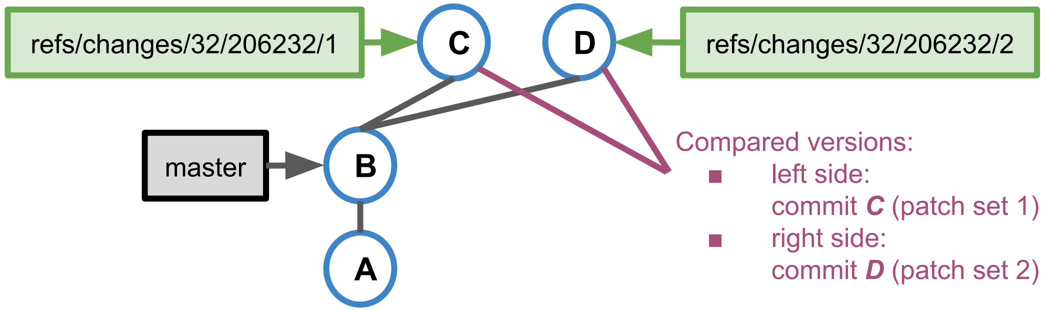
REPLY QUOTE ACK DONE

```
                            cmd -> cmd,
                            cmd -> {
                              String msg = cmd.getMessage();
                              if (msg != null) {
                                return cmd.getResult() + " (" + msg + ")";
                              }
                              return cmd.getResult();
                            }))));
  }
}
```

```
import com.google.common.base.Joiner;
import com.google.common.base.Splitter;
import com.google.common.base.Strings;
@@                +10↑ - Show 560 common lines - +10↓
    warnAboutMissingChangeId(newChanges);
    preparePatchSetsForReplace(newChanges);
    insertChangesAndPatchSets(newChanges, replaceProgress);
    newProgress.end();
    replaceProgress.end();
    queueSuccessMessages(newChanges);
    refsPublishDeprecationWarning();

    logger.atFine().log(
        "Command results: %s",
        lazy(() ->
commands.stream().map(ReceiveCommits::commandToString).collect(joining(","))));
  }
}
```

- Users that have previously reviewed the old patch set likely want to see what has changed with the new patch set. They can do so by comparing the old and new patch set.



Compared versions:

- left side:
  commit ***C*** (patch set 1)
- right side:
  commit ***D*** (patch set 2)

Demo Time!

1. Go to this link : **https://github.com/sophie177/321Presentation**
2. For participation points, make a branch and name it ` firstName_lastNameInitial`
   a. Example branch : Adrianna_K
3. Pull up the code review checklist:

   https://github.com/mgreiler/code-review-checklist

   and make changes or comments according to what you find!

Venera’s Code Review Examples

Give them a better feel of what is expected in a code review, for example:
Documentation of Professional Reviews:

https://gerrit-review.googlesource.com/Documentation/user-review-ui.html

https://gerrit-review.googlesource.com/Documentation/user-review-ui.html#inline-comments

https://docs.google.com/presentation/d/1C73UgQdzZDw0gzpaEqIC6SPujZJhqamyqO1XOHjH-uk/edit#slide=id.g4d6c16487b_1_710

[link] A crowdsourced informal code review

https://codereview.stackexchange.com/questions/127515/first-c-program-snake-game