Code Reviews

Contents

{ CITATION Kai08 \l 1033 }

Code Reviews

Code Reviewing is a systematic approach of examining the source code in detail. The goal is to assess the quality of the source code not the quality of the process or the author of the code. Code reviews must be planned and executed in a professional manner so there is a need for mutual respect, openness, trust, and sharing of expertise in the group.

Inspections are a step-by-step peer group review of a work product, with each step checked against a predetermined criteria.

Walkthroughs are reviews where the author leads the team through a manual or simulated execution of the product using predetermined scenarios.

"**Buddy checks**" are very informal reviews where someone else examines the source code.

Steps in the Code Review Process

1. Readiness
2. Preparation
3. Examination
4. Rework
5. Validation
6. Exit

Readiness

The author of the unit ensures that the unit is ready for review. A unit is ready for review if it meets the following criteria:

- **Completeness** – All related code must be available.
- **Minimal Functionality** – The code must compile and link and must be tested to make sure it meets its basic functionalities.
- **Readability** – The code must be readable based on the defined standards.
- **Complexity** – The code must be relatively complex to warrant a code review. McCabe's Complexity Measure is a good value to use. The formula to calculate the cyclomatic complexity of a program graph is $v=e-n+2$. Where v = cyclomatic complexity, e = number of edges, n = number of nodes. If there is an edge between the entry node and the exit node then the formula is $v=e-n+1$.
- **Requirements and Design Documents** – The latest versions of the requirements and design documents should be available.

The review group involves a number of people with different roles:

- **Moderator** – The chair of the review meeting. He or she chooses the reviewers and schedules the review meetings. If possible, the moderator should be external to the project.

- **Author** – The person who wrote the code to be reviewed.
- **Presenter** – The person who presents the code and should not be the author.
- **Record-keeper** – The person who documents the problems found during the review process and the suggested follow-up action. The record-keeper should not be the moderator or the author.
- **Reviewers** – These are the "experts" in the subject area of the code under review.
- **Observers** – These are people who want to learn about the code under review. They do not participate in the review process.

Preparation

Before the meeting, each reviewer carefully reviews the code. Each review develops the following:

- **List of Questions**
- **Potential CR** – A reviewer may issue a change request (CR).
- **Suggested Improvement Opportunities** – Reviewers may suggest how to fix any identified problems

Examination

1. The author gives an overview of the logic, computation, and dependencies used in the code.
2. The presenter reads the code line by line. The reviewers may raise questions if the code is seen to have defects. However, problems are not resolved in the meeting. The author is responsible for correcting the code.
3. The record-keeper documents the CR and any suggestions on how to fix the problems (if any). A CR includes the following details:
   3.1. A brief history of the issue or action item
   3.2. A priority level
   3.3. A person to follow up the issue.
   3.4. A deadline for addressing the CR.
4. The moderator ensures that the meeting remains focused on the review process.
5. At the end of the meeting, a decision is taken whether or not to call another meeting to further review the code.

Rework

At the end of the meeting the record-keeper produces a summary of the meeting that includes the following information:

- A list of all CRs, the deadlines, and the follow-up persons
- A list of improvement opportunities
- The minutes of the meeting (optional)

Validation

The CRs are independently validated by the moderator or another person designated for that purpose.

Exit

The review process is complete if the following actions have been taken:

- All lines of code in the unit has been inspected.
- If too many defects are found in a module, the module is once again reviewed after corrections are applied by the author.
- The author and the reviewers reach a consensus that when corrections have been applied the code will be potentially free of defects.
- All the CRs are documented and validated by the moderator or someone else. The author's actions are documented.
- A summary report of the meeting including the CRs is distributed to all the members of the review group.

Code Review Metrics

- Number of lines of code (LOC) reviewed per hour
- Number of CRs generated per thousand lines of code (KLOC)
- Number of CRs generated per hour
- Total number of CRs generated per project
- Total number of hours spent on code review per project

Code Review Checklist (A Few)

Data Reference Errors	
	For all array references, each subscript is an integer within the defined bounds
	There are no "off-by-one" errors
	There are no unseen or uninitialized variables referenced
Data Declaration Errors	
	The definition of a variable is consistent with its declared type
	Variable names are meaningful to its intent
	Variable types are consistent to its intended use.
Computation Errors	
	The types on both sides of an assignment statement are consistent and will not result in data loss.
	There is no possibility of data overflow or underflow.
	There is no possibility of division by zero.
Comparison Errors	
	Floating point equality comparisons are handled correctly.
	The correct comparison operators are being used.
	The ranges in the comparisons are consistent with the specifications.
Control Flow Errors	
	Loops will terminate eventually
	Loops will not exit prematurely
	Loops that can be skipped are not an oversight.
	There are no off-by-one errors.
	All statements inside a loop are essential to the loop.

Commenting and Maintainability	
	All comments are necessary.
	All comments are up-to-date.
	There is no commented out code.
	There are no TODOs or WIPs.
	Literal values are replaced by meaningful constant identifiers
	The indent of an identifier is clear.
Error Handling	
	Error messages are understandable and complete.
	Parameters and arguments are valid.
	Exceptions are thrown and handled appropriately.
	Method preconditions are ensured before the method is called.
Code Decisions	
	The code is at the right level of abstraction.
	Method calls have the appropriate number and type of arguments.
	Class members have the appropriate accessibility
	Enums used instead of int constants
	Dependencies are abstracted and are injected
	All new objects are necessary
	The code does not pose a security concern.
Resource Leaks	
	All resources are released.
	Resources are released once.
	The most efficient class is used.

References

Naik, K., & Tripathy, P. (2008). *Software Testing and Quality Assurance Theory and Practice.* Wiley.