Software Review and Inspection

Overview of Software Review and Inspection

Software quality assurance is a structured approach to improve software quality. It involves defect prevention, detection, and removal, as well as defect containment activities. Software testing and inspection are the two (2) commonly used methods to detect defects in software, while software inspections and reviews are concerned with the review of software artifacts.

Software inspection or **software review** is a formalized peer review process for detecting and correcting defects or bugs in software artifacts. Software artifacts are any things that have been documented and stored during the software development process, such as requirements document, source code, user documentation, and test plans and test cases. Requirements, design, and other non-executable artifacts cannot be tested by machines and have to be reviewed by humans. With inspection, both requirements and design can be read and analyzed, which do not require the artifacts to be complete or executable. The objective of software inspections is to help developers build quality into the software product.

Software Inspection Techniques

There are several approaches to software inspections, and these vary in the level of formality employed. There are formal software inspection methodologies that include different formal activities and guidelines. Some of the most well-known software inspection methodologies are the following:

- **Fagan inspection methodology** – This methodology was developed by Michael Fagan. The objectives of this methodology are to identify and remove errors in the software products and to identify any systematic defects in the processes that are used to create software products. This process requires that software artifacts be formally inspected by experts independent of the author.

  These are the following inspector roles that each participant plays as defined in the Fagan inspection process:

  - **Moderator** – S/He leads the inspection team and takes care of logistics.
  - **Author** – S/He is the creator of the software artifacts under inspection. The author will actively answer all the questions regarding the artifacts during the inspection meeting. The author is responsible for fixing identified defects and bugs in the artifacts.
  - **Reader** – S/He is an experienced peer who can be a subject matter expert on the software artifacts under inspection.
  - **Tester** – S/He is responsible for writing and executing test cases for the software module or product. The role of the tester is focused on how the product would be tested.

  The inspection process will also consider whether the software design is correct with respect to the requirements and whether the source code is correct with respect to the design. Refer to *Table 1* for the seven (7) activities of the Fagan inspection methodology.

ACTIVITY	ROLE	OBJECTIVE
Planning	Moderator	• Identify inspectors and roles. • Verify if the materials are ready for inspection. • Distribute inspection materials to participants. • Arrange meeting place and time.
Overview	Author	• Brief participants on information of materials.
Preparation	Inspectors	• Participants prepare for the meeting. • Checklists may be employed to facilitate the preparation. • Read through the deliverables and mark up issues and questions.
Inspection Meeting	Moderator/Inspectors	• The moderator sets the time limit for inspection and keeps the meeting focused. • The inspectors perform their roles to find defects. The emphasis is to identify as many possible defects, not solutions. • Record and classify defects. • An inspection outcome is agreed.

ACTIVITY	ROLE	OBJECTIVE
Process Improvement	Inspectors	• Continuously improve the development and inspection process. • Record the causes of major identified defects. • Perform root cause analysis to identify any systemic defect with development or inspection process. • Determine corrective action to address any systemic defects in the software process. • Make recommendations to the process improvement team.
Rework	Author	• The author corrects the defects and carries out any necessary investigations.
Follow-up	Moderator/Author	• The moderator verifies that the author has resolved the defects and investigations.

Table 1. Fagan inspection process
Source: Concise Guide to Software Engineering: From Fundamentals to Application Methods, 2017. p. 93

The inspection meeting consists of a formal meeting between the author and the inspectors. It is concerned with finding defects in the particular deliverable and verifying the correctness of the inspected material.

INSPECTION TYPE	PURPOSE	PROCEDURE
Requirements	• Find requirements defects. • Confirm if the requirements are correct.	• Inspectors review each page of requirements and raise questions or concerns. • The moderator records identified defects.
Design Inspection	• Find defects in design. • Confirm if the design is correct with respect to requirements.	• Inspectors review each page of design, compare it to requirements, and raise questions or concerns.
Code Inspection	• Find defects in the source code. • Confirm if codes are correct with respect to design and/or requirements.	• Inspectors review the code, compare it to requirements and/or design, and raise questions or concerns. • The moderator records identified defects.
Test Plan Inspection	• Find defects in test cases or test plan. • Confirm if test cases can verify the design and requirements.	• Inspectors review each page of test plan or specification, compare it to requirements and/or design, and raise questions or concerns. • The moderator records identified defects.

Table 2. Inspection meeting
Source: Concise Guide to Software Engineering: From Fundamentals to Application Methods, 2017. p. 98

The inspection meeting consists of a formal meeting between the author and the inspectors. It is concerned with finding defects in the particular deliverable and verifying the correctness of the inspected material.

- **Structured Walkthrough** – It is a peer review in which the author of a deliverable, such as a document or source code, brings one (1) or more reviewers through the deliverable. Its objectives are to get feedback from the reviewers on the quality of the document or code and to familiarize the review audience with the author's work. The walkthrough includes several roles, namely, the review leader, the author, the scribe, and the review audience.

  These are the following steps of a structured walkthrough:

  Step 1: The author circulates the deliverable, either physically or electronically, to the review audience.

  Step 2: The author schedules a meeting with the reviewers.

  Step 3: The reviewers familiarize themselves with the deliverable.

  Step 4: The review leader chairs the meeting.

  Step 5: The author brings the review audience through the deliverable, explains what each section aims to achieve, and requests comments from them as to its correctness.

Step 6: The scribe records errors, decisions, and any action items.

Step 7: A meeting outcome is agreed, and the author addresses all agreed items. If the meeting outcome results in a second review, then go back to Step 1.

Step 8: The deliverable is circulated to reviewers for sign off, and the reviewers sign off indicating that the author has correctly amended the deliverable.

Step 9: The author or project leader stores the comments and sign-offs.

The objective of software inspection is to build quality into the software product. It is more cost-effective to build quality during software development process rather than adding it later. The effectiveness of inspection is influenced by the expertise and experience of the inspectors, adequate preparation and the speed of the inspection, and compliance to the inspection process.

Code Reviewing Techniques

Code reviewing or **inspection** is the act of systematically convening with fellow programmers to check each other's code for faults to ensure the code meets quality standards. Code reviews may focus on coding standards only or as well as finding defects in the software code. The identified issues during the review are recorded and may include items requiring further investigation.

There are two (2) families of approach to identify defects in source codes:

- **Tool-Driven Code Review** – This approach uses automated code analysis techniques that can pick out problematic patterns within the source code. The source code analysis drive automated code reviews. The code checkers are often built-in into Integrated Development Environments (IDEs). The IDEs automatically assess the codes to identify code problems. For example, if a variable is declared but never used or used without being initialized, this is commonly flagged up without the developer having to invoke a tool explicitly.

  The nature of the code problems that can be identified varies from one (1) tool to another and depends to an extent on the nature of the underlying programming language. Automated tools are especially good at identifying problems that are relatively localized to a single method or class.

- **Developer-Driven Code Review** – Some properties of code quality are difficult to assess automatically. In this approach, the developers themselves reviews the code to pick out problems. Then, the reviewers manually assess source codes before deployment. This is usually a two-step process where the developers have to understand the program first so that they can form an opinion of its implementation and design. Only then they can properly evaluate the program.

  When evaluating the program, the specific goals depend on several factors. Developers may focus on evaluating the security and maintainability of the codes.

Modern Code Review

The **modern code review** is a lightweight variant of the code inspections because it does not rely on face-to-face meetings. This operates on tools and development processes.

The specifics of modern code review vary from one (1) development context to another. These also often depend upon the nature of the development process, the makeup of the development teams, and the workflow required by the associated code review tools used.

These are the two (2) version repository-based tools used to compose lightweight code reviews.

- **Tool-Driven Code Review** – To enable code reviews, integrate reviewing tools such as Google's Gerrit and Microsoft's CodeFlow with version repositories to act as gatekeepers to commits. A repository is a central file storage location. It is used by version control systems to store multiple versions of files that can be accessed by multiple users. *Figure 1* illustrates a typical modern code review workflow using a code reviewing tool.

1. The developer checks out a version from the repository and makes changes.
2. This change is then submitted for review. This is often automatically initiated when the developer tries to push his/her change to the central version repository.
3. Before the commit can be finalized, the patch submitted by the developer is subject to review by a pool of reviewers.
4. If the reviewers accept the patch, it is then passed to someone who verifies it by testing it, and finally commits the reviewed, verified code fragment to the version repository.

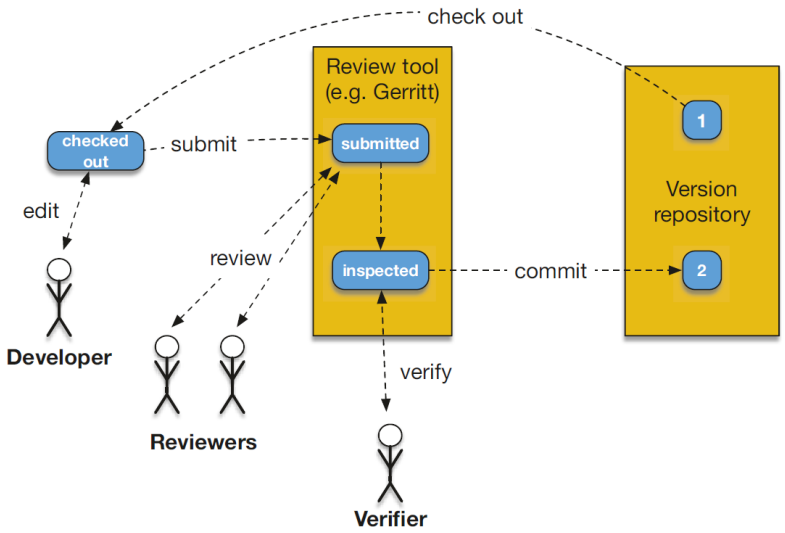

Figure 1. Typical modern code review work-flow using a code reviewing tool
Source: Software Quality Assurance: Consistency in the Face of Complexity and Change, 2017. p. 129

- **Pull-Based Development** – This development approach enforces modern code review and does not use code reviewing tools. This revolves around a mechanism known as a "pull request." Refer to *Figure 2* for the illustration of the flow of a pull-based development.

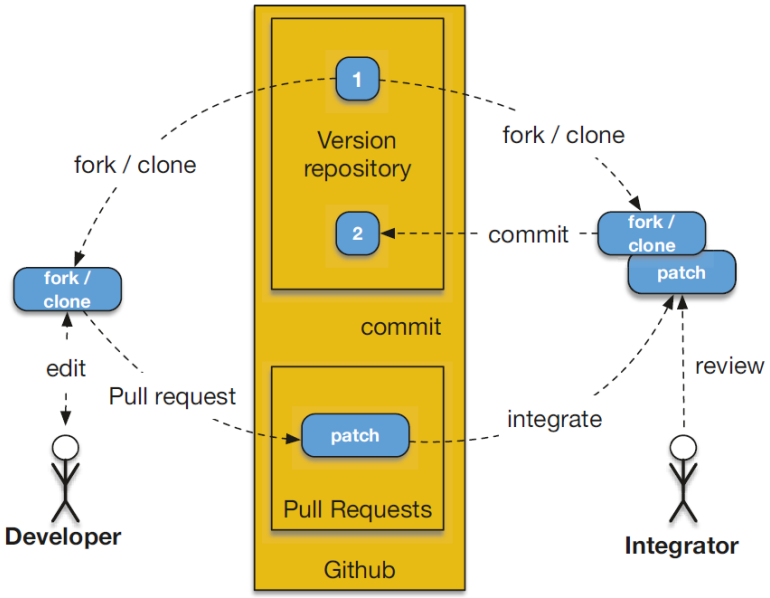

Figure 2. Pull-based development workflow
Source: Software Quality Assurance: Consistency in the Face of Complexity and Change, 2017. p. 130

1. A developer who does not have the privileges to push his/her changes directly to the central repository takes a clone of the code from the repository then makes changes.
2. The developer then submits his/her proposed changes to the repository as a pull request.
3. A developer with appropriate privileges (referred to as "integrator") examines the proposed changes, integrates them with their own clone of source code, and makes the commit.

REFERENCES:

Ming Zhu, Y. (2016). *Software reading techniques. Twenty techniques for more effective software review and inspection.* Retrieved from https://books.google.com.ph/books?id=MTCIDQAAQBAJ&printsec=frontcover#v=onepage&q&f=false

O'Regan, G. (2017). *Concise guide to software engineering. From fundamentals to application methods.* Cham, Switzerland: Springer International Publishing AG

Walkinshaw, N. (2017). *Software quality assurance: Consistency in the face of complexity and change.* Cham, Switzerland: Springer International Publishing AG.