

*CS5351 2024/25*

# Modern Code Review

---

Dr W.K. Chan

Department of Computer Science

Email: `wkchan@cityu.edu.hk`

Website: `http://www.cs.cityu.edu.hk/~wkchan`

# Outline

---

- ◆ What is Code Review?
- ◆ Why do we do it?
- ◆ Code Inspection
- ◆ Modern Code Review
- ◆ Background readings
  - [https://en.wikipedia.org/wiki/Fagan\\_inspection](https://en.wikipedia.org/wiki/Fagan_inspection)
  - <https://ieeexplore.ieee.org/document/6606617/>
    - downloadable when connecting to the campus network
- ◆ Try a code review tool by a group of students (Ex MCR-2)

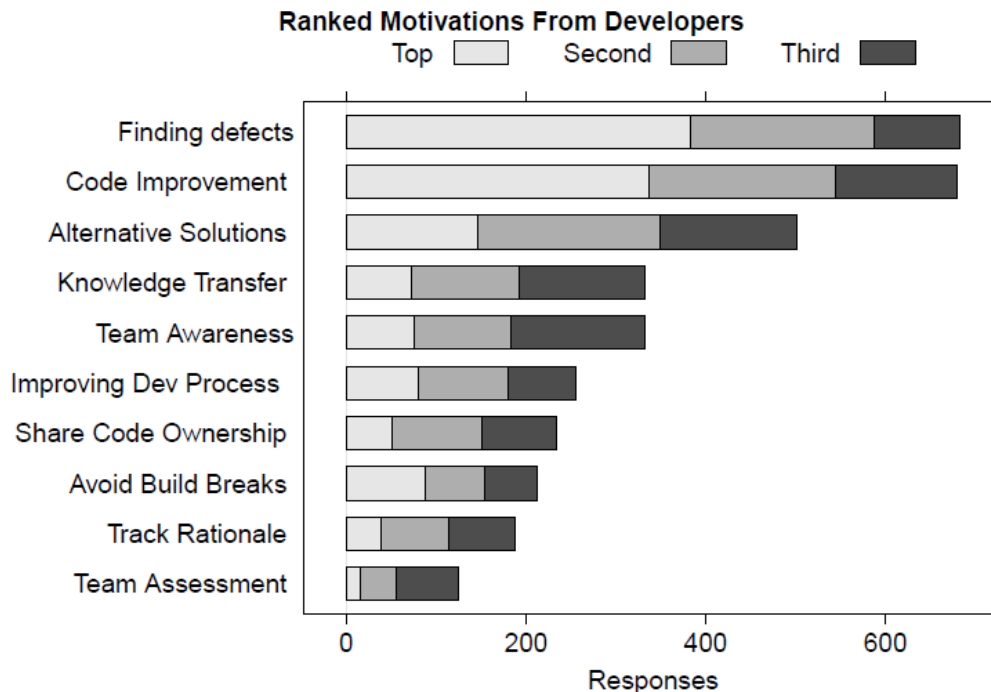
# What is Code Review

---

- ◆ Performing human inspection on the source code by developers other than the author of the code.
- ◆ is a well-established software engineering practice
- ◆ aim at *maintaining and promoting* source *code quality*, as well as *sustaining* the development *community* by means of *knowledge transfer* of design and implementation solutions applied by others [12]

# What is Code Review

- ◆ Performing human inspection of the source code by developers other than the author of the code.



Apart from finding bugs, code review has **many other purposes**: code improvement, alternative solutions, knowledge transfer, team building, and so on.

Fig. 3. Developers' motivations for code review. [1]

# Home Reading

---

[1] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 712-721.

<https://ieeexplore.ieee.org/document/6606617/>

You can get a copy of the article using a computer in CityU.

# Different Levels of Formality

---

## ◆ Code walkthrough

- informal meeting where the programmer leads the review team through his/her code, and the reviewers try to identify faults
- More effective if the programmer and reviewer are not the same person
- Sometimes, integrate with Pair Programming

## ◆ Code Inspection

- Formal meeting
- Effective to find bugs in code.
- **Home Reading:** [https://en.wikipedia.org/wiki/Fagan\\_inspection](https://en.wikipedia.org/wiki/Fagan_inspection)

## ◆ Modern Code Review

```

bool isWin(char game[3][3]){
    bool win = false;
    if (game[0][0] == game[0][1] && game[0][1] == game[0][2] && (game[0][0] == 'X' || game[0][0] == 'O')) win = true;
    if (game[1][0] == game[1][1] && game[1][1] == game[1][2] && (game[1][0] == 'X' || game[1][0] == 'O')) win = true;
    if (game[2][0] == game[2][1] && game[2][1] == game[2][2] && (game[2][0] == 'X' || game[2][0] == 'O')) win = true;
    // column
    if (game[0][0] == game[1][0] && game[1][0] == game[2][0] && (game[0][0] == 'X' || game[0][0] == 'O')) win = true;
    if (game[0][1] == game[1][1] && game[1][1] == game[2][1] && (game[0][1] == 'X' || game[0][1] == 'O')) win = true;
    if (game[0][2] == game[1][2] && game[1][2] == game[2][2] && (game[0][2] == 'X' || game[0][2] == 'O')) win = true;
    // diagonal
    if (game[0][0] == game[1][1] && game[1][1] == game[2][2] && (game[0][0] == 'X' || game[0][0] == 'O')) win = true;
    if (game[0][2] == game[1][1] && game[1][1] == game[2][0] && (game[0][2] == 'X' || game[0][2] == 'O')) win = true;
    return win;
}

```

```

int main(){
    int i, j;
    char game[3][3] = { ' ' }; // Tic-tac-toe
    char player1 = 'X';
    char player2 = 'O';
    bool turn = true; // false for player 1's turn, true for player 2's turn. Player 1 first.
    cout << "X = Player 1" << endl << "O = Player 2" << endl;
    for (int n=0; n<9; n++){
        turn = !turn; // use the not-operator to change true to false or false to true.
        if (turn == false)
            cout << "Player 1: ";
        else
            cout << "Player 2: ";
        cout << "Which cell to mark? i:[1..3], j:[1..3]: ";
        cin >> i >> j;
        if (turn == false)
            game[i][j] = 'X';
        else
            game[i][j] = 'O';
        if (isWin(game)){
            cout << "Win!" << endl;
            break; // need to terminate the problem
        }
    }
    if (i==3) // all celles with i=0 to 2 have been inputted above but no winner yet
        cout << "Tie!" << endl;

    // show the game to console
    cout << game[0][0] << " " << game[0][1] << " " << game[0][2] << endl;
    cout << game[1][0] << " " << game[1][1] << " " << game[1][2] << endl;
    cout << game[2][0] << " " << game[2][1] << " " << game[2][2] << endl;
    return 0;
}

```

## Code walkthrough:

isWin() aims to do what and so ....

Main() performs “these” in these few lines and “those” in those lines

# Code Walkthrough

---

- ◆ Result from an old paper [21]
  - Similar to testing in effectiveness for finding bugs
  - But code walkthrough requires more human effort: less effective
- ◆ In recent years, developers have become aware of technical debts (non-bug issues), where testing is ineffective in addressing them.
  - E.g., readability, reusability, maintainability, etc.
- ◆ The remote collaboration raised from the needs of “work from home” makes code walkthrough by visually impaired developers join code walkthroughs remotely. Some strategies proposed are shown on the next slide.



# e.g., Code Walkthrough [22]

Table 3: Use of audio cues to convey awareness indicators in Follow mode (unless specified otherwise in the row)

Awareness Information	Non-Speech Indicator	Non-Speech Indicator Frequency	Speech Indicator	Speech Indicator Frequency	Built-in Visual Indicator
Viewport scrolls	Click wheel sound	Every scroll event	None	None	Screen scrolls
Scroll direction	Falling or rising tone depending on direction	When scrolling stops	None	None	Can be inferred from the scrolling viewport
Current Viewport	None	None	"Lines X to Y on screen"	When scrolling stops	Visible on screen
Cursor moves by single line to line N	Keyboard click	Every cursor move	"Line N"	1.5 seconds after cursor moves end	Cursor moves on screen
Cursor moves multiple lines to line N	Falling or rising tone depending on move direction	Every event	"Line N"	1.5 seconds after cursor moves end	Cursor moves on screen
Cursor moves by multiple lines to line N	Falling or rising tone depending on move direction	Every event	"Line N"	Every event	Cursor moves on screen
Selection	Depends on selection (key-board/mouse)	Every event	"Selection on line N"	1.5 seconds after selection is made	Selection visible on screen
Edits on follower's line	Keyboard type	For every character typed	None	None	Cursor moves on screen; edits visible on screen
Edits on follower's line ( <i>Follow mode off</i> )	Keyboard type	For every character typed	"<collaborator> is editing the same line as you"	As long as edits continue on the same line	Cursor moves on screen; edits visible on screen
Edits within 5 lines of follower ( <i>Follow mode off</i> )	Proximity sound	For every character typed	"<collaborator> is editing nearby"	As long as edits continue on the same line	Cursor moves on screen; edits visible on screen
Follow status	Pull and push sound	When follower starts and stops following leader	"You are now following <collaborator>"	Every event	None

```
16 ##comment
17 print('Hello')
18 print('World')
19 print('welcome')
```

BVI follower's cursor (grey) tethered to sighted leader's cursor (orange)

```
16 ##comment
17 print('Hello')
18 print('World')
19 print('welcome')
```

CLICK

```
16 ##comment
17 print('Hello')
18 print('World')
19 print('welcome')
```

CLICK CLICK 1.5s Delay 🔊 "Line 19"

```
16 ##comment
17 print('Hello')
18 print('World')
19 print('welcome')
```

"Selection: print"

# Code Inspection [6]

---

- ◆ Is a highly structured and formal process of review
  - Sometimes called *static testing* in companies
  - Effective to find bugs and spot process improvements
  - Decreasingly popular
- ◆ Things to check
  - Efficiency of algorithm
  - Logics and correctness of code
    - E.g., control flow and data-flow, I/O format, and order, parameters and return value types and order, etc
  - Comments and consistency with code
  - Develop your own checklist

# Main Ideas in Code Inspection [6]

---

## ◆ Main Ideas

- **Pre-meeting:** Authors educate reviewers, and reviewers read/question the code from a testing angle.
- **In-meeting:** Walk through the code (by code author) with reviewers and check against checklists. Answer questions and resolve ambiguity. Each reviewer has a predefined role assigned pre-meeting. (e.g., a focal area to check)
  - Reviewers “execute” the code in their heads
- **Post-meeting:** Authors follow up and rectify the code. The moderator verifies the required corrective actions taken properly.

## ◆ Characteristics

- Formal, synchronous, check code, tool support is secondary, heavy-weighted

## *Code Inspection:*

# In-Class Activity MCR-1

---

- ◆ Form a group of at least 6 (**neither too small nor too large is useful**)
  - Author **A** of the code: 1 student
  - Moderator **M** of the meeting: 1 student leader (non-author)
  - Reviewer **R**: 1 x syntax, 1 x logics, 1 x input/output (and the environment)
- ◆ On an **online shared Word** document, write down the role of each student in the activity
- ◆ Keep the document and the corrected code for later use for our Exercise MCR-2
- ◆ Timeline (in minutes)
  - 00-10: **A** studies the code C (see the next slide); **M** finds a meeting place (**virtually**)
  - 10-20: **A** educates **R** on C; **M** prepares a checklist on C
  - 20-30: each **R** reviews C privately; **M** calls for a meeting at time-point 29
  - 30-45: code walkthrough, Q&A (**All**); **M** jots down notes N in the Word document
  - 45-50: **M** submits N to Canvas; **A** corrects C to resolve all the problems in N
  - 50-51: **M** and **A** together check the corrected C against N
  - 52: **A** submits the corrected C (**as code C1**) to Canvas/Discussion if **M** agrees
    - Make sure each group keeps C1 as we need it for the next in-class activity MCR-2

```

bool isWin(char game[3][3]){
    bool win = false;
    if (game[0][0] == game[0][1] && game[0][1] == game[0][2] && (game[0][0] == 'X' || game[0][0] == 'O')) win = true;
    if (game[1][0] == game[1][1] && game[1][1] == game[1][2] && (game[1][0] == 'X' || game[1][0] == 'O')) win = true;
    if (game[2][0] == game[2][1] && game[2][1] == game[2][2] && (game[2][0] == 'X' || game[2][0] == 'O')) win = true;
    // column
    if (game[0][0] == game[1][0] && game[1][0] == game[2][0] && (game[0][0] == 'X' || game[0][0] == 'O')) win = true;
    if (game[0][1] == game[1][1] && game[1][1] == game[2][1] && (game[0][1] == 'X' || game[0][1] == 'O')) win = true;
    if (game[0][2] == game[1][2] && game[1][2] == game[2][2] && (game[0][2] == 'X' || game[0][2] == 'O')) win = true;
    // diagonal
    if (game[0][0] == game[1][1] && game[1][1] == game[2][2] && (game[0][0] == 'X' || game[0][0] == 'O')) win = true;
    if (game[0][2] == game[1][1] && game[1][1] == game[2][0] && (game[0][2] == 'X' || game[0][2] == 'O')) win = true;
    return win;
}

int main(){
    int i, j;
    char game[3][3] = { ' ' }; // Tic-tac-toe
    char player1 = 'X';
    char player2 = 'O';
    bool turn = true; // false for player 1's turn, true for player 2's turn. Player 1 first.
    cout << "X = Player 1" << endl << "O = Player 2" << endl;
    for (int n=0; n<9; n++){
        turn = !turn; // use the not-operator to change true to false or false to true.
        if (turn == false)
            cout << "Player 1: ";
        else
            cout << "Player 2: ";
        cout << "Which cell to mark? i:[1..3], j:[1..3]: ";
        cin >> i >> j;
        if (turn == false)
            game[i][j] = 'X';
        else
            game[i][j] = 'O';
        if (isWin(game)){
            cout << "Win!" << endl;
            break; // need to terminate the problem
        }
    }
    if (i==3) // all celled with i=0 to 2 have been inputted above but no winner yet
        cout << "Tie!" << endl;

    // show the game to console
    cout << game[0][0] << " " << game[0][1] << " " << game[0][2] << endl;
    cout << game[1][0] << " " << game[1][1] << " " << game[1][2] << endl;
    cout << game[2][0] << " " << game[2][1] << " " << game[2][2] << endl;
    return 0;
}

```

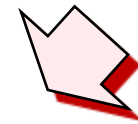
## ***MCR-1:***

# **Materials Authors to Pass to Reviewers**

---

Console screen:

Inputs are underlined



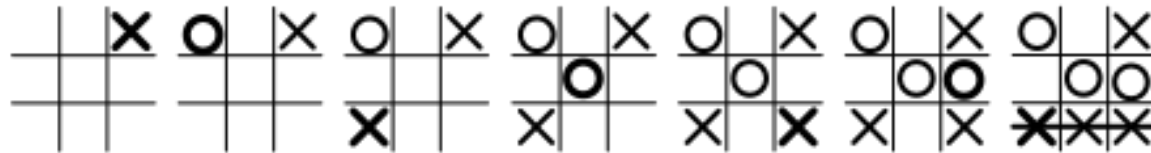
```
X = Player 1
O = Player 2
Player 1: Which cell to mark? i:[0..2], j:[0..2]: 0 2
Player 2: Which cell to mark? i:[0..2], j:[0..2]: 0 0
Player 1: Which cell to mark? i:[0..2], j:[0..2]: 2 0
Player 2: Which cell to mark? i:[0..2], j:[0..2]: 1 1
Player 1: Which cell to mark? i:[0..2], j:[0..2]: 2 2
Player 2: Which cell to mark? i:[0..2], j:[0..2]: 1 2
Player 1: Which cell to mark? i:[0..2], j:[0..2]: 2 1
Win!
O   X
   O O
X X X
```

## *MCR-1:*

# Authors' Private Materials

---

Tic-tac-toe (過三關) is a game involving two users. In the game, there is a 3x3 grid. The grid is initially empty. Each cell can place with the character 'X', the character 'O', or be left empty. When there is a row, a column or a diagonal all filling with the same character, then the user Wins. For instance, the diagram shows that the bottom row is all filled with the character 'X'. So, the user for 'X' wins the game.



The written program is to allow two players (Player 1 and Player 2) to input the coordinates of their chosen cells in turn. After receiving each input, the program determines whether there is any winner, and if there is a winner, it outputs the message “Win!”. If there is no winner after all the 9 cells filled, output “Tie!”. Right after showing the message “Win!” or “Tie!”, it shows<sub>15</sub> the whole tic-tac-toe grid and terminates the game.

# Modern Code Review (MCR)

---

- ◆ MCR is a lightweight and tool-based code review of *code change* (not the whole piece of code!)
  - is the norm for many software development projects
- ◆ Characteristics of MCR
  - Informal; tool-based logistics; asynchronous; and focused on reviewing code changes; lighter weight than code inspection
- ◆ Main Ideas
  - Author **A** makes a patch **P** on a code block **C** to address some problems and sends the patch **P** via e-channel (e.g., email, WhatsApp, WeChat, or GitHub tool) to a set of reviewers **R**
  - Each reviewer **R** evaluates **P**: deems **P** good, requests changes on **P**, or rejects **P**
  - Author **A** commits **P**; post-commit review by other **R**s possible.
- ◆ Authors and reviewers exchange ideas, find bugs, and discuss alternative solutions to better design the structure of a submitted code change.



# MCR Process

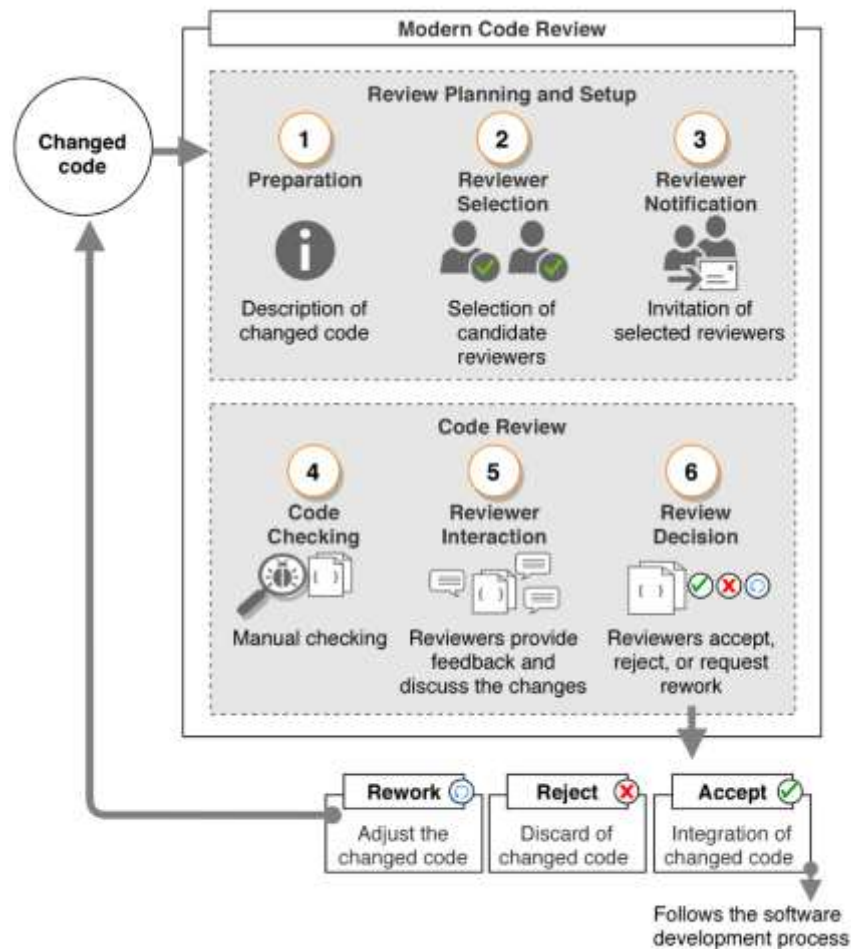


Fig. 1. An overview of the tasks of modern code review. [19]

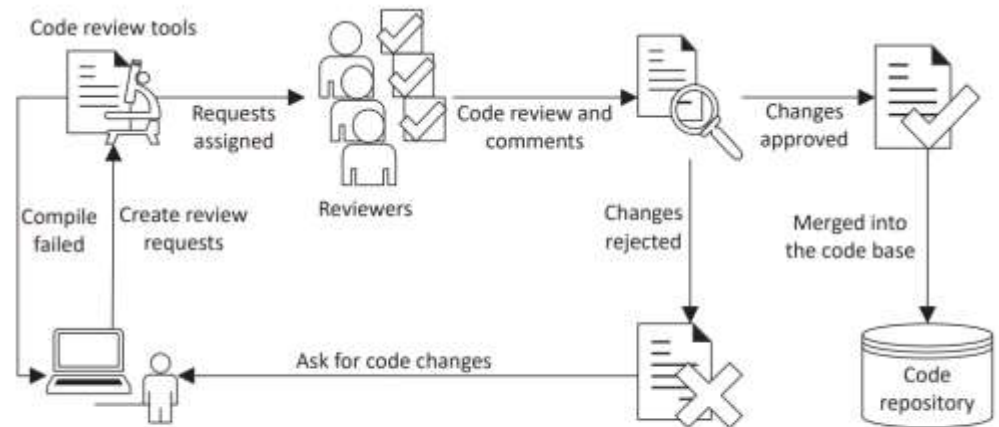


Fig. 1. An overview of the code review process. [20]

# Example [12]



Change 107871 - Merged

1

## Implement EDP for a Spark standalone cluster

This change adds an EDP engine for a Spark standalone cluster. The engine uses the spark-submit script and various linux commands via ssh to run, monitor, and terminate Spark jobs.

Currently, the Spark engine can launch "Java" job types (this is the same type used to submit Oozie Java action on Hadoop clusters)

A directory is created for each Spark job on the master node which contains jar files, the script used to launch the job, the job's stderr and stdout, and a result file containing the exit status of spark-submit. The directory is named after the Sahara job and the job execution id so it is easy to locate. Preserving these files is a big help in debugging jobs.

A few general improvements are included:

- \* engine.cancel\_job() may return updated job status
- \* engine.run\_job() may return job status and fields for job\_execution.extra in addition to job id

Still to do:

- \* create a proper Spark job type (new CR)
- \* make the job dir location on the master node configurable (new CR)
- \* add something to clean up job directories on the master node (new CR)
- \* allows users to pass some general options to spark-submit itself (new CR)

Partial implements: blueprint edp-spark-standalone

Change-Id: I2c84e9cdb75e846754896d7c435e94bc6cc397ff

Author	Alice <alice@redhat.com>
Committer	Alice <alice@redhat.com>
Commit	5698799ee3642a28797c6022dd35f228616764e1
Parent(s)	e23efe5471ed3e3ef3356918f80d91838f1c6585
Change-id	I2c84e9cdb75e846754896d7c435e94bc6cc397ff

2

## History

Alice Uploaded patch set 1

## Bob

Patch Set 1:

sahara/service/edp/job\_manager.py

Line 68:

should this be guarded with:  
if job\_info.get('status') in job\_utils.terminated\_job\_states:  
just in case 'status' doesn't exist?

## Alice

Patch Set 4:

The patch LGTM, apart from the small comment on the commit message. One important question, though, is about the data sources. How is input and output specified for each job submitted through Spark EDP? Spark does not support Swift for now, so I would expect only HDFS to be available.

Owner	Trevor McKay			
	Bob	Alice	John	Rob
Reviewers	Edward	Sam	Ryan	Alex
	Enzo	Frank		
Project	5698799ee3642a28797c6022dd35f228616764e1			
Branch	e23efe5471ed3e3ef3356918f80d91838f1c6585			

3

Files	Comments
sahara/plugins/spark/plugin.py	33
sahara/service/edp/job_utils.py	46
sahara/service/edp/oozie/engine.py	46
sahara/service/edp/job_utils.py	18
sahara/service/edp/oozie/oozie.py	7
sahara/service/edp/resources/launch_command.py	66
sahara/service/edp/spark/engine.py	161
sahara/tests/unit/service/edp/spark/_init_.py	0
sahara/tests/unit/service/edp/spark/test_spark.py	383
sahara/tests/unit/service/edp/test_job_manager.py	10

4

See next slide

5

# Example [12]

## Expanding a file in the change

```
class FormPostTest(ObjectStorageFixture):

    @classmethod
    def setUpClass(cls):
        super(FormPostTest, cls).setUpClass()
        cls.key_cache_time = (
            cls.objectstorage_api_config.tempurl_key_cache_time)
        cls.tempurl_key = cls.behaviors.VALID_TEMPURL_KEY
        cls.object_name = cls.behaviors.VALID_OBJECT_NAME
        cls.object_data = cls.behaviors.VALID_OBJECT_DATA
        cls.content_length = str(len(cls.behaviors.VALID_OBJECT_DATA))
        cls.http_client = HTTPClient()
        cls.redirect_url = "http://example.com/form_post_test"
```

```
class FormPostTest(ObjectStorageFixture):

    @classmethod
    def setUpClass(cls):
        super(FormPostTest, cls).setUpClass()
        cls.key_cache_time = (
            cls.objectstorage_api_config.tempurl_key_cache_time)

        cls.object_name = cls.behaviors.VALID_OBJECT_NAME
        cls.object_data = cls.behaviors.VALID_OBJECT_DATA
        cls.content_length = str(len(cls.behaviors.VALID_OBJECT_DATA))
        cls.http_client = HTTPClient()
        cls.redirect_url = "http://example.com/form_post_test"
```

**Alice**

Apr 22, 2015

Should there be a default value for the `cls.tempurl_key`, or should the fixture assert if `keys_set` is empty? All of the tests depend on the attribute, but for whatever reason, if the `X-Account-Meta-Temp-Url-Key` is not present in the headers, the attribute will not exist, and the tests will error out in an ungraceful manner.

[Reply](#) [Quote](#) [Done](#)

**Bob**

Apr 22, 2015

So the `check_account_tempurl_keys` method will first check to see if the account keys are set and if they aren't, it will set them to some defaults. If, for some reason, it fails to set them properly, `keys_set` should be `False` instead of `True`. So, what I will do is have an `else` statement here which will raise an `Exception` but in all likelihood it would fail in the behaviors beforehand.

[Reply](#) [Quote](#) [Done](#)

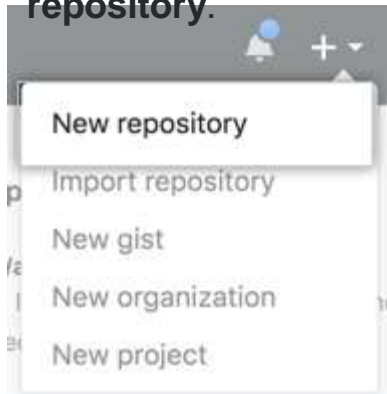
```
40 @ObjectStorageFixture.required_features('formpost')
41 def test_object_formpost_redirect(self):
42     """
```

```
44 @ObjectStorageFixture.required_features('formpost')
45 def test_object_formpost_redirect(self):
46     """
```

```
47
```

# Example [17] GitHub

1. In the upper-right corner of any page, use the drop-down menu, and select **New repository**.



2. Type a short, memorable name for your repository. For example, "hello-world".



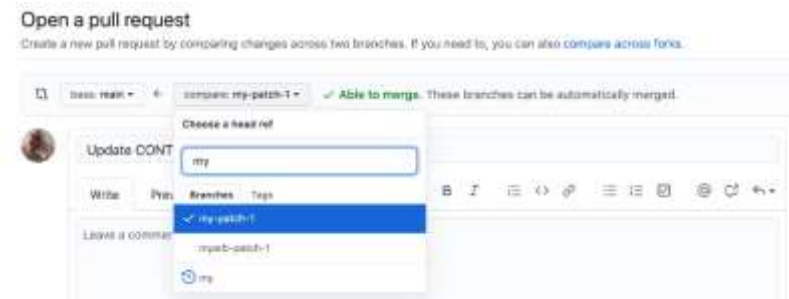
3. make any changes you need to the file.



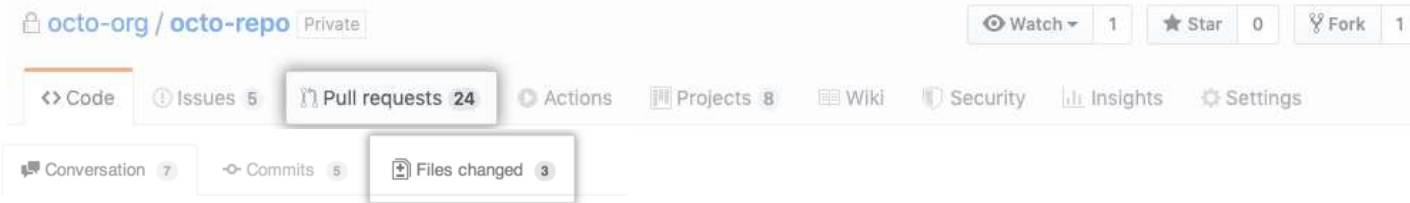
4. decide whether to add your commit to the current branch or to a new branch.



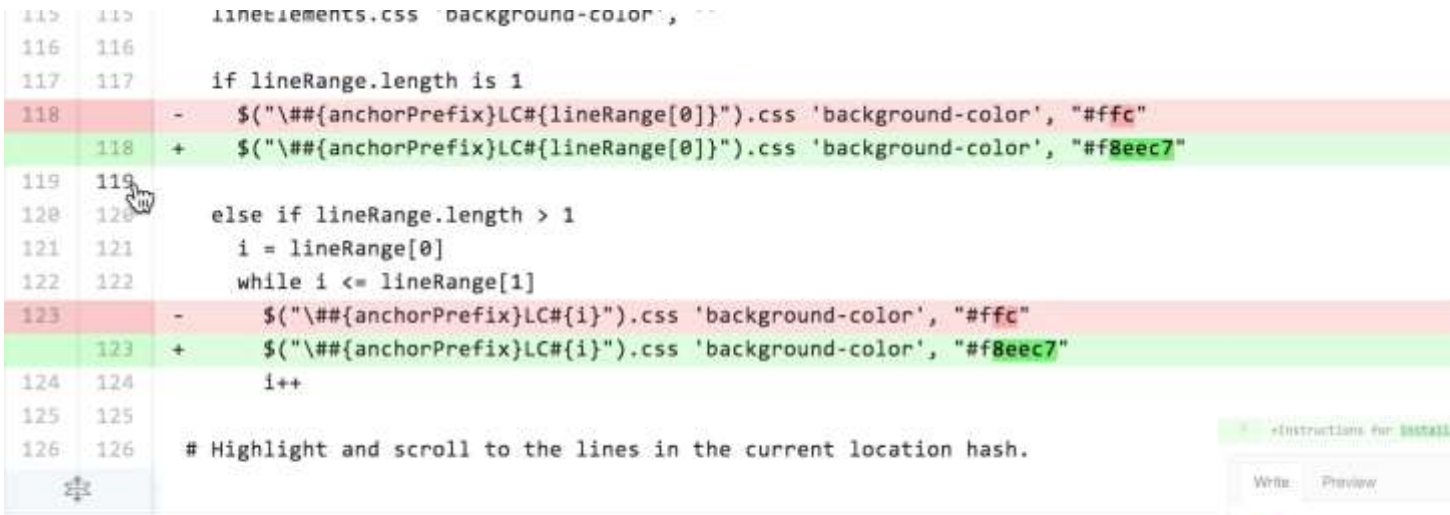
5. Create a pull request to propose and collaborate on changes to a repository.



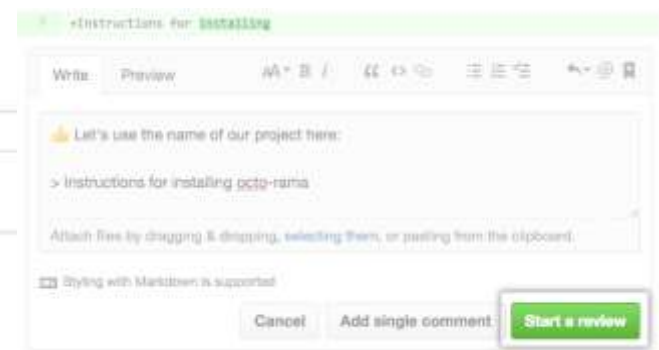
# Example [17] GitHub



6. Hover over the line of code where you'd like to add a comment and click the blue comment icon. To add a comment on multiple lines, click and drag to select the range of lines, then click the blue comment icon.



7. When you're done, click Start a review. If you have already **started a review**, you can click Add review comment.





# Characteristics of Modern Code Review

---

- ◆ Informal
  - Ad hoc group of reviewers without predefined roles, and the review process does not follow a formal procedure
- ◆ Tool-based logistics
  - Email, WhatsApp, and WeChat are popular.
  - Open-source tools: GitHub, Gerrit (for Git), ReviewBoard, Phabricator
- ◆ Asynchronous
  - Authors and reviewers need not engage in the same task simultaneously (but not extensively delayed)
- ◆ Focus the **review** on the **patch** (code change, the delta)

# Purposes of Reviews/Comments/Posts [1]

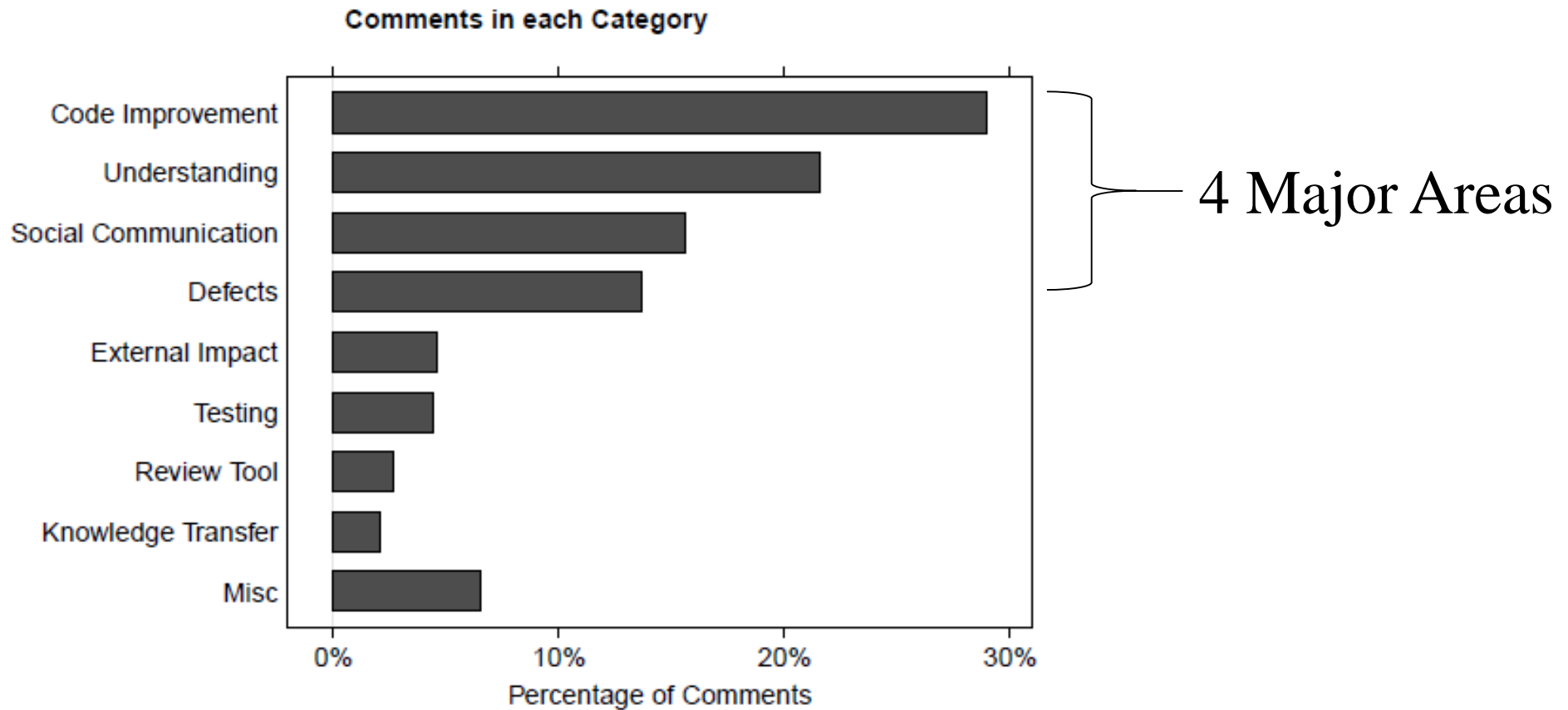


Fig. 4. Proportion of comments by card sort category.

# Understanding the Code for What? [1]

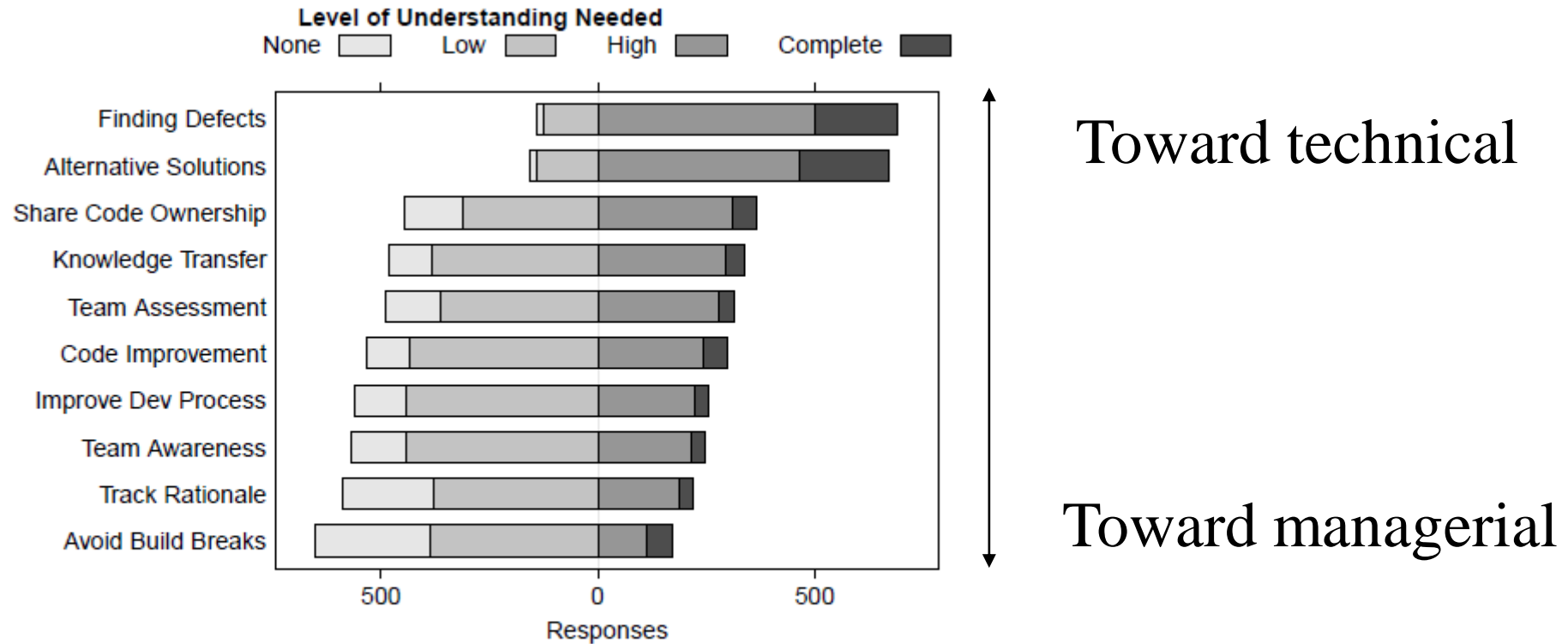


Fig. 5. Developers responses in surveys of the amount of code understanding for code review outcomes.



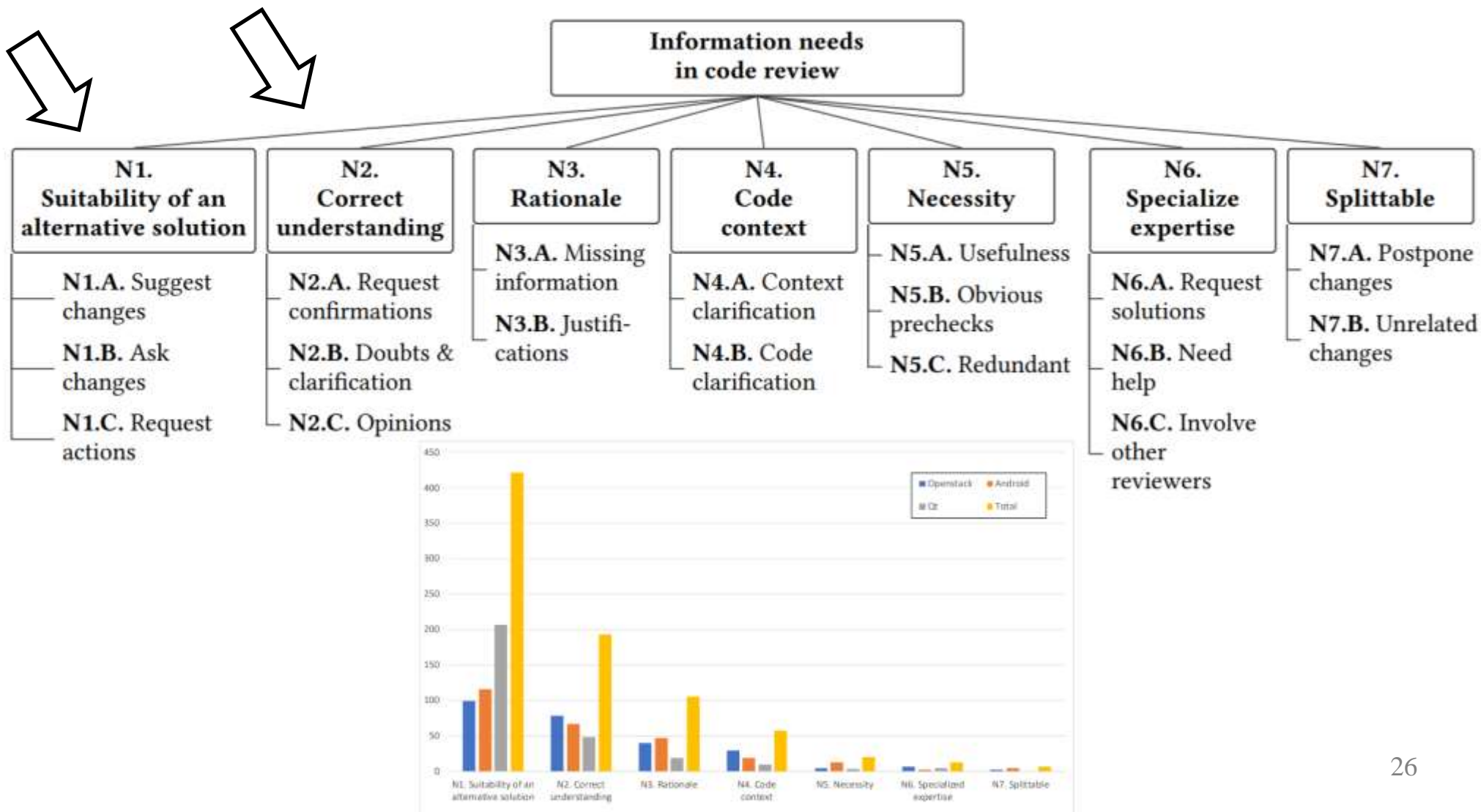
# Strategy to Deal with Confusion [14]

THE REASONS, IMPACTS AND COPING STRATEGIES DEVELOPERS USE TO DEAL WITH CONFUSION; IN THE PARENTHESIS ARE THE NUMBERS OF CARDS.

	<b>Review process</b> 18 topics (120)	<b>Artifact</b> 15 topics (300)	<b>Developer</b> 15 topics (124)	<b>Link</b> 9 topics (177)
<b>Reasons</b> 30 topics (507)	<b>Organization of work (17)</b> Issue tracker, version control (7) Unnecessary change (6) Not enough time (3) Dependency between changes (3) Code ownership (2) Community norms (2)	<b>Missing rationale (66)</b> Discussion of the solution: non-functional (49) Unsure about system behavior (37) Lack of documentation (29) Discussion of the solution: strategy (29) Long, complex change (25) Lack of context (19) Discussion of the solution: correctness (14) Impact of change (11) Irreproducible bug (6) Lack of tests (5)	<b>Disagreement (18)</b> Communicative intention (9) Language issues (3) Propagation of confusion (3) Fatigue (1) Noisy work environment (1)	<b>Lack of familiarity with existing code (47)</b> <b>Lack of programming skills (40)</b> Lack of understanding of the problem (21) Lack of understanding of the change (17) Lack of familiarity with the technology (14) Lack of knowledge about the process (3)
<b>Impacts</b> 14 topics (98)	Delaying (31) Decreased review quality (11) Additional discussions (11) Blind approval (8) Review rejection (4) Increased development effort (4) Assignment to other reviewers (2)	Better solution (1) Incorrect solution (1)	Decreased confidence (10) Abandonment (6) Frustration (5) Anger (2) Propagation of confusion (2)	
<b>Coping strategies</b> 13 topics (116)	Improved organization of work (5) Delaying (2) Assignment to other reviewers (1) Blind approval (1)	Small, clear changes (4) Improved documentation (4)	<b>Information requests (36)</b> Off-line discussions (12) Providing/accepting suggestions (10) Disagreement resolution (6)	<b>Improved familiarity with existing code (28)</b> Testing the change (5) Improved familiarity with the technology (2)

No particularly good strategy

# What Information is Needed by Reviewers? [12]



# How about a review patch is for code refactoring?

- ◆ Industrial case study in Xerox [23]
- ◆ “Coping with poor design and coding style is the main driver for developers to apply refactoring in their code changes. Yet, functional changes and bug fixing activities often trigger developers to refactor their code as well.”

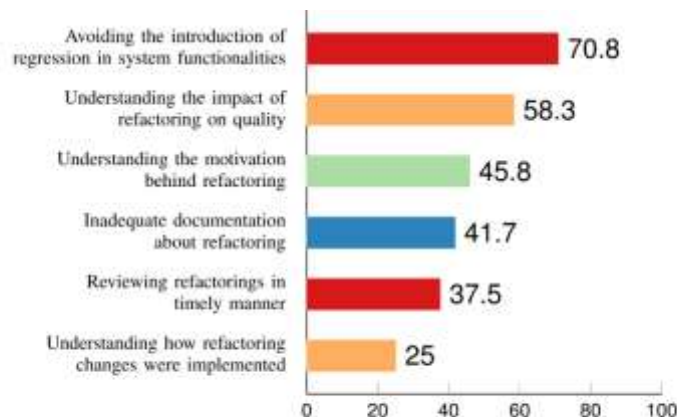


Figure (4) Challenges faced by developers when reviewing refactoring.

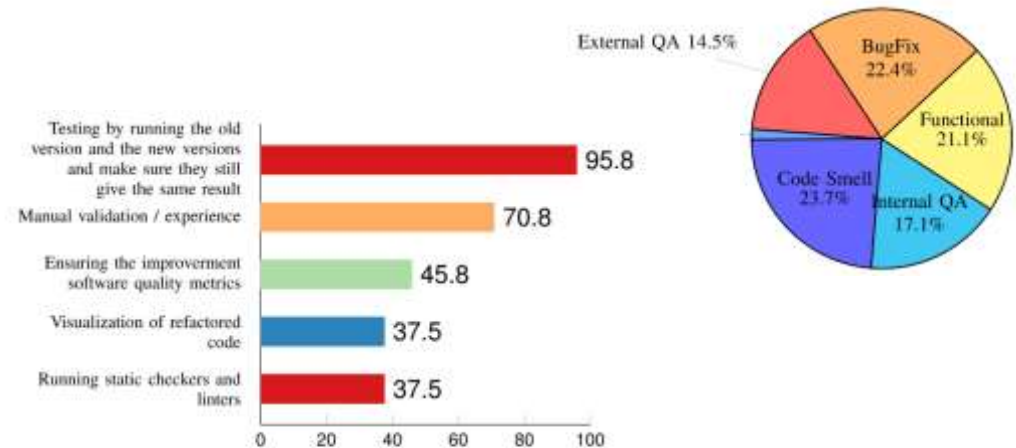


Figure (5) Mechanisms used to ensure the correctness after the application of refactoring.

# How about Knowledge Transfer?

---

- ◆ Does conducting more rounds of MCR improve the code quality of a developer?
  - Unfortunately, there is no evident yet both within the same project and across multiple projects. [16]
- ◆ Then, let's look at the MCR in big tech companies. See the next few slides.

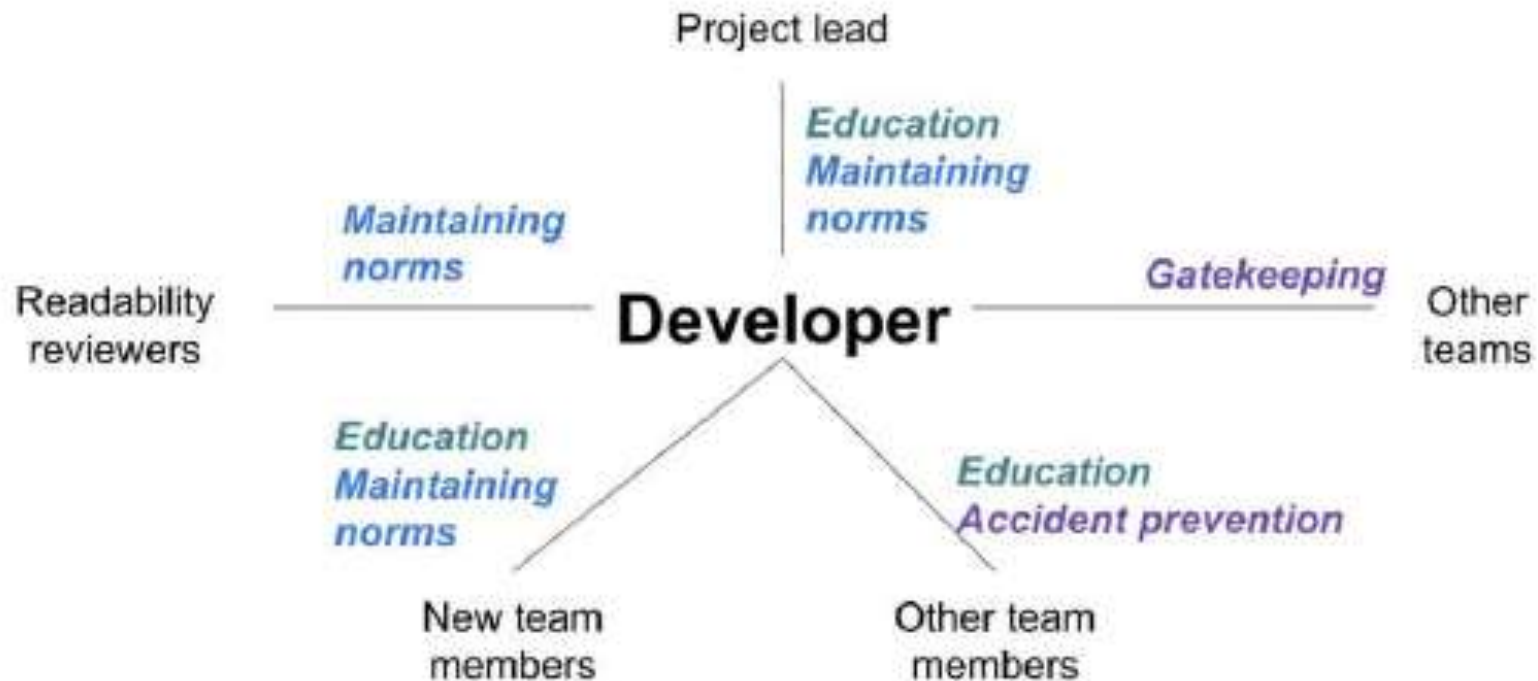
# MCR Best Practices [1, 3, 7]

---

- ◆ peer review follows a lightweight, flexible process
- ◆ Reviews happen early (before a change is committed), in time, and frequently
- ◆ Reviewers have a prior knowledge of the context and the code to complete reviews quicker and provide more valuable feedback to the authors
- ◆ Change sizes are small
- ◆ Small numbers of reviewers (1 to 5 reviewers)
- ◆ Read informal posts and comments on the code and patch
- ◆ Review is not just to find bugs at group-level: problem solving (solution development), code readability and maintainability, following the norm (of the belonging company), accident prevention, gate-keeping (recall the role of moderator in code inspection?)

# MCR Best Practices [3]

---



- ◆ The above diagram shows the relationships between Developer (i.e., Author) and Reviewers in a code project in Google

# MCR Best Practices in Google [3]

---

- ◆ Suggesting Reviewers: ask persons who can serve as reviewers in turn and in a round-robin manner.
  - Do not consistently overload the best persons in the team
- ◆ Perform code analysis to supplement human reviewers
  - Apply style checkers, code checkers
  - E.g., Checkstyle for Java
- ◆ Review frequency: 4 hours of review (and 4 reviews) per week (median)
- ◆ Review speed: authors wait for 1-5 hours
- ◆ Changed code size: 1 line of code (~10%), one file (~30%), fewer than 10 files (~90%).

# MCR Best Practices

---

## ◆ Reviewer candidates?

- Code owner
- Developers who made previous changes on the code
- Developers who develop the code before (e.g., in producing the code through pair programming)
- Experienced reviewers
- Developers who develop similar features of the changed fragments in some other code
- ...



# MCR Issues

---

- ◆ Review Quality and Code Size
  - **Reviews on code with higher complexity get fewer discussions and feedback.**
  - **Code with fewer review feedbacks will encounter more post-release bugs**
- ◆ Distance
  - Causing delay in the review process or leading to misunderstanding
  - Aspect 1: Geographical issue
    - Large physical distance between authors and reviewers
  - Aspect 2: Organization issue
    - Reviewers from different teams or taking different roles
- ◆ Social Interaction
  - Tone: comments with negative tones are less useful
  - Power: ask another person to change their comments. Authors unhappy
- ◆ Context
  - Misunderstanding due to mismatched expectations on the code change (e.g., nice to have vs. urgent fix, full solution vs. high level sketch)

# Other Issues in Ineffective MCR: Anti-Pattern in MCR [18]

TABLE I: MCR anti-patterns and their associated symptoms and consequences.

MCR anti-pattern	Symptoms	Potential Consequences
Confused reviews (CR)	<ul style="list-style-type: none"> <li>Reviewers ask question(s) about the rationale or the solution approach of the patch.</li> <li>Reviewers express incertitude in review comments.</li> </ul>	<ul style="list-style-type: none"> <li><i>Process</i>: Confusion decreases the efficiency and the effectiveness of the review [6], [10].</li> <li><i>Artifact</i>: The patch may still have poor quality after the review as reviewers may not fully understand the patch [6].</li> <li><i>People</i>: Reviewers may feel frustrated and may express negative sentiments due to the confusion [6].</li> </ul>
Divergent reviews (DR)	<ul style="list-style-type: none"> <li>The review decisions do not reach consensus.</li> <li>The review scores are diverged.</li> <li>Reviewers post conflict review comments to each others.</li> </ul>	<ul style="list-style-type: none"> <li><i>Process</i>: The divergence can slow down the integration process [16]</li> <li><i>Artifact</i>: It is correlated with negative development outcomes (i.e., patches without changing) [16]</li> <li><i>People</i>: The divergence increases contributors' likelihood of leaving the communities and lead to the poor team performance [11], [29]</li> </ul>
Low review participation (LRP)	<ul style="list-style-type: none"> <li>The patch does not have other developers as a reviewer.</li> <li>The patch receive few/short comments.</li> <li>The patch does not receive prompt/timely feedback from reviewers.</li> </ul>	<ul style="list-style-type: none"> <li><i>Process</i>: The lack of review participation has a negative impact on review efficiency and effectiveness [30].</li> <li><i>Artifact</i>: Patches with low number of reviewers can be more defect-prone [7]</li> <li><i>People</i>: The inefficient reviewer feedback could make the patch author forget the change. [5]</li> </ul>

TABLE II: The detection results of MCR anti-patterns.







Anti-pattern	# instances	Kappa	Agreement
Confused Reviews (CR)	21 	0.93	Perfect
Divergent Reviews (DR)	20 	1.0	Perfect
Low Review Participation (LRP)	32 	1.0	Perfect

TABLE III: The prevalence of MCR anti-patterns.

Category	Count
Code reviews affected by one anti-pattern	67 
Code reviews affected by two anti-patterns	21 
Code reviews affected by three anti-patterns	4 

# Should We Review Test Code too?

- ◆ The simple answer is “yes”, but on different focuses and **less often** (and review with the production code)

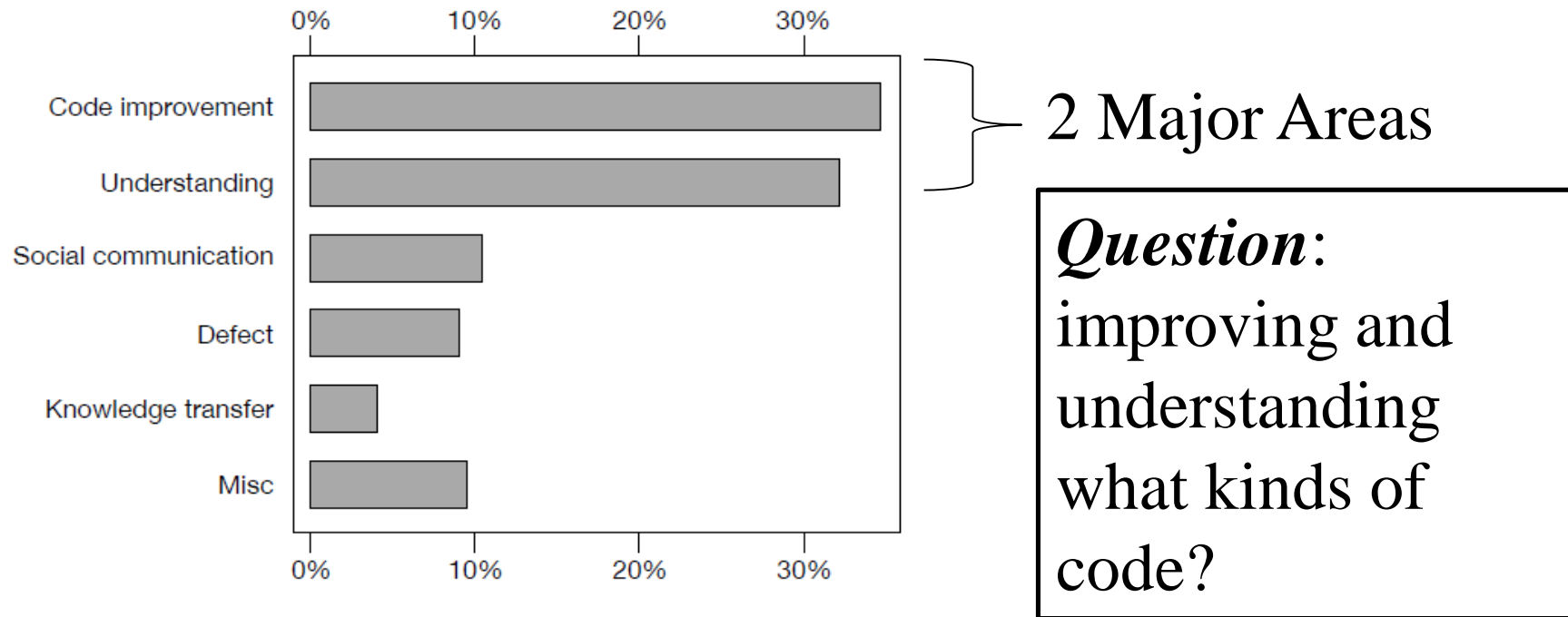


Figure 1: The outcomes of comments in code review of test files, after manual analysis in 600 comments (CL=99%, CI=5). [4]

# Understanding and Improving What Kinds of Code by Reviewing Test Code? [15]

Conducted by German researchers from 92 developers.

INTERVIEWEES' EXPERIENCE (IN YEARS) AND WORKING CONTEXT

ID	Developer	Reviewer	Working context	Applying TDR
P1	8	8	OSS	Almost never
P2	3	3	Company A	Almost Always
P3	15	15	Company A	Almost Always
P4	10	10	Company B	Almost Never
P5	10	5	Company C	Always
P6	3	2	Company D	Sometimes
P7	16	16	Company E	Always
P8	4	4	Company F / OSS	Sometimes
P9	3	3	Company G / OSS	Never

**Finding 1.** In reviews in which tests are presented first, participants found significantly more bugs in test code, yet fewer maintainability issues in production code. The production bugs and test maintainability issues found is stable across the treatments.

**Finding 3.** Perceived problems with TDR: Developers report to (1) consider tests as less important than production, (2) not being able to extract enough knowledge from tests, (3) not being able to start a review from tests of poor quality, and (4) being comfortably used to read the patch as presented by their code review tool.

**Finding 4.** Perceived advantages of TDR: Developers report that TDR (1) allows them to have a concise, high-level overview of the code under test and (2) helps them in being more testing-oriented, hence improving the overall test code quality.

# *Modern Code Review:*

## **In-Class Activity MCR-2**

---

- ◆ Form a group of students
  - Author **A** of the code: 1 student
  - Reviewer **R**: no pre-defined role
  - Agree among members on the tool for review (we use Github in the tutorial)
- ◆ Timeline (in minutes)
  - 00-15: **A** prepares the patch P for the code C on the Github platform, along with some explanation of the patch P, and notifies the groupmate via the e-channel
  - 15-25: **R** reviews P and posts comments on GitHub. Also, notify via the e-channel when finishing the review
  - 15-25: **A** answers issues and provides clarifications made by R and revises patch P' iteratively.
  - 25-30: **R** approves P, and **A** applies the final patch P' get the final corrected code C2
  - 30: **A** **submits the corrected C2 with comments (images or link to the GitHub repository?) to Canvas Discussion for peer sharing** in addition to using C2 as a commit

# Self-Study Home Exercise

## [no submission is needed]

---

### ◆ Read Ref [1]

- Find out the methods to increase team awareness
- On page 5, a summary from a senior developer stated five benefits of code review. What are the benefits?
- On pages 5-6, what can a team do to increase knowledge transfer?
- On page 7, what are the stated problems if the quality of review code is slow?
- On page 9, what are the recommendations? Do you agree?

### ◆ Read Ref [5]

- On slide 23, what are the key features of a tool to support code reviews that the author of the Slide Presentation believes the tool to have?

### ◆ Self-Reflection

- If your company asks you to incorporate MCR into your project but without providing additional budget (\$), what and how you implement MCR for it?
- Distance is a major MCR issue. How does a team (with members located in different cities and probably with different culture backgrounds) do MCR?



# References and Resources

---

1. Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 712-721.
2. Christopher Thompson and David Wagner. 2017. A Large-Scale Study of Modern Code Review and Security in Open Source Projects. In *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE)*. ACM, New York, NY, USA, 83-92. DOI: <https://doi.org/10.1145/3127005.3127014>
- Code review appears not to be a good choice for identifying security bugs
3. Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '18)*. ACM, New York, NY, USA, 181-190. DOI: <https://doi.org/10.1145/3183519.3183525>
4. Davide Spadini, Maurício Aniche, Margaret-Anne Storey, Magiel Bruntink, and Alberto Bacchelli. 2018. *When testing meets code review: why and how developers review tests*. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, New York, NY, USA, 677-687. DOI: <https://doi.org/10.1145/3180155.3180192>
5. SlideShare: <https://www.slideshare.net/excellaco/modern-code-review>

# References and Resources

---

6. M. E. Fagan. 1976. Design and code inspections to reduce errors in program development. IBM System Journal, 15, 3 (September 1976), 182-211. DOI=<http://dx.doi.org/10.1147/sj.153.0182>
7. Peter C. Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013). ACM, New York, NY, USA, 202-212. DOI: <https://doi.org/10.1145/2491411.2491444>
8. Wikipedia. List of tools for code review. [https://en.wikipedia.org/wiki/List\\_of\\_tools\\_for\\_code\\_review](https://en.wikipedia.org/wiki/List_of_tools_for_code_review) . 2018
9. Gerrit: <https://www.gerritcodereview.com/> Last access 2021
10. ReviewBoard: <https://www.reviewboard.org/> Last access 2021
11. Phabricator <https://www.phacility.com/> Last access 2021
12. Luca Pascarella, Davide Spadini, Fabio Palomba, Magiel Bruntink, and Alberto Bacchelli. 2018. Information Needs in Contemporary Code Review. Proc. ACM Hum.-Comput. Interact. 2, CSCW, Article 135 (November 2018), 27 pages. DOI: <https://doi.org/10.1145/3274404>
13. Achyudh Ram, Anand Ashok Sawant, Marco Castelluccio, and Alberto Bacchelli. 2018. What makes a code change easier to review: an empirical investigation on code change reviewability. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018). 201-212. DOI: <https://doi.org/10.1145/3236024.3236080>
14. Felipe Ebert, Fernando Castor, Nicole Novielli, Alexander Serebrenik: Confusion in Code Reviews: Reasons, Impacts, and Coping Strategies. In Proceedings of 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019. 49-60. <https://doi.org/10.1109/SANER.2019.8668024>



# References and Resources

---

15. Davide Spadini, Fabio Palomba, Tobias Baum, Stefan Hanenberg, Magiel Bruntink, and Alberto Bacchelli. 2019. Test-driven code review: an empirical study. In Proceedings of the 41st International Conference on Software Engineering (ICSE '19). IEEE Press, 1061–1072. DOI:<https://doi.org/10.1109/ICSE.2019.00110>
16. Maria Caulo, Bin Lin, Gabriele Bavota, Giuseppe Scanniello, Michele Lanza. Knowledge Transfer in Modern Code Review. In Proceedings of 28th IEEE/ACM International Conference on Program Comprehension (ICPC 2020). <https://binlin.info/downloads/Caulo2020a.pdf>
17. Github Docs: <https://docs.github.com/en/github> Last access 2021
18. Moataz Chouchen, Ali Ouni, Raula Gaikovina Kula, Dong Wang, Patanamon Thongtanunam, Mohamed Wiem Mkaouer, Kenichi Matsumoto. 2021. Anti-patterns in Modern Code Review: Symptoms and Prevalence. SANDER 2021, pp 531-535, 2021.
19. Nicole Davila, Ingrid Nunes. A systematic literature review and taxonomy of modern code review. Journal of Systems and Software 177 (2021): article 110951 . <https://doi.org/10.1016/j.jss.2021.110951>
20. Ruiyin Li, Mohamed Soliman, Peng Liang, Paris Avgeriou. Symptoms of Architecture Erosion in Code Reviews: A Study of Two OpenStack Projects. 2022 IEEE 19th International Conference on Software Architecture (ICSA 2022), pp 24-35, 2022.
21. Glenford J. Myers. A Controlled Experiment in Program Testing and Code Walkthroughs Inspections. CACM, 1978 21(9): 760-768, 1979.
22. Venkatesh Potluri, Maulishree Pandey, Andrew Begel, Michael Barnett, Scott Reitherman: CodeWalk: Facilitating Shared Awareness in Mixed-Ability Collaborative Software Development. ASSETS 2022: 20:1-20:16

# References and Resources

---

23. Eman Abdullah AlOmar, Hussein Alrubaye, Mohamed Wiem Mkaouer, Ali Ouni, Marouane Kessentini: Refactoring Practices in the Context of Modern Code Review: An Industrial Case Study at Xerox. ICSE (SEIP) 2021: 348-357