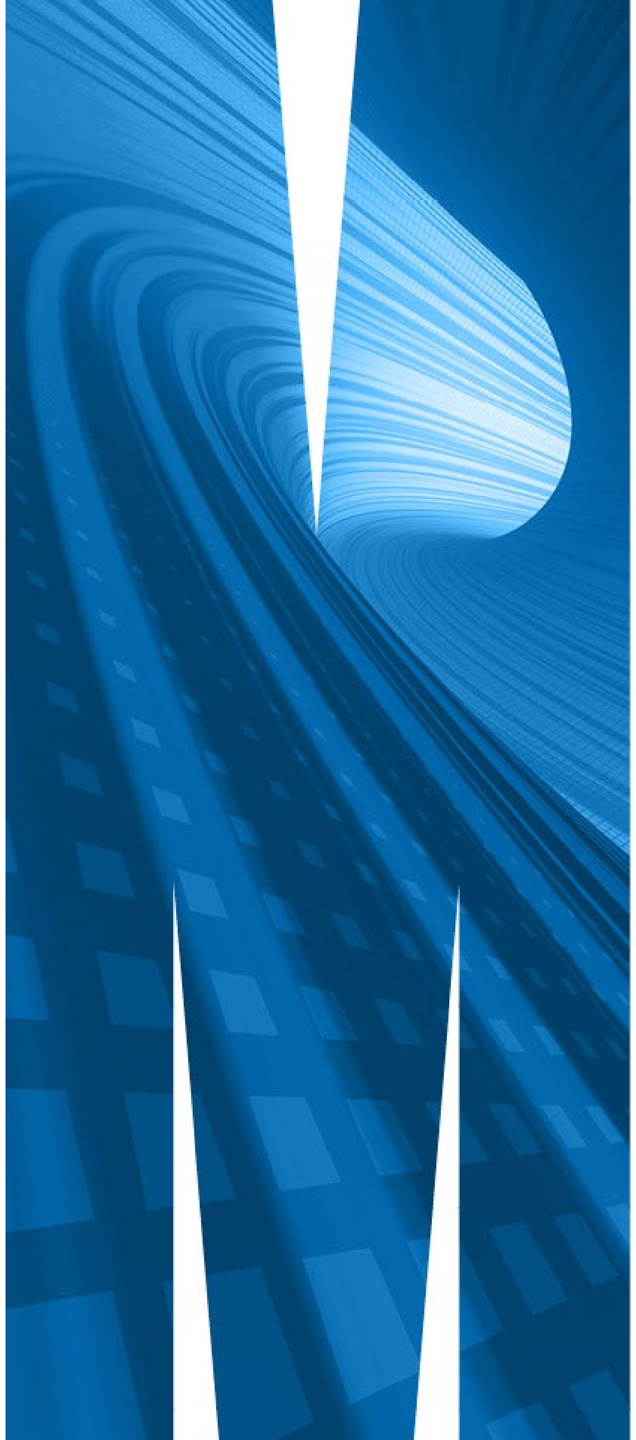
FIT2107-Software Quality & Testing

Lecture 10 – Code Reviews

20th October 2020

Dr Najam Nazar



Announcement

- SETU
- Assignment 3 is released
- It has 4 parts
    - Part A is a checklist preparation
    - Part B is a Group Review
    - Part C is a review by a reviewer
    - Part D is a reflection report
- It has different submission deadlines

Outline

- Part A: Reviews
  - What is Review?
  - Why Review
  - Different Types of reviews
- Part B: Code Reviews
  - Code Reviews and their purpose
  - Different Roles in Code Reviews
  - Code Review Quality Metric
  - Getting the Best Code Review
  - Code Review Checklists
- Part C: Modern Code Reviews (MCR)
  - What is MCR and Process
- Part D: Good Code Reviews
  - Guideline for Authors
  - Guideline for Reviewers
- Part E: Programming Practices
- Part F: Code Review Tools

PART A:
REVIEWS

Reviews

- Reviews: activities to evaluate software artefact through manual scrutiny
    - Inspections: Formally defined
    - Walkthroughs: Author defined.
- Sometimes called static testing.
- Two Types
    - Walkthroughs
    - Inspections
- Applied on Artefacts
    - Source Code
    - Class Diagrams
    - Requirements

Why Reviews?

feature
defect detection

What Do We Know about Defect Detection Methods?

[illegible]

- Main objective
  - Detect faults
- Other objectives
  - Inform
  - Educate
  - Learn from (others) mistakes -> Improve!
- (Undetected) faults may affect software quality negatively - even during the development process!
- Inspections find 25-50% of an artefact’s defects
- Testing finds 30-60% of defects in the code
- Used collectively

Review Method: Walkthroughs

- Author guides through artifact
- Attendants scrutinize and question
- Possibly “simulation”

- Objective
  - Detect faults
  - Become familiar with the product
- Roles
  - Presenter (author)
  - Reviewers

- Elements
  - Planned meeting
  - Team (2 to 7 people)
  - Brainstorm
- Disadvantage
  - Find fewer faults than (formal) inspections

Review Method: Inspections

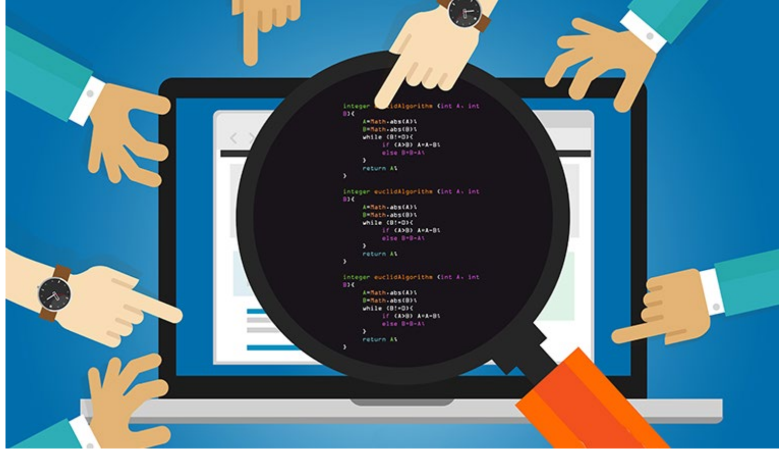
- A step by step peer review of the product
- Each step checked against predetermined criteria (Checklists)

- Objective
  - Detect faults
  - Collect Data
  - Communicate Info.
- Roles
  - Moderators
  - Reviewers
  - Presenters
  - Authors

- Elements
  - Planned Structural meeting
  - Preparation Important
  - Team (3 to 6 people)
- Disadvantage
  - Short Term Cost

PART B:
CODE REVIEWS

Code Reviews



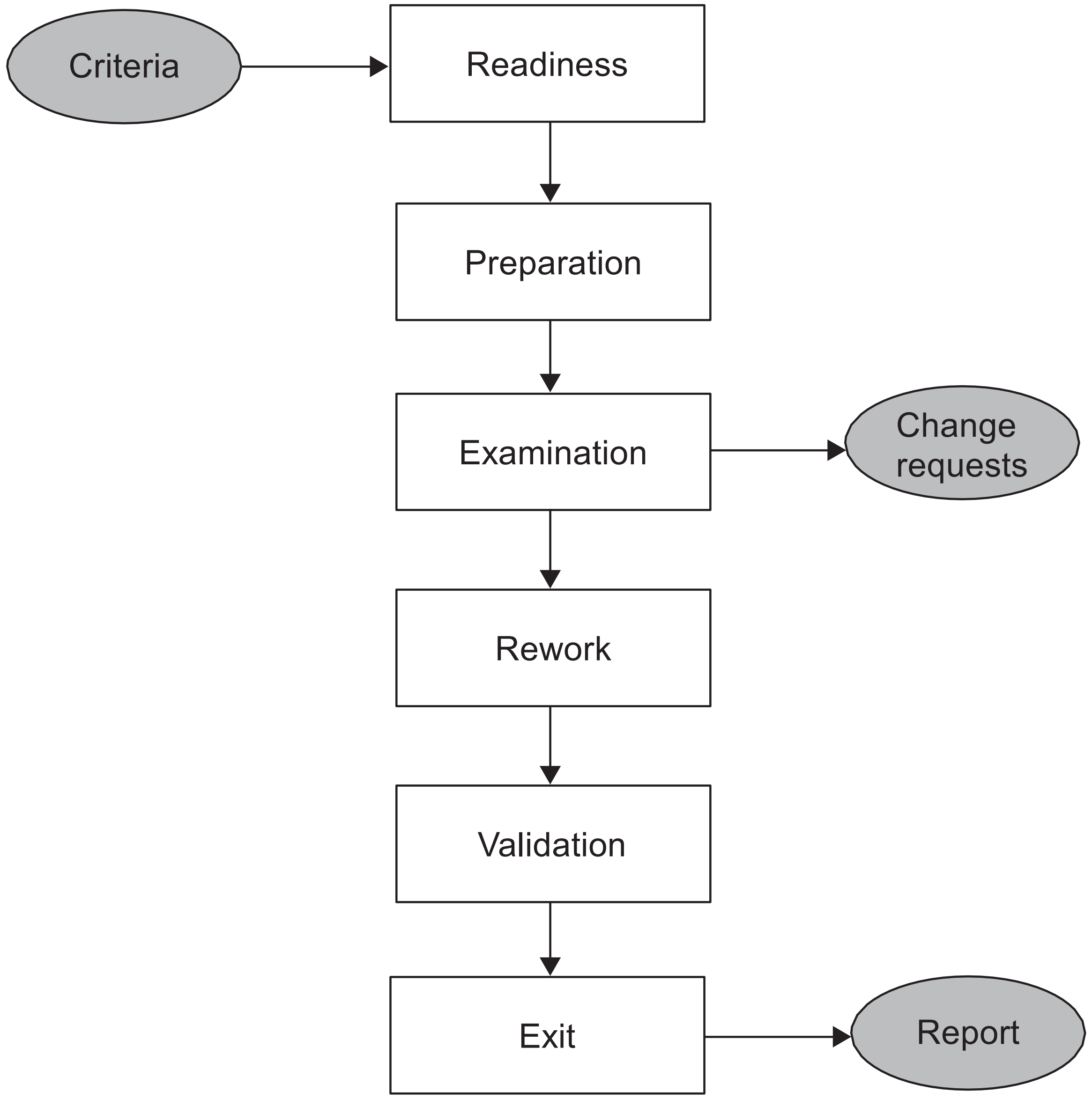
- Software quality assurance activity in which one or several people check a program mainly by viewing and reading parts of its source code.
- Combination of goals:
  - Code Quality.
  - Finding Defects.
  - Knowledge Transfer.
  - Sense of Mutual Responsibility.
  - Finding Better Solutions.
  - Complying to QA Guidelines.

Roles in Code Reviews

- Normally following roles:
    - Moderator: guides the pace of the review process, selects the reviewers and schedules the review meetings, Objectivity.
    - Author(s): has written the code to be reviewed.
    - Presenter: other than the author of the code.
    - Record Keeper: documents the problems found during the review process, automated.
    - Reviewer(s): experts in the subject area of the code under review.
    - Observer: Observe
- Roles may vary depending on the nature of the software and the process used.

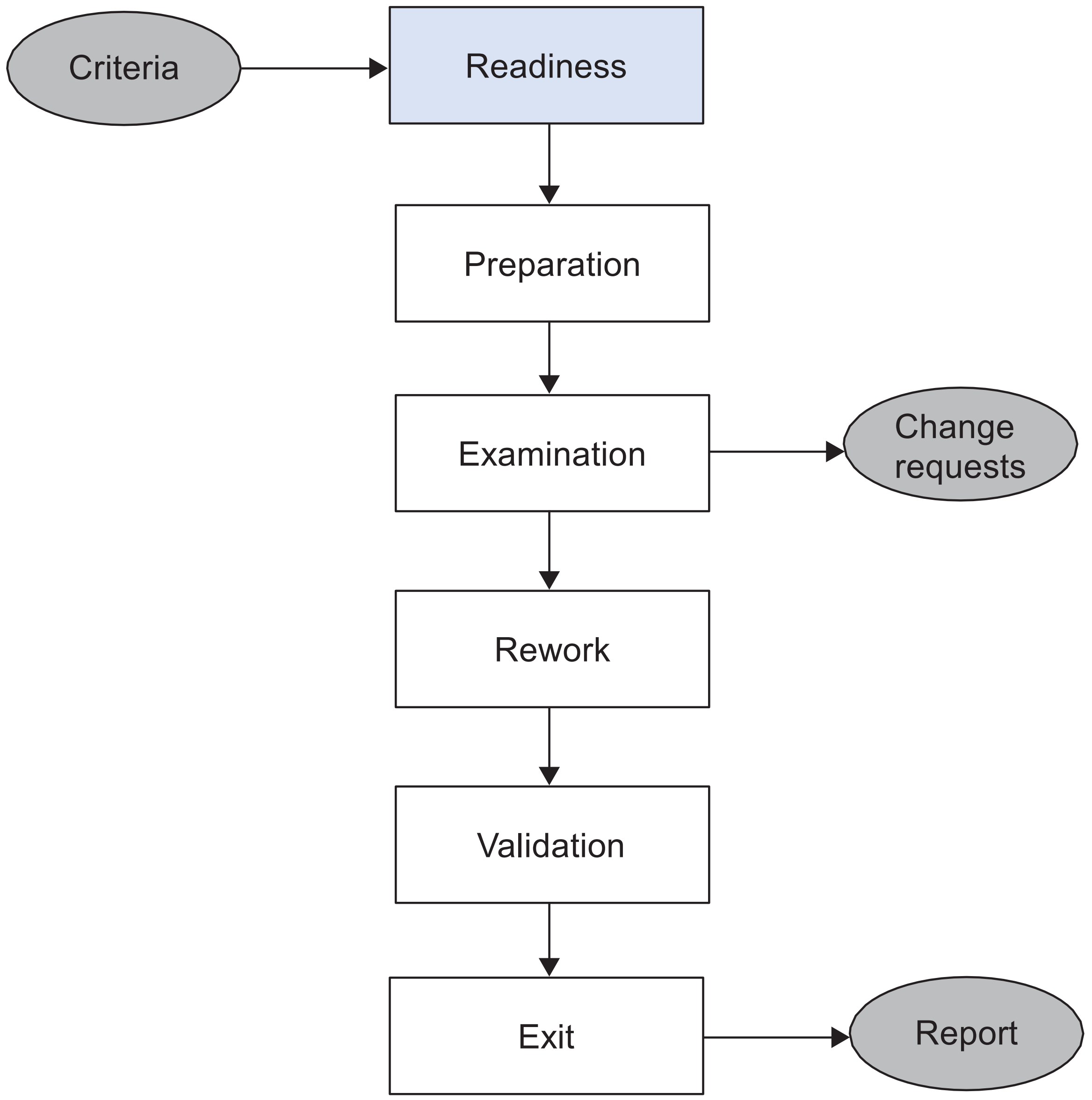
Code Review Process

- Main steps are
  - Readiness
  - Preparation
  - Examination
  - Rework
  - Validation
  - Exit



Code Review Process

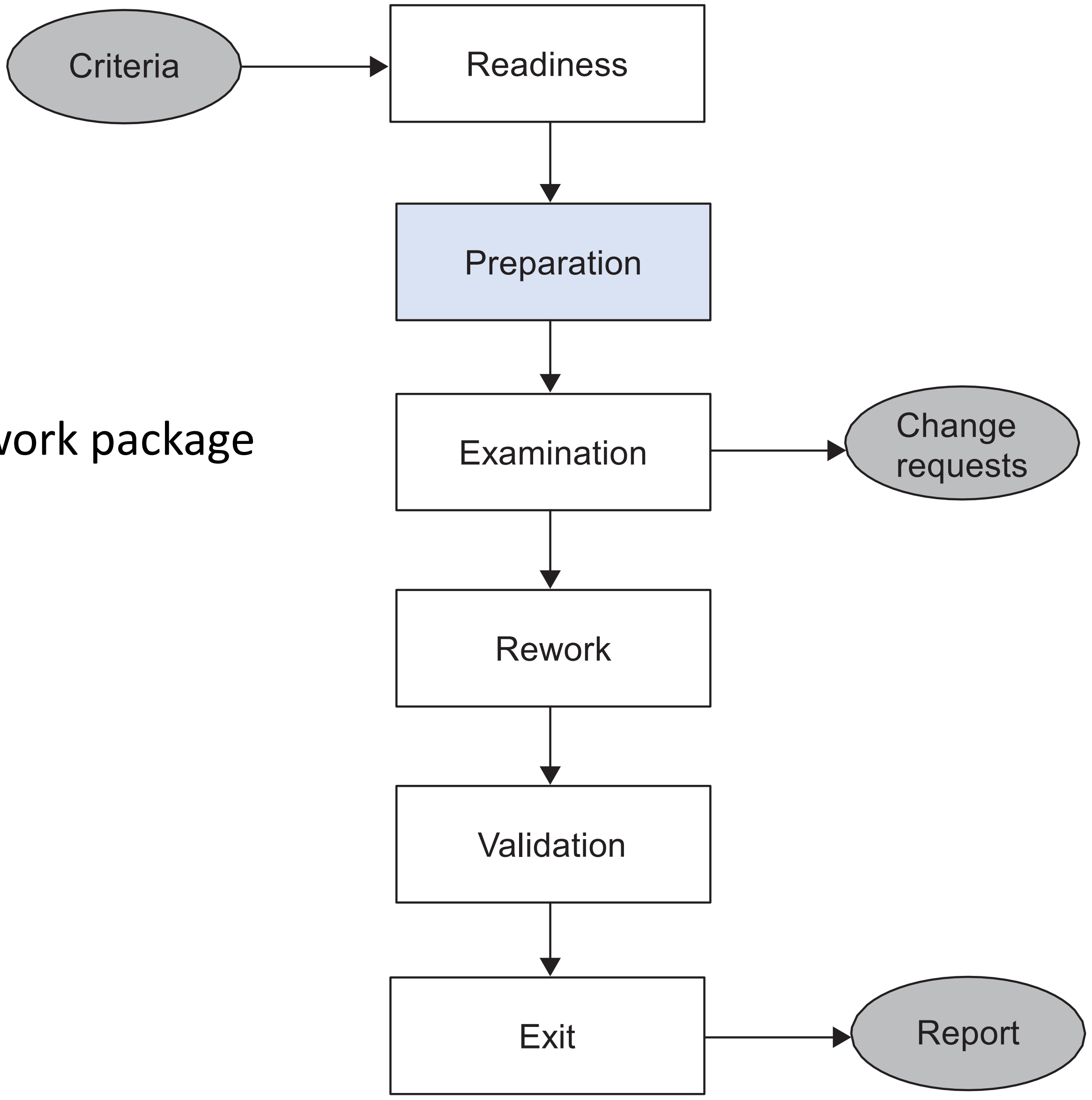
- Main steps are
  - Readiness
    - unit under test is ready for review
      - Completeness
      - Minimal Functionality
      - Readability
      - Complexity
      - Requirements
  - Preparation
  - Examination
  - Rework
  - Validation
  - Exit



MONASH University

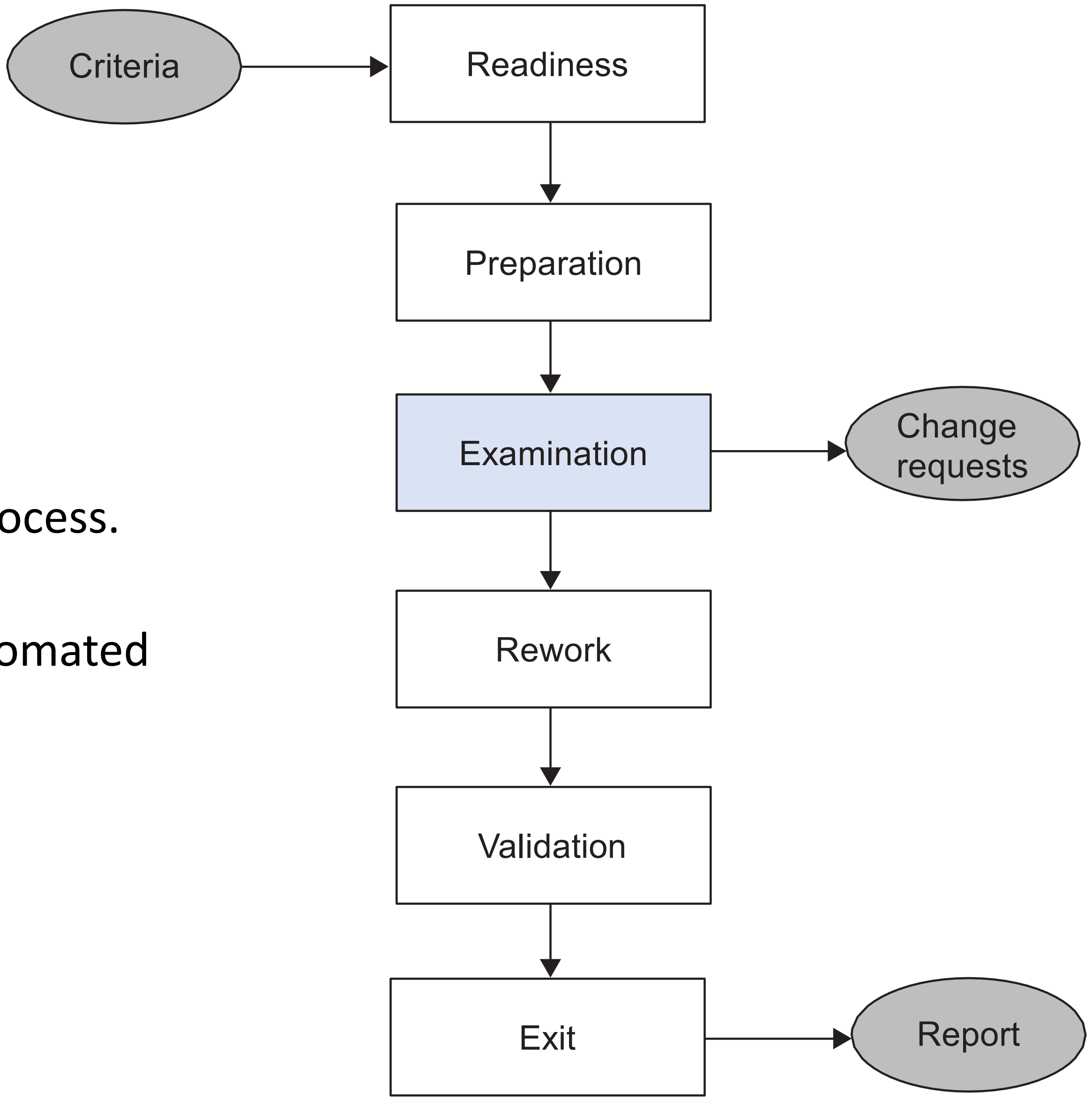
Code Review Process

- Main steps are
    - Readiness
    - Preparation
        - each reviewer/authors carefully reviews the work package
        - List of questions
        - Checklists
        - Change Requests (optional)
        - Suggested improvements (optional)
    - Examination
    - Rework
    - Validation
    - Exit



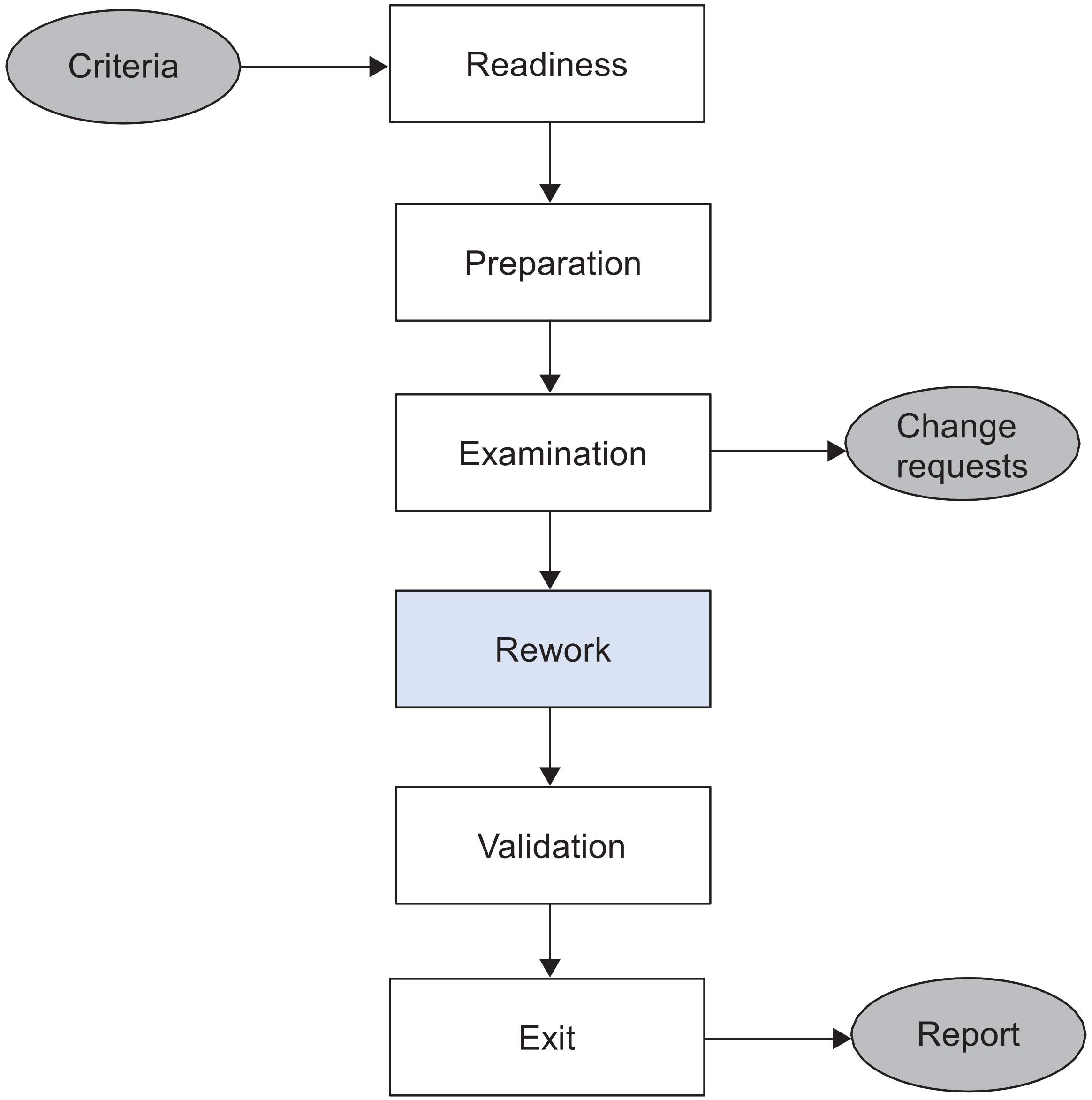
Code Review Process

- Main steps are
  - Readiness
  - Preparation
  - Examination
    - Varies from teams to teams and process to process.
    - XP uses pair programming
    - Issues/defects identification -> manual or automated
    - If require more meetings (optional)
  - Rework
  - Validation
  - Exit



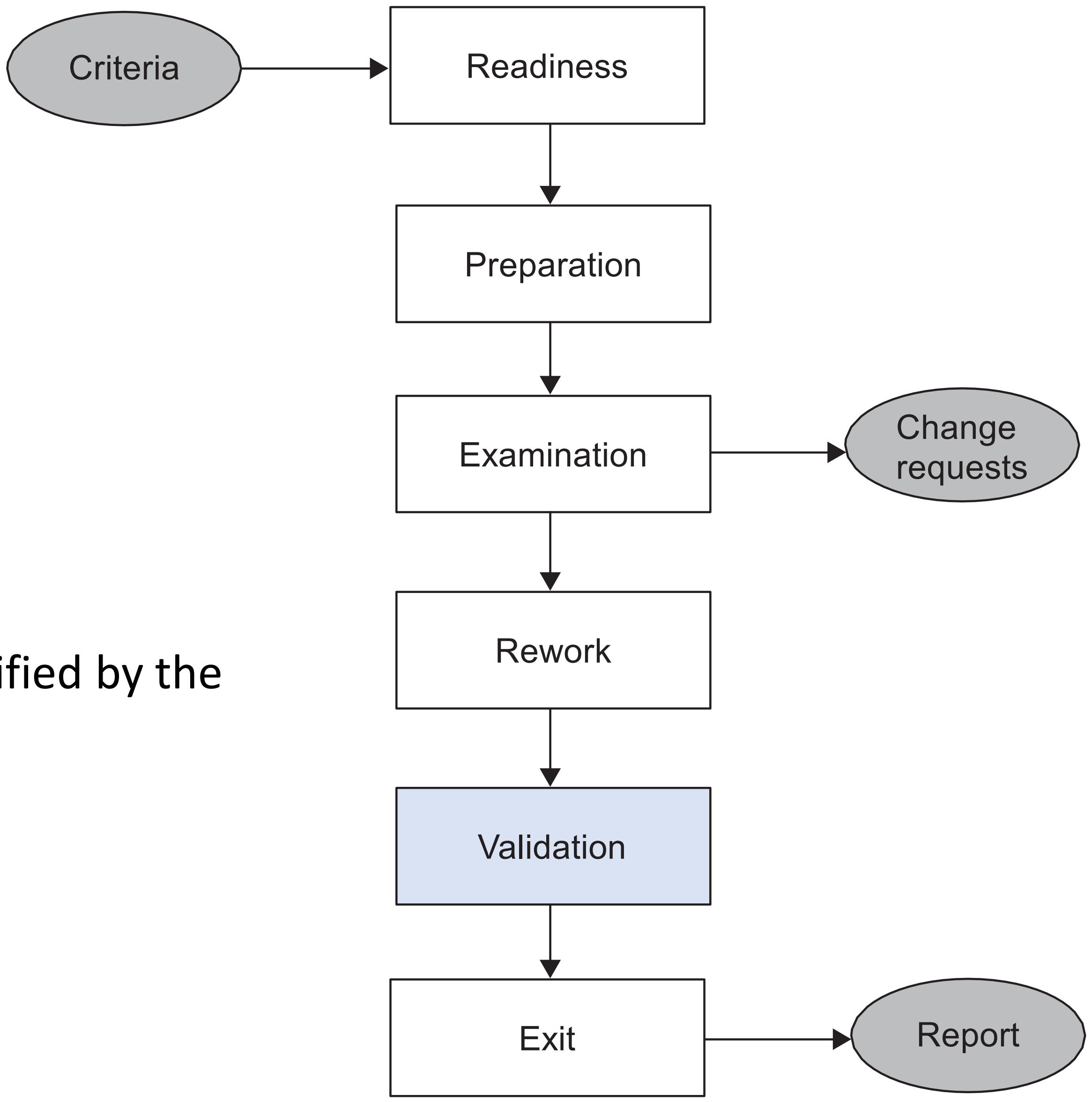
Code Review Process

- Main steps are
  - Readiness
  - Preparation
  - Examination
  - Rework
    - Summary
    - Change requests/issues
    - Who is responsible to fix them
    - Minutes (optional)
  - Validation
  - Exit



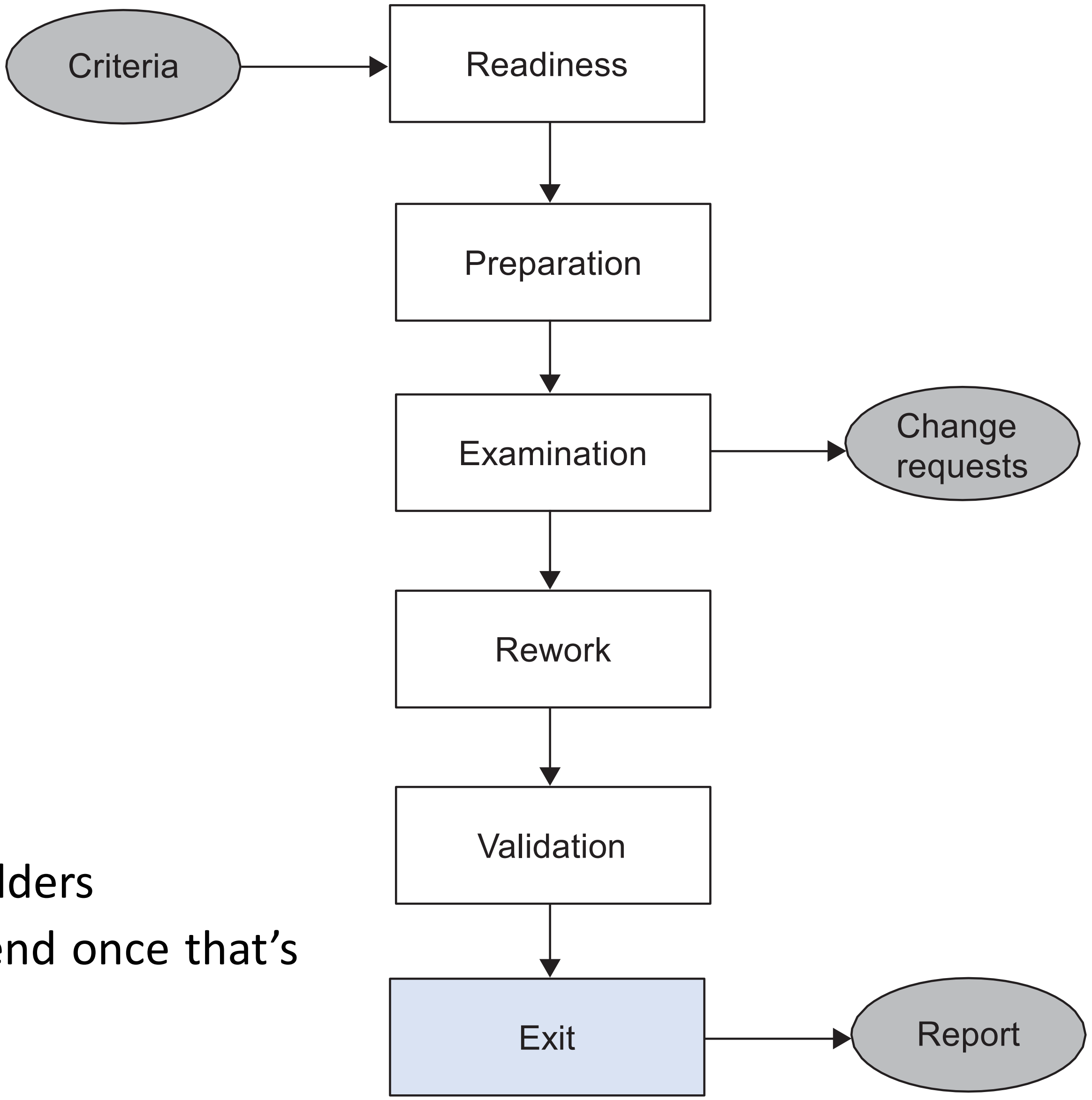
Code Review Process

- Main steps are
  - Readiness
  - Preparation
  - Examination
  - Rework
  - Validation
    - The change requests/issues identified are verified by the moderators
    - The outcome
  - Exit



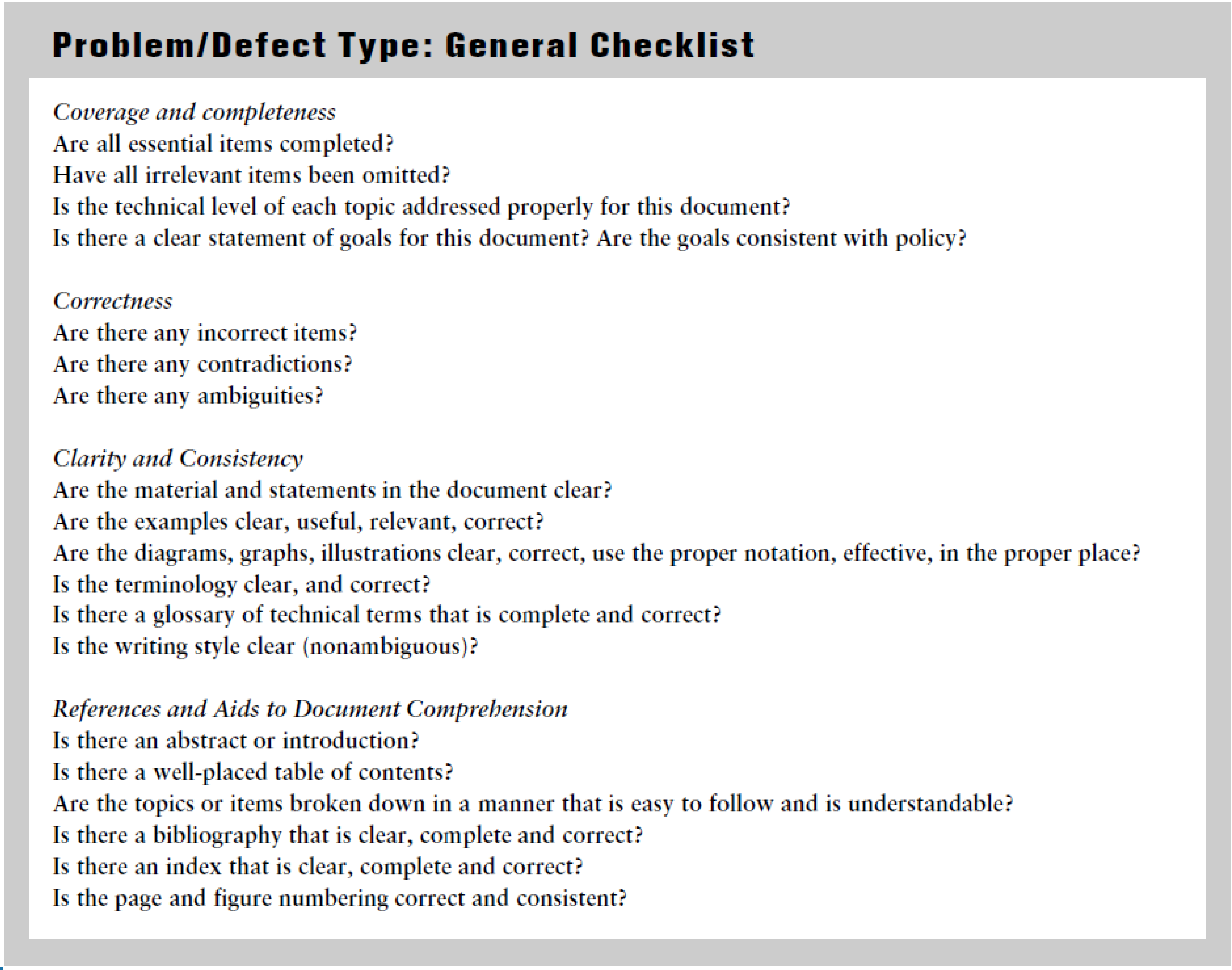
Code Review Process

- Main steps are
    - Readiness
    - Preparation
    - Examination
    - Rework
    - Validation
    - Exit
        - It also varies how the process is conducted
        - All reviews done
        - Summary report and submitted to all stakeholders
        - If defects need to be fixed, the process will end once that's done



Checklists

- Sample Check List
- The team must create their own checklists.
- The checklist items may vary from project to project.
- It also depends on the artefact to be reviewed.



Problem/Defect Type: General Checklist

Coverage and completeness
Are all essential items completed?
Have all irrelevant items been omitted?
Is the technical level of each topic addressed properly for this document?
Is there a clear statement of goals for this document? Are the goals consistent with policy?

Correctness
Are there any incorrect items?
Are there any contradictions?
Are there any ambiguities?

Clarity and Consistency
Are the material and statements in the document clear?
Are the examples clear, useful, relevant, correct?
Are the diagrams, graphs, illustrations clear, correct, use the proper notation, effective, in the proper place?
Is the terminology clear, and correct?
Is there a glossary of technical terms that is complete and correct?
Is the writing style clear (nonambiguous)?

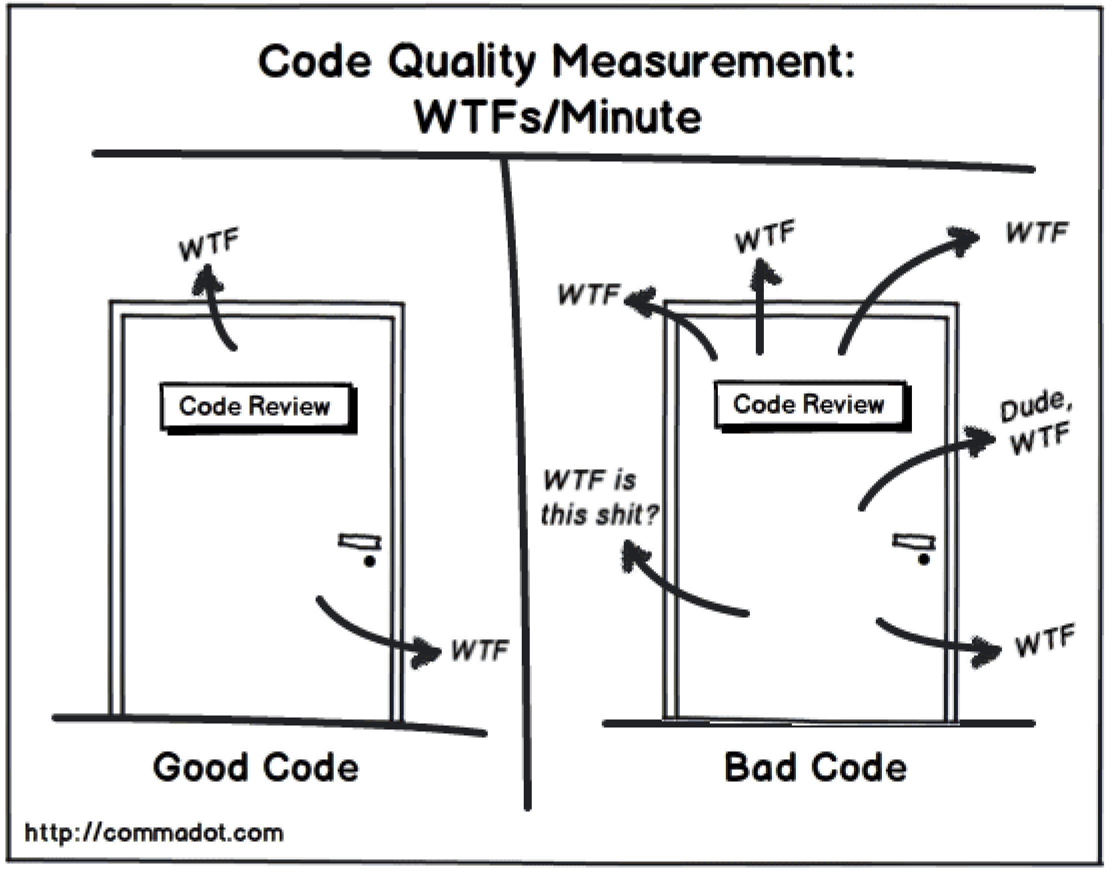
References and Aids to Document Comprehension
Is there an abstract or introduction?
Is there a well-placed table of contents?
Are the topics or items broken down in a manner that is easy to follow and is understandable?
Is there a bibliography that is clear, complete and correct?
Is there an index that is clear, complete and correct?
Is the page and figure numbering correct and consistent?

Getting the best from reviews

- The author
  - "… is in the hot seat"
  - How do you react?
- The development team
  - Better prepared
  - Feedback +/–
  - Communication
- The review team
  - Critical thinking
  - Ability to detect omissions
  - Who should participate in the review?
- Cost-effective verification
  - Minimising cost of correction
  - Is it cost-effective?

Code Quality Measure

- Cars have KPH or MPH
  - The BETTER the road the more MPH or KPH it can attain
- With Code Quality the measure is WTFPM
  - the WORSE the code the more WTFPM can be obtained.
- With Code Reviews
  - WTFPM should be relatively small and controllable.

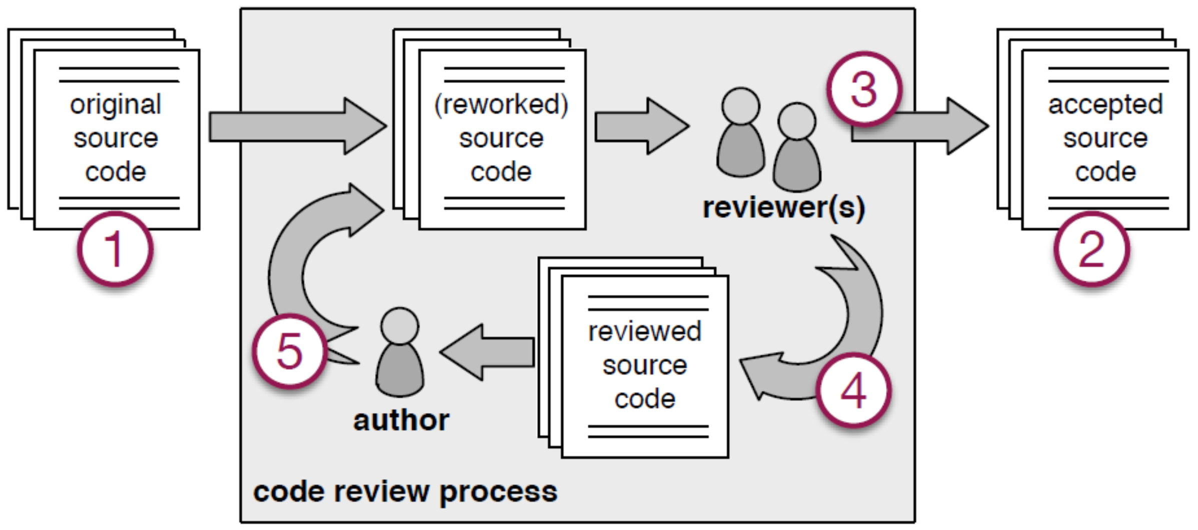


PART C:
MODERN CODE REVIEWS

Modern Code Reviews (MCR)

- It's a review
    - Inspection
- Opensource as well as proprietary software
- Lightweight
- Fewer formal requirements
- Shorter time
- Agile -> XP
- Automated or Semi Automated
    - GIT
    - Gerrit

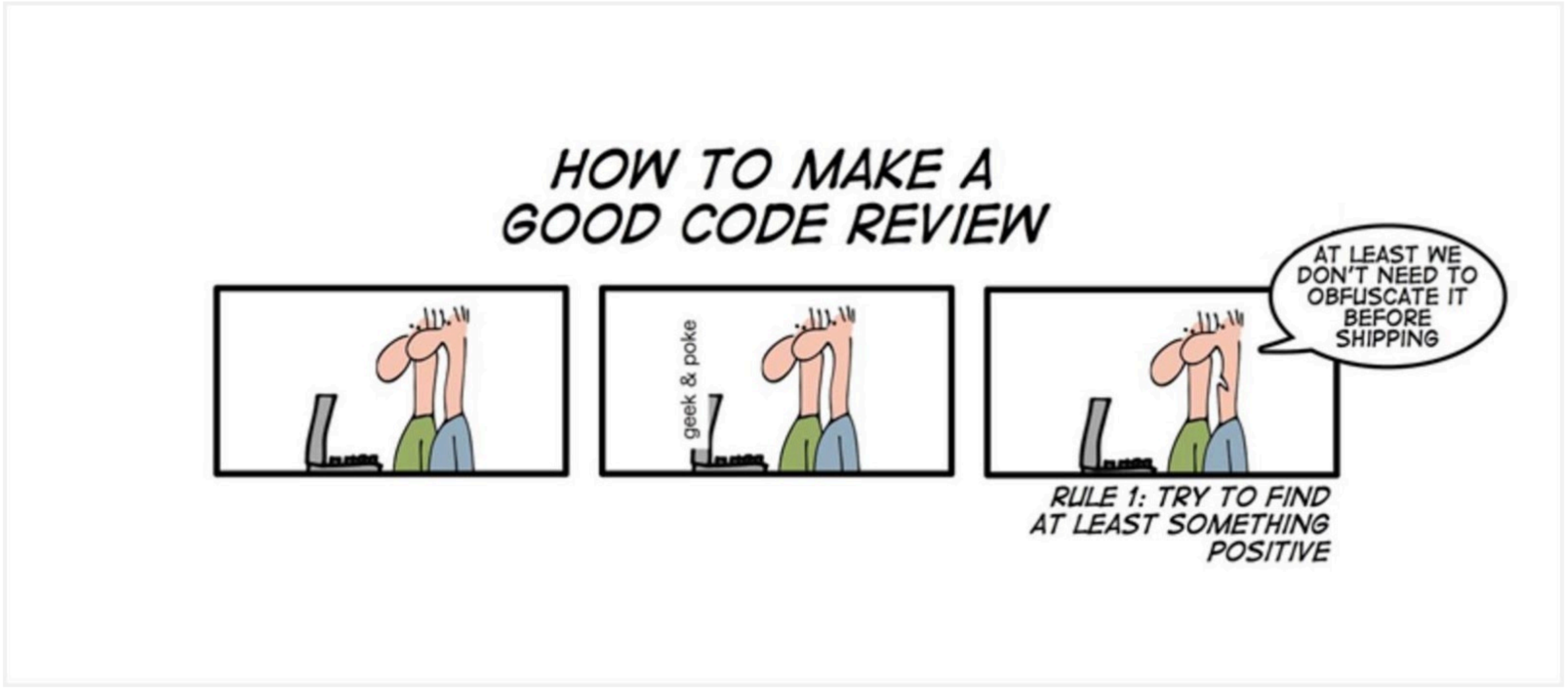
MCR Process



- The original source code considered for review
- The authors rework and make it available for review
- Reviewers review the code, list the issues or defects and suggest changes
- The authors update the code and review process runs again.
- If reviewers accept the code in the first instance than the code is considered accepted.

PART D:
GOOD CODE REVIEWS

How to make a good review?

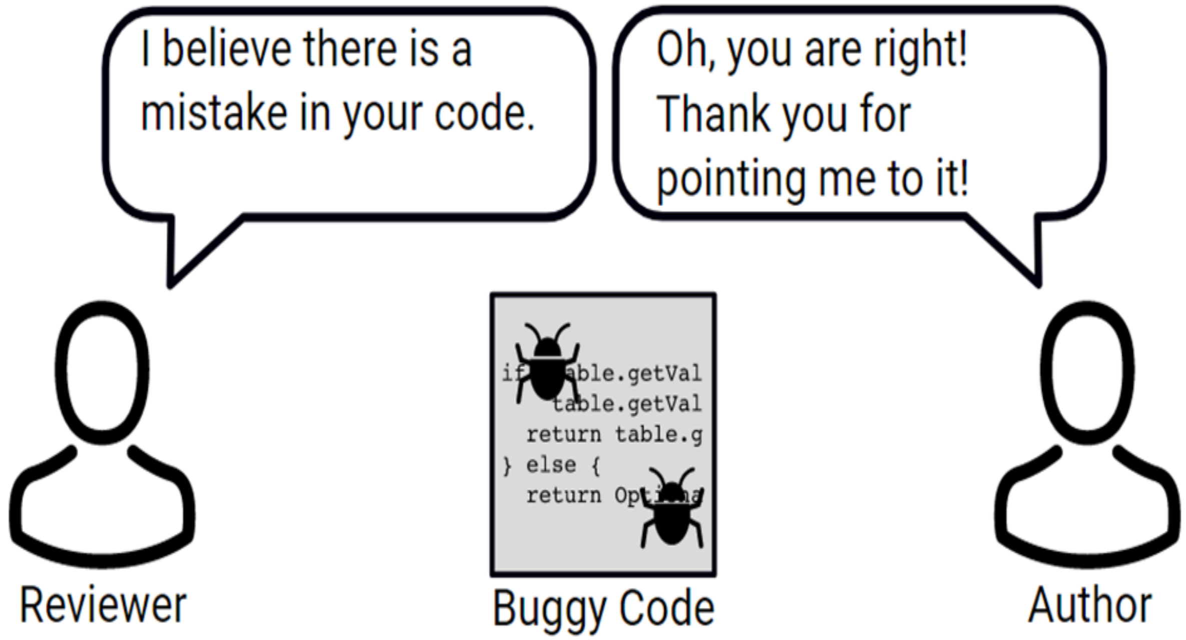

HOW TO MAKE A GOOD CODE REVIEW
geek & poke
AT LEAST WE DON'T NEED TO OBFUSCATE IT BEFORE SHIPPING
RULE 1: TRY TO FIND AT LEAST SOMETHING POSITIVE

- Follow Guidelines
- Experience

Review Guidelines

For Authors

- Be humble
    - No Body is perfect
    - Mistakes happen
    - Many ways to do the same thing
    - Learning
    - Self Improvement

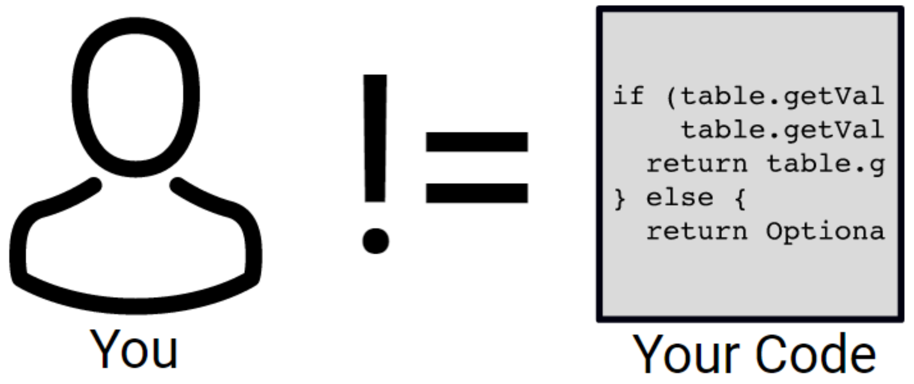

I believe there is a mistake in your code.
Oh, you are right! Thank you for pointing me to it!
Reviewer
Buggy Code
Author

```
if  able.getVal
    table.getVal
  return table.g
} else {
  return Opt  a
```

MONASH University

Review Guidelines

For Authors



```
if (table.getVal
    table.getVal
  return table.g
} else {
  return Optiona
```

- Your code doesn't represent you as a person
  - Flaws in your code doesn't mean that there are flaws in you
  - Feedback about your code is essentially just that – it is not about you!
  - Comments and recommendations not as a personal attack.
  - If you receive poor feedback you should not lose confidence in yourself or your abilities.

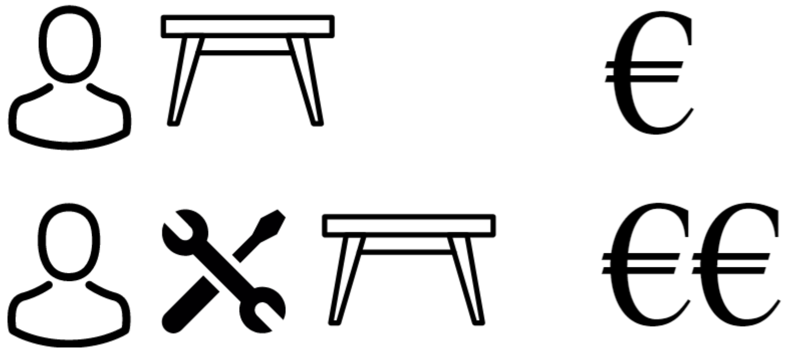
Review Guidelines

For Authors

- The reviewer and you are friends
  - Usually, the reviewer will also be a stakeholder in the project you are developing the code or deliverable for.
  - Hence, there is no reason for the reviewer to make your job difficult purposely and unnecessarily as this also impacts them negatively!
  - Reviewers are out there to help you achieve the common goal that the both of you (and the entire team) share, which is to deliver good quality software which meets the project's budget, timeframe and requirements.

Review Guidelines

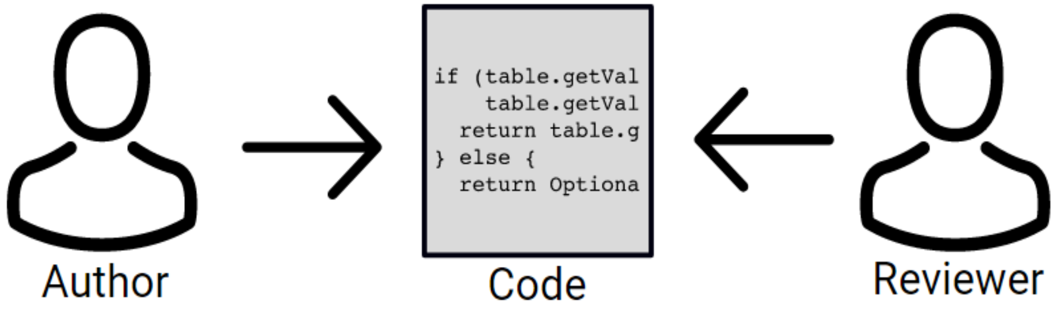
For Authors



- IKEA Effect (Cognitive Bias)
  - A consumer places a disproportionately high value on products they partially created.
  - The effect was demonstrated by the following experiment:
    - Two groups should price the value of IKEA furniture. One group got already assembled furniture, the other group had to assemble them first. The results showed that the second group was willing to pay 63 % more than the first group.
  - you will usually value your own code more highly compared to equivalent code written by someone else
  - Emotional Attachment

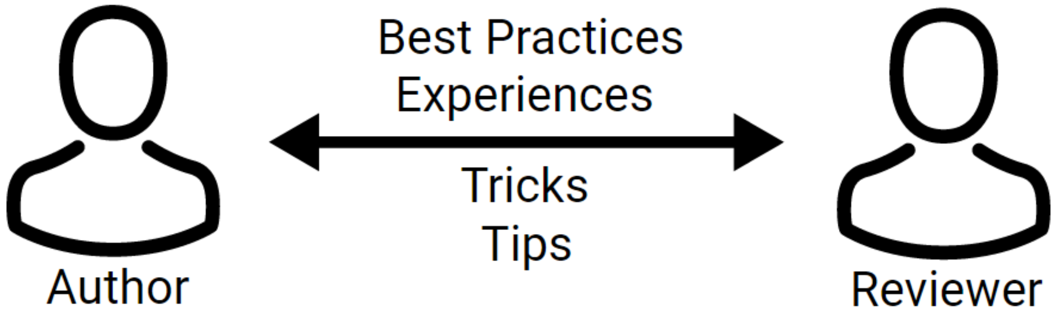
Review Guidelines

For Authors



```
if (table.getVal
    table.getVal
  return table.g
} else {
  return Optiona
```

- Be Open to others' perspectives
  - Everyone sees things differently
  - Different background, assumptions, knowledge etc.
  - You are exchanging best practices and experiences.
  - Valuable source of knowledge and learning.
- Bandwagon Effect
  - Do not blindly oppose or accept everything the reviewer says
  - But it doesn't mean that you underestimate the reviewers' knowledge



Review Guidelines

For Reviewers

- Focus on reviewing the code only
    - objectively review the code and not subjectively review the coder!

> Wrong: “**You’re requesting** the service multiple times, which is inefficient.“

> Right: “**This code is requesting** the service multiple times, which is inefficient.“

- Collective Code Ownership

Review Guidelines

For Reviewers

- Use I-Messages
  - Increase the acceptance of your feedback by using I-messages

> Wrong: “**You** are writing cryptic code.“

> Right: “It’s hard **for me** to grasp what’s going on in this code.“

Review Guidelines

For Reviewers

- Prefer asking questions
    - Asking questions is a soft way to give feedback and to discover the author’s intention.
    - Not criticism.

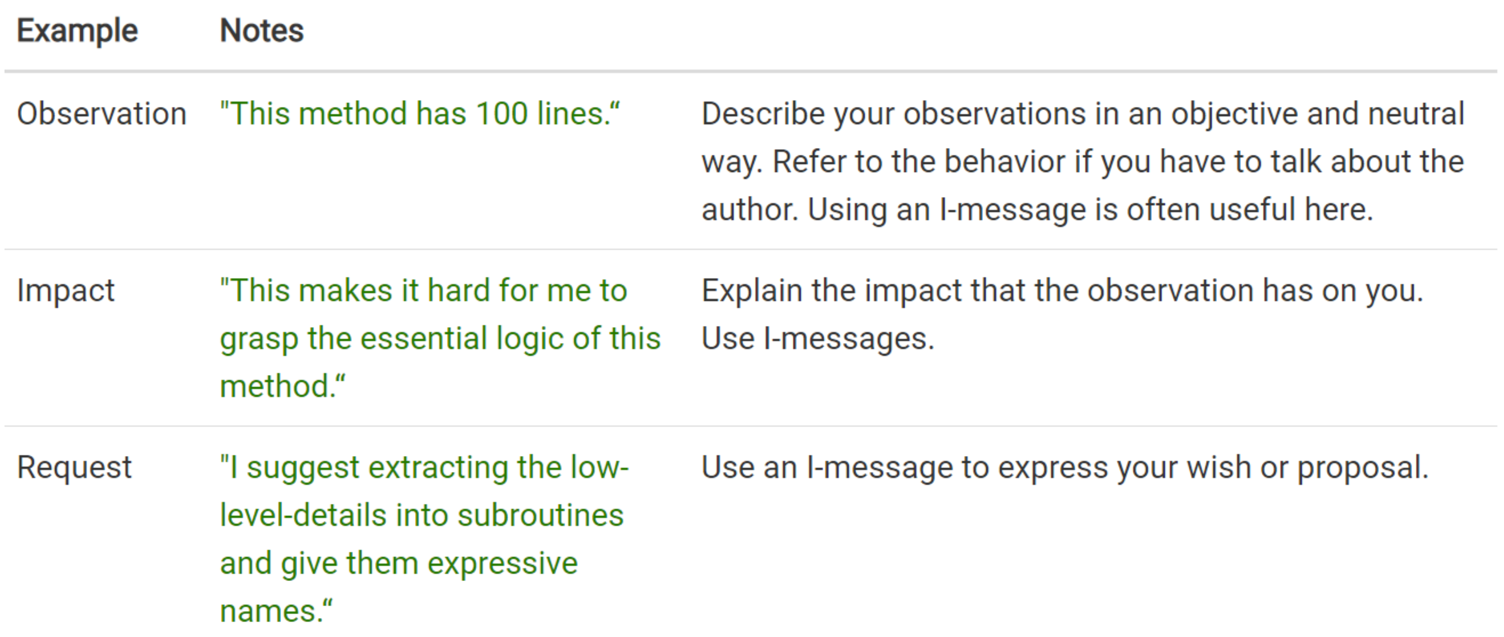
> Wrong: “This variable should have the name ‘userId’.“

> Right: “What do you think about the name ‘userId’ for this variable**?**“

Review Guidelines

For Reviewers

- OIR Rule of giving Feedbacks
    - Structure the feedback into three parts: Observation, Impact and Request.



Example	Notes	
Observation	"This method has 100 lines."	Describe your observations in an objective and neutral way. Refer to the behavior if you have to talk about the author. Using an I-message is often useful here.
Impact	"This makes it hard for me to grasp the essential logic of this method."	Explain the impact that the observation has on you. Use I-messages.
Request	"I suggest extracting the low-level-details into subroutines and give them expressive names."	Use an I-message to express your wish or proposal.

Review Guidelines

For Reviewers

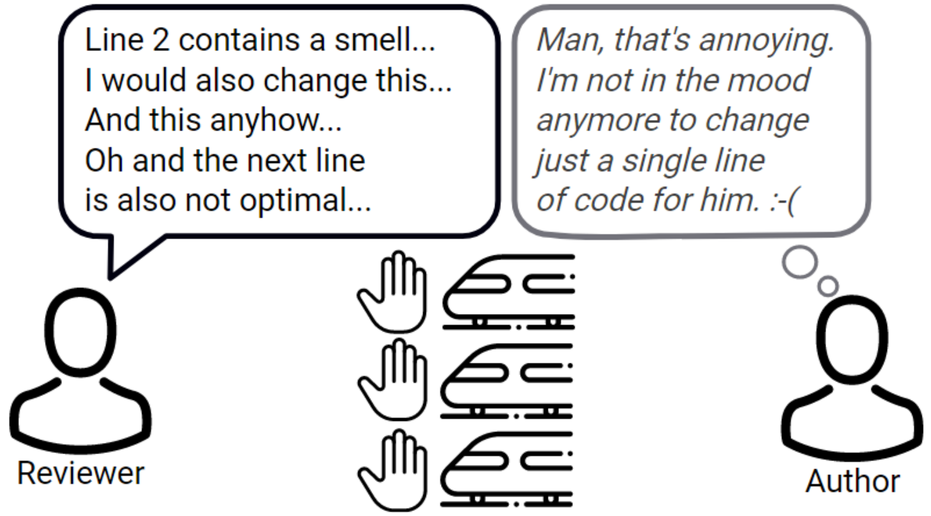
- Accept that there are different solutions
  - Be aware of your personal taste, accept other solutions and make compromises.

> Wrong: “I always use fixed timestamps in tests and you should too.“

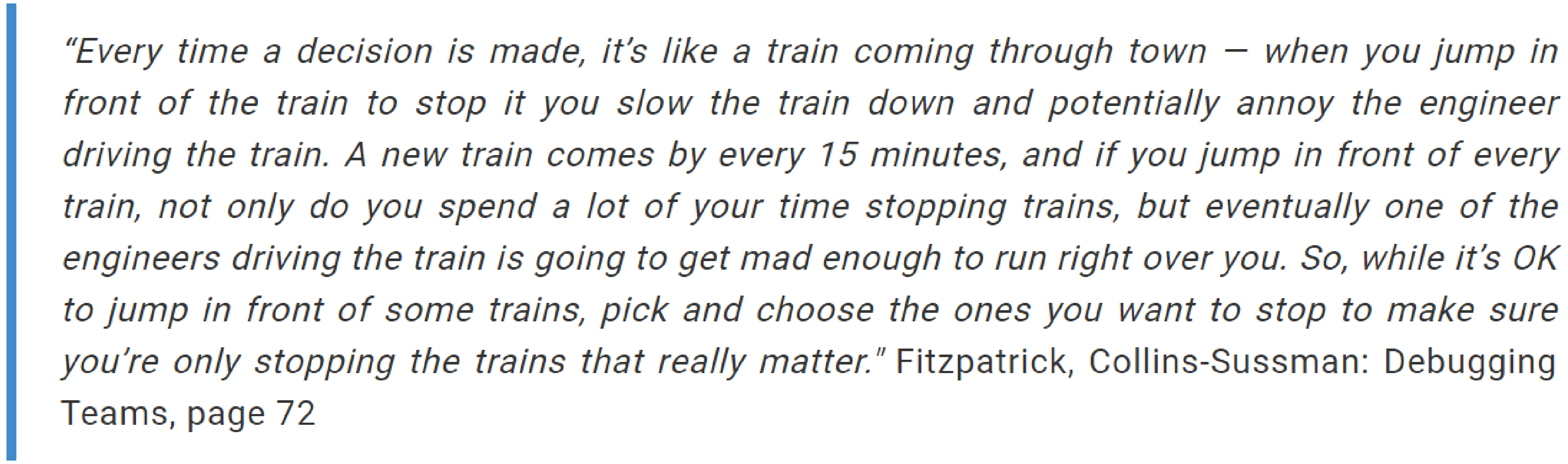
> Right: “I would always use fixed timestamps in tests for better reproducibility but in this simple test, using the current timestamp is also ok.“

Review Guidelines

For Reviewers


Line 2 contains a smell...
I would also change this...
And this anyhow...
Oh and the next line
is also not optimal...
Man, that's annoying. I'm not in the mood anymore to change just a single line of code for him. :-(
Reviewer
Author

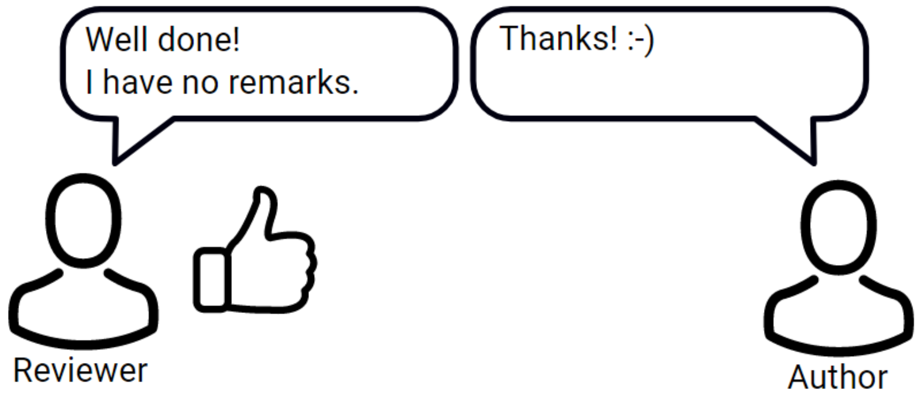
- Don’t jump in front of every train
  - Be Don’t be a pedant.
  - Don’t criticize every single line of code.



> *“Every time a decision is made, it’s like a train coming through town — when you jump in front of the train to stop it you slow the train down and potentially annoy the engineer driving the train. A new train comes by every 15 minutes, and if you jump in front of every train, not only do you spend a lot of your time stopping trains, but eventually one of the engineers driving the train is going to get mad enough to run right over you. So, while it’s OK to jump in front of some trains, pick and choose the ones you want to stop to make sure you’re only stopping the trains that really matter.”* Fitzpatrick, Collins-Sussman: Debugging Teams, page 72

Review Guidelines

For Reviewers



- Praise
  - Don’t forget to express your appreciation if you have reviewed good code.
  - Praising doesn’t hurt you but will motivate the author and improve your relationship.

> Wrong: “Most of your code looks good, but the method `calc()` is too big.“

> Right: “I really like the class `ProductController`, Tim. It has a clear single responsibility, is coherent, and contains nicely named methods. Good Job!
> Despite this, I spotted the method `calc()` which is too big for me.“

Review Guidelines

For Reviewers

- Three Filters for Feedback
  - Is it True?
  - Is it Necessary?
  - Is it Kind?

Is it True?

> Wrong: “You should use getter and setter. This code is wrong.“

> Right: “In this case, I would recommend using getter and setter, because…“

Is it Necessary?

> Wrong: “We should refactor this whole package.“

> Right: “Let’s discuss the refactoring in a dedicated meeting.“

Is it Kind?

> Wrong: “A factory is **badly over-engineered** here. The **trivial** solution is to **just** use the constructor.“

> Right: “This factory feels complicated to me. Have you considered to use a constructor instead?“

PART E:

GOOD PROGRAMMING PRACTICES

Good Programming Practices

The day you write your code, you and God understand how it works. Six months later, only God does.

-An old saying but not too old

Good Programming Practices

Meaningful Names

- Proper and meaningful names
- Recognisable
- You want to declare a variable that calculates the weigh of truck.
    - tw
    - trkw8
    - truckWeight
    - truck_weight

Good Programming Practices

Commenting & Documentation

- Name of the module
- Purpose of the Module
- Description of the Module
- Original Author
- Modifications
- Authors who modified code with a description on why it was modified.

Good Programming Practices

Code Indentation

- No right or wrong standard
- Consistent
- Some languages are strict about indentation
  - Python
- Some not
  - Java

Good Programming Practices

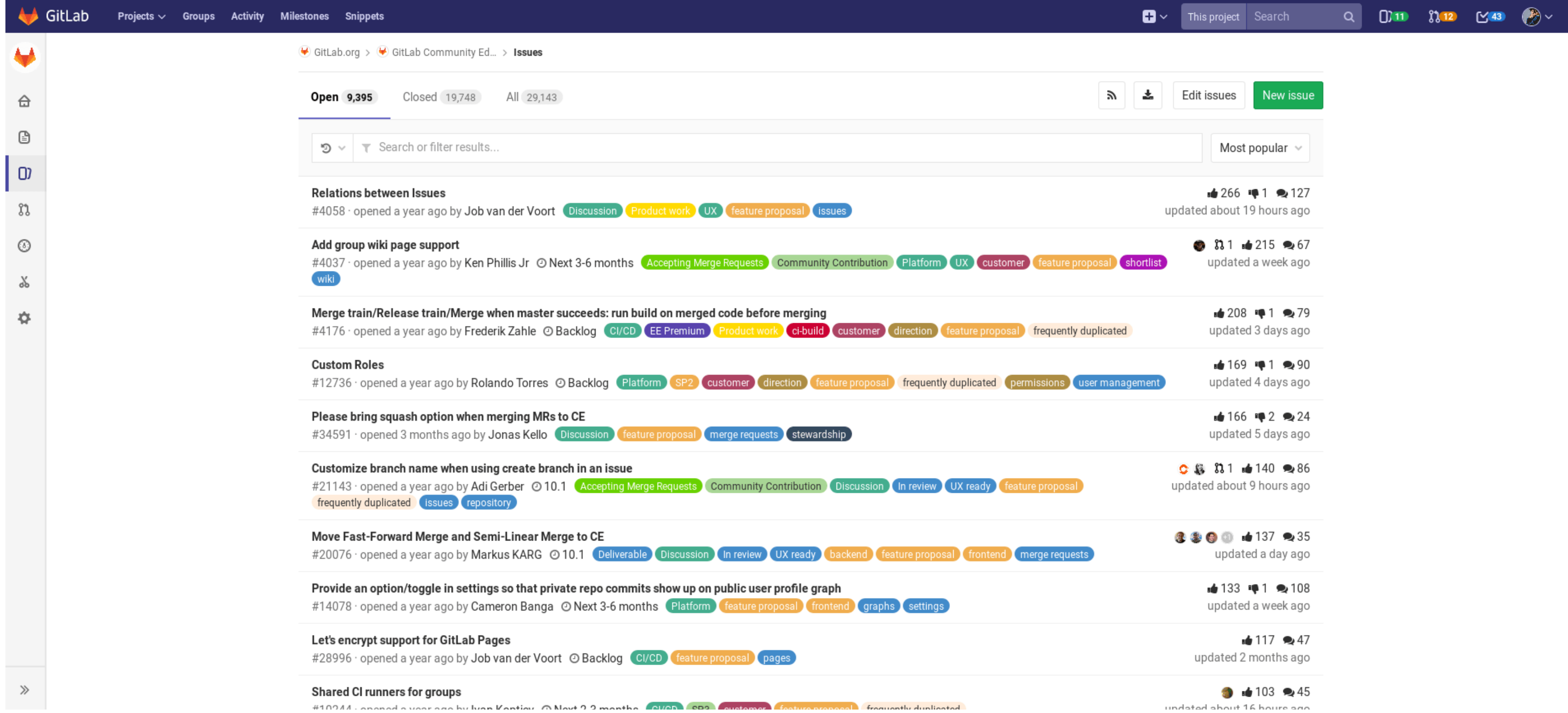
Code Smells

- Duplications
    - Apply DRY principle
- Multiple parameters
    - Not more than three
        - Extract Method
        - Parameter Object Pattern
- Excessive use of literals
- God Class/Object
- https://refactoring.com/catalog/

PART F:
CODE REVIEW TOOLS

Code Review Tools

- Many tools -> some famous and commonly used tools are
  - GITLAB
  - GERRIT
  - Static Analysis tools
    - Pylint



Relations between Issues
#4058 · opened a year ago by Job van der Voort Discussion Product work UX feature proposal issues — 266 1 127 updated about 19 hours ago

Add group wiki page support
#4037 · opened a year ago by Ken Phillis Jr Next 3-6 months Accepting Merge Requests Community Contribution Platform UX customer feature proposal shortlist wiki — 1 215 67 updated a week ago

Merge train/Release train/Merge when master succeeds: run build on merged code before merging
#4176 · opened a year ago by Frederik Zahle Backlog CI/CD EE Premium Product work ci-build customer direction feature proposal frequently duplicated — 208 1 79 updated 3 days ago

Custom Roles
#12736 · opened a year ago by Rolando Torres Backlog Platform SP2 customer direction feature proposal frequently duplicated permissions user management — 169 1 90 updated 4 days ago

Please bring squash option when merging MRs to CE
#34591 · opened 3 months ago by Jonas Kello Discussion feature proposal merge requests stewardship — 166 2 24 updated 5 days ago

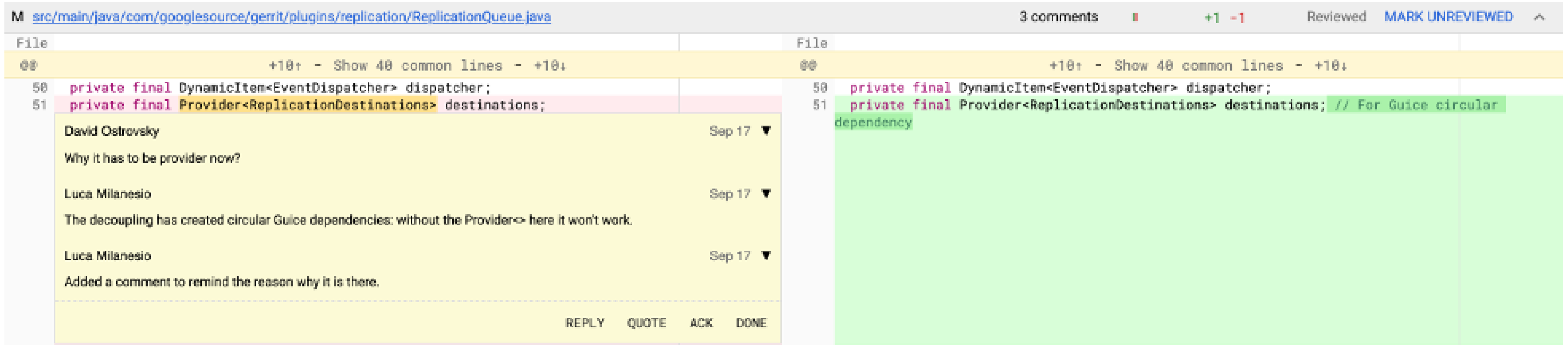
Customize branch name when using create branch in an issue
#21143 · opened a year ago by Adi Gerber 10.1 Accepting Merge Requests Community Contribution Discussion In review UX ready feature proposal frequently duplicated issues repository — 1 140 86 updated about 9 hours ago

Move Fast-Forward Merge and Semi-Linear Merge to CE
#20076 · opened a year ago by Markus KARG 10.1 Deliverable Discussion In review UX ready backend feature proposal frontend merge requests — 137 35 updated a day ago

Provide an option/toggle in settings so that private repo commits show up on public user profile graph
#14078 · opened a year ago by Cameron Banga Next 3-6 months Platform feature proposal frontend graphs settings — 133 1 108 updated a week ago

Let's encrypt support for GitLab Pages
#28996 · opened a year ago by Job van der Voort Backlog CI/CD feature proposal pages — 117 47 updated 2 months ago

Shared CI runners for groups — 103 45


M src/main/java/com/googlesource/gerrit/plugins/replication/ReplicationQueue.java
3 comments
+1 -1
Reviewed
MARK UNREVIEWED
File
@@
+10↑ - Show 40 common lines - +10↓

```
50  private final DynamicItem<EventDispatcher> dispatcher;
51  private final Provider<ReplicationDestinations> destinations;
```

David Ostrovsky — Sep 17
Why it has to be provider now?

Luca Milanesio — Sep 17
The decoupling has created circular Guice dependencies: without the Provider<> here it won't work.

Luca Milanesio — Sep 17
Added a comment to remind the reason why it is there.

```
50  private final DynamicItem<EventDispatcher> dispatcher;
51  private final Provider<ReplicationDestinations> destinations; // For Guice circular dependency
```

QUESTIONS