Software review

A software review is "A process or meeting during which a software product is examined by a project personnel, managers, users, customers, user representatives, or other interested parties for comment or approval".

In this context, the term "software product" means "any technical document or partial document, produced as a deliverable of a software development activity", and may include documents such as contracts, project plans and budgets, requirements documents, specifications, designs, source code, user documentation, support and maintenance documentation, test plans, test specifications, standards, and any other type of specialist work product.

Varieties of software review

Software reviews may be divided into three categories:

Software peer reviews are conducted by the author of the work product, or by one or more colleagues of the author, to evaluate the technical content and/or quality of the work.

Software management reviews are conducted by management representatives to evaluate the status of work done and to make decisions regarding downstream activities.

Software audit reviews are conducted by personnel external to the software project, to evaluate compliance with specifications, standards, contractual agreements, or other criteria.

Different types of reviews

Code review is systematic examination (often as peer review) of computer source code.

Pair programming is a type of code review where two persons develop code together at the same workstation.

Inspection is a very formal type of peer review where the reviewers are following a well-defined process to find defects.

Walkthrough is a form of peer review where the author leads members of the development team and other interested parties through a software product and the participants ask questions and make comments about defects.

Technical review is a form of peer review in which a team of qualified personnel examines the suitability of the software product for its intended use and identifies discrepancies from specifications and standards.

Formal versus informal reviews

"Formality" identifies the degree to which an activity is governed by agreed (written) rules. Software review processes exist across a spectrum of formality, with relatively unstructured activities such as "buddy checking" towards one end of the spectrum, and more formal approaches such as walkthroughs, technical reviews, and software inspections, at the other. IEEE Std. 1028-1997 defines formal structures, roles, and processes for each of the last three ("formal peer reviews"), together with software audits.

Research studies tend to support the conclusion that formal reviews greatly outperform informal reviews in cost-effectiveness. Informal reviews may often be unnecessarily expensive (because of time-wasting through lack of focus), and frequently provide a sense of security which is quite unjustified by the relatively small number of real defects found and repaired.

IEEE 1028 generic process for formal reviews

IEEE Std 1028 defines a common set of activities for "formal" reviews (with some variations, especially for software audit). The sequence of activities is largely based on the software inspection process originally developed at IBM by Michael Fagan. Differing types of review may apply this structure with varying degrees of rigour, but all activities are mandatory for inspection:

0. [Entry evaluation]: The Review Leader uses a standard checklist of entry criteria to ensure that optimum conditions exist for a successful review.

1. Management preparation: Responsible management ensure that the review will be appropriately resourced with staff, time, materials, and tools, and will be conducted according to policies, standards, or other relevant criteria.

2. Planning the review: The Review Leader identifies or confirms the objectives of the review, organizes a team of Reviewers, and ensures that the team is equipped with all necessary resources for conducting the review.

3. Overview of review procedures: The Review Leader, or some other qualified person, ensures (at a meeting if necessary) that all Reviewers understand the review goals, the review procedures, the materials available to them, and the procedures for conducting the review.

4. [Individual] Preparation: The Reviewers individually prepare for group examination of the work under review, by examining it carefully for anomalies (potential defects), the nature of which will vary with the type of review and its goals.

5. [Group] Examination: The Reviewers meet at a planned time to pool the results of their preparation activity and arrive at a consensus regarding the status of the document (or activity) being reviewed.

6. Rework/follow-up: The Author of the work product (or other assigned person) undertakes whatever actions are necessary to repair defects or otherwise satisfy the requirements agreed to at the Examination meeting. The Review Leader verifies that all action items are closed.

7. [Exit evaluation]: The Review Leader verifies that all activities necessary for successful review have been accomplished, and that all outputs appropriate to the type of review have been finalised.

Value of reviews

The most obvious value of software reviews (especially formal reviews) is that they can identify issues earlier and more cheaply than they would be identified by testing or by field use (the defect detection process). The cost to find and fix a defect by a well-conducted review may be one or two orders of magnitude less than when the same defect is found by test execution or in the field.

A second, but ultimately more important, value of software reviews is that they can be used to train technical authors in the development of extremely low-defect documents, and also to identify and remove process inadequacies that encourage defects (the defect prevention process).

This is particularly the case for peer reviews if they are conducted early and often, on samples of work, rather than waiting until the work has been completed. Early and frequent reviews of small work samples can identify systematic errors in the Author's work processes, which can be corrected before further faulty work is done. This improvement in Author skills can dramatically reduce the time it takes to develop a high-quality technical document, and dramatically decrease the error-rate in using the document in downstream processes.

As a general principle, the earlier a technical document is produced, the greater will be the impact of its defects on any downstream activities and their work products. Accordingly, greatest value will accrue from early reviews of documents such as marketing plans, contracts, project plans and schedules, and requirements specifications. Researchers and practitioners have shown the effectiveness of reviewing process in finding bugs and security issues.

Pair programming

Pair programming is an agile software development technique in which two programmers work together at one work station. One types in code while the other reviews each line of code as it is typed in. The person typing is called the driver. The person reviewing the code is called the observer (or navigator). The two programmers switch roles frequently.

While reviewing, the observer also considers the strategic direction of the work, coming up with ideas for improvements and likely future problems to address. This frees the driver to focus all of his or her attention on the "tactical" aspects of completing the current task, using the observer as a safety net and guide.

Costs and benefits

Programmers working in pairs produce shorter programs, with better designs and fewer bugs, than programmers working alone. Studies have found reduction in defect rates of 15% to 50%, varying depending on programmer experience and task complexity. Pairs typically consider more design alternatives than programmers working solo, and arrive at simpler, more-maintainable designs, as well as catch design defects early. Pairs usually complete work faster than one programmer assigned to the same task. Pairs often find that seemingly "impossible" problems become easy or even quick, or at least possible to solve when they work together.

Even though coding is often completed faster than when one programmer works alone, total programmer time (number of programmers × time spent) increases. A manager needs to balance faster completion of the work and reduced testing and debugging time against the higher cost of coding. The relative weight of these factors can vary from project to project and task to task. The benefit is strongest on tasks that are not yet understood by the programmers, calling for more creativity, challenge, and sophistication. On simple tasks, which the pair already fully understands, pairing results in a net drop in productivity.

Knowledge passes between pair programmers as they work. They share knowledge of the specifics of the system, and they pick up programming techniques from each other. New hires quickly pick up the practices of the team and learn the specifics of the system. With "promiscuous pairing"—each programmer cycling through all the other programmers on the team rather than pairing only with one partner—knowledge of the system spreads throughout the whole team, reducing risk to management if one programmer leaves the team.

Pairing usually brings improved discipline and time management. Programmers are less likely to skip writing unit tests, spend time web-surfing or on personal email, or cut corners when they are working with a pair partner. The pair partner "keeps them honest". People are more reluctant to interrupt a pair than they are to interrupt someone working alone.

Additional benefits reported include increased morale and greater confidence in the correctness of the code.

Scientific studies

According to The Economist,

"Laurie Williams of the University of Utah in Salt Lake City has shown that paired programmers are only 15% slower than two independent individual programmers, but produce 15% fewer bugs. (N.B.: The original study showed that 'error-free' code went from 70% to 85%; it may be more intuitive to call this a 50% decrease of errors, from 30% to 15%.) Since testing and debugging are often many times more costly than initial programming, this is an impressive result."

The Williams et al. 2000 study showed an improvement in correctness of around 15% and 20 to 40% decrease in time, but between a 15 and 60% increase in effort. Williams et al. 2000 also cites an earlier study (Nosek 1998) which also had a 40% decrease in time for a 60% increase in effort.

A study (Lui 2006) presents a rigorous scientific experiment in which novice–novice pairs against novice solos experience significantly greater productivity gains than expert–expert pairs against expert solos.

A larger recent study (Arisholm et al. 2007) had 48% increase in correctness for complex systems, but no significant difference in time, whilst simple systems had 20% decrease in time, but no significant difference in correctness. Overall there was no general reduction in time or increase in correctness, but an overall 84% increase in effort.

Lui, Chan, & Nosek (2008) shows that pair programming outperforms for design tasks.

A full-scale meta-analysis of pair programming experimental studies, from before or during 2007, (Hannay et al. 2009) confirms "that you cannot expect faster and better and cheaper". Higher quality for complex tasks costs higher effort, reduced duration for simpler tasks comes with noticeably lower quality - the meta-analysis "suggests that pair programming is not uniformly beneficial or effective".

Variants

Remote pair programming

Remote pair programming, also known as virtual pair programming or distributed pair programming, is pair programming where the two programmers are in different locations, working via a collaborative real-time editor, shared desktop, or a remote pair programming IDE plugin. Remote pairing introduces difficulties not present in face-to-face pairing, such as extra delays for coordination, depending more on "heavyweight" task-tracking tools instead of "lightweight" ones like index cards, and loss of non-verbal communication resulting in confusion and conflicts over such things as who "has the keyboard".

Numerous tools, such as Eclipse plug-ins are available to support remote pairing. Some teams have tried VNC and RealVNC with each programmer using their own computer. Others use the multi-display mode (-x) of the text-based GNU screen. Apple Inc. OSX has a built-in Screen Sharing application.

Ping pong pair programming

In ping pong pair programming, the observer writes a failing unit test, the driver modifies the code to pass the test, the observer writes a new unit test, and so on. This loop continues as long as the observer is able to write failing unit tests.

Code review

Code review is systematic examination (often as peer review) of computer source code. It is intended to find and fix mistakes overlooked in the initial development phase, improving both the overall quality of software and the developers' skills. Reviews are done in various forms such as pair programming, informal walkthroughs, and formal inspections.

Introduction

Code reviews can often find and remove common vulnerabilities such as format string exploits, race conditions, memory leaks and buffer overflows, thereby improving software security. Online software repositories based on Subversion (with Redmine or Trac), Mercurial, Git or others allow groups of individuals to collaboratively review code. Additionally, specific tools for collaborative code review can facilitate the code review process.

Automated code reviewing software lessens the task of reviewing large chunks of code on the developer by systematically checking source code for known vulnerabilities.

Capers Jones' ongoing analysis of over 12,000 software development projects showed that the latent defect discovery rate of formal inspection is in the 60-65% range. For informal inspection, the figure is less than 50%. The latent defect discovery rate for most forms of testing is about 30%.

Typical code review rates are about 150 lines of code per hour. Inspecting and reviewing more than a few hundred lines of code per hour for critical software (such as safety critical embedded software) may be too fast to find errors. Industry data indicate that code review can accomplish at most an 85% defect removal rate with an average rate of about 65%.

Types

Code review practices fall into three main categories: pair programming, formal code review and lightweight code review.

Formal code review, such as a Fagan inspection, involves a careful and detailed process with multiple participants and multiple phases. Formal code reviews are the traditional method of review, in which software developers attend a series of meetings and review code line by line, usually using printed copies of the material. Formal inspections are extremely thorough and have been proven effective at finding defects in the code under review.

Lightweight code review typically requires less overhead than formal code inspections, though it can be equally effective when done properly. Lightweight reviews are often conducted as part of the normal development process:

Over-the-shoulder – One developer looks over the author's shoulder as the latter walks through the code.

Email pass-around – Source code management system emails code to reviewers automatically after checkin is made.

Pair Programming – Two authors develop code together at the same workstation, such is common in Extreme Programming.

Tool-assisted code review – Authors and reviewers use specialized tools designed for peer code review.

Some of these may also be labeled a "Walkthrough" (informal) or "Critique" (fast and informal).

Many teams that eschew traditional, formal code review use one of the above forms of lightweight review as part of their normal development process. A code review case study published in the book Best Kept Secrets of Peer Code Review found that lightweight reviews uncovered as many bugs as formal reviews, but were faster and more cost-effective.

Criticism

Historically, formal code reviews have required a considerable investment in preparation for the review event and execution time.

Some believe that skillful, disciplined use of a number of other development practices can result in similarly high latent defect discovery/avoidance rates. For example, the practice of pair programming mandated in Extreme Programming (XP) likely results in defect discovery rates of 50% noted generally for informal code reviews. Further, XP proponents might argue, layering additional XP practices, such as refactoring and test-driven development will result in latent defect levels rivaling those achievable with more traditional approaches, without the investment.

Use of code analysis tools can support this activity. Especially tools that work in the IDE as they provide direct feedback to developers of coding standard compliance.

Deskchecks

A deskcheck is a simple review in which the author of a work product distributes it to one or more reviewers. In a deskcheck, the author sends a copy of the work product to selected project team members. The team members read it, and then write up defects and comments to send back to the author. Unlike an inspection, a deskcheck does not produce written logs which can be archived with the document for later reference. There is no follow-up meeting or approval process. It is simply a way for one team member to check another's work. Deskchecks are not formal reviews (where "formal" simply means that it generates a written work product which meets a certain standard and is archived with the rest of the project documentation); there is no standard for the results of the deskcheck. The reviewers simply review the work product and return the results. There is no moderator, and there is not necessarily any consensus generated.

There are times when a full inspection is neither necessary nor useful. Some work products do not benefit enough to warrant the attention of an entire inspection team because they do not need consensus or approval. In these cases, the author simply needs input from others to prevent defects, but does not require that they approve the document. In these cases, the deskcheck is a useful review practice.

The illustration below contains an example of comments from a deskcheck which was used by a tester to find defects in an automation script. In this case, the entire review was performed via e-mail: the author mailed the script to the reviewer, and the reviewer read it and e-mailed the comments back to the author. These comments are much simpler than the inspection log in Figure 5-1. In an inspection, each log entry must either resolve a defect or indicate that it is an open issue which must be resolved. Deskcheck comments can simply point out issues or raise questions without having to supply solutions or promise a resolution. There was no follow-up or approval, and the reviewer had no more contact with this script.

Deskchecks can be used as predecessors to inspections. In many cases, having an author of a work product pass his work to a peer for an informal review will significantly reduce the amount of effort involved in the inspection. Many defects can be caught by a single person reviewing a document. Approval and consensus is built later on during the inspection meeting; this is simply a way of saving effort. After a deskcheck, many authors will feel much more comfortable sending their document into an inspection—it will often help the author to be more objective and to take the inspection comments less personally.

Software inspection

Inspection in software engineering, refers to peer review of any work product by trained individuals who look for defects using a well defined process. An inspection might also be referred to as a Fagan inspection after Michael Fagan, the creator of a very popular software inspection process.

Introduction

An inspection is one of the most common sorts of review practices found in software projects. The goal of the inspection is for all of the inspectors to reach consensus on a work product and approve it for use in the project. Commonly inspected work products include software requirements specifications and test plans. In an inspection, a work product is selected for review and a team is gathered for an inspection meeting to review the work product. A moderator is chosen to moderate the meeting. Each inspector prepares for the meeting by reading the work product and noting each defect. The goal of the inspection is to identify defects. In an inspection, a defect is any part of the work product that will keep an inspector from approving it. For example, if the team is inspecting a software requirements specification, each defect will be text in the document which an inspector disagrees with.

The process

The inspection process was developed by Michael Fagan in the mid-1970s and it has later been extended and modified.

The process should have entry criteria that determine if the inspection process is ready to begin. This prevents unfinished work products from entering the inspection process. The entry criteria might be a checklist including items such as "The document has been spell-checked".

The stages in the inspections process are: Planning, Overview meeting, Preparation, Inspection meeting, Rework and Follow-up. The Preparation, Inspection meeting and Rework stages might be iterated.

Planning: The inspection is planned by the moderator.

Overview meeting: The author describes the background of the work product.

Preparation: Each inspector examines the work product to identify possible defects.

Inspection meeting: During this meeting the reader reads through the work product, part by part and the inspectors point out the defects for every part.

Rework: The author makes changes to the work product according to the action plans from the inspection meeting.

Follow-up: The changes by the author are checked to make sure everything is correct.

The process is ended by the moderator when it satisfies some predefined exit criteria.

Inspection roles

During an inspection the following roles are used.

Author: The person who created the work product being inspected.

Moderator: This is the leader of the inspection. The moderator plans the inspection and coordinates it.

Reader: The person reading through the documents, one item at a time. The other inspectors then point out defects.

Recorder/Scribe: The person that documents the defects that are found during the inspection.

Inspector: The person that examines the work product to identify possible defects.

Related inspection types

Code review

A code review can be done as a special kind of inspection in which the team examines a sample of code and fixes any defects in it. In a code review, a defect is a block of code which does not properly implement its requirements, which does not function as the programmer intended, or which is not incorrect but could be improved (for example, it could be made more readable or its performance could be improved). In addition to helping teams find and fix bugs, code reviews are useful for both cross-training programmers on the code being reviewed and for helping junior developers learn new programming techniques.

Peer Reviews

Peer Reviews are considered an industry best-practice for detecting software defects early and learning about software artifacts. Peer Reviews are composed of software walkthroughs and software inspections and are integral to software product engineering activities. A collection of coordinated knowledge, skills, and behaviors facilitates the best possible practice of Peer Reviews. The elements of Peer Reviews include the structured review process, standard of excellence product checklists, defined roles of participants, and the forms and reports.

Software inspections are the most rigorous form of Peer Reviews and fully utilize these elements in detecting defects. Software walkthroughs draw selectively upon the elements in assisting the producer to obtain the deepest understanding of an artifact and reaching a consensus among participants. Measured results reveal that Peer Reviews produce an attractive return on investment obtained through accelerated learning and early defect detection. For best results, Peer Reviews are rolled out within an organization through a defined program of preparing a policy and procedure, training practitioners and managers, defining measurements and populating a database structure, and sustaining the roll out infrastructure.

Software walkthrough

In software engineering, a walkthrough or walk-through is a form of software peer review "in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems".

"Software product" normally refers to some kind of technical document. As indicated by the IEEE definition, this might be a software design document or program source code, but use cases, business process definitions, test case specifications, and a variety of other technical documentation may also be walked through.

A walkthrough differs from software technical reviews in its openness of structure and its objective of familiarization. It differs from software inspection in its ability to suggest direct alterations to the product reviewed, its lack of a direct focus on training and process improvement, and its omission of process and product measurement.

Process

A walkthrough may be quite informal, or may follow the process detailed in IEEE 1028 and outlined in the article on software reviews.

Objectives and participants

In general, a walkthrough has one or two broad objectives: to gain feedback about the technical quality or content of the document; and/or to familiarize the audience with the content.

A walkthrough is normally organized and directed by the author of the technical document. Any combination of interested or technically qualified personnel (from within or outside the project) may be included as seems appropriate.

IEEE 1028 recommends three specialist roles in a walkthrough:

The Author, who presents the software product in step-by-step manner at the walk-through meeting, and is probably responsible for completing most action items;

The Walkthrough Leader, who conducts the walkthrough, handles administrative tasks, and ensures orderly conduct (and who is often the Author); and

The Recorder, who notes all anomalies (potential defects), decisions, and action items identified during the walkthrough meetings.

Software technical review

A software technical review is a form of peer review in which "a team of qualified personnel ... examines the suitability of the software product for its intended use and identifies discrepancies from specifications and standards. Technical reviews may also provide recommendations of alternatives and examination of various alternatives".

"Software product" normally refers to some kind of technical document. This might be a software design document or program source code, but use cases, business process definitions, test case specifications, and a variety of other technical documentation, may also be subject to technical review.

Technical review differs from software walkthroughs in its specific focus on the technical quality of the product reviewed. It differs from software inspection in its ability to suggest direct alterations to the product reviewed, and its lack of a direct focus on training and process improvement.

The term formal technical review is sometimes used to mean a software inspection.

Objectives and participants

The purpose of a technical review is to arrive at a technically superior version of the work product reviewed, whether by correction of defects or by recommendation or introduction of alternative approaches. While the latter aspect may offer facilities that software inspection lacks, there may be a penalty in time lost to technical discussions or disputes which may be beyond the capacity of some participants.

IEEE 1028 recommends the inclusion of participants to fill the following roles:

The Decision Maker (the person for whom the technical review is conducted) determines if the review objectives have been met.

The Review Leader is responsible for performing administrative tasks relative to the review, ensuring orderly conduct, and ensuring that the review meets its objectives.

The Recorder documents anomalies, action items, decisions, and recommendations made by the review team.

Technical staff are active participants in the review and evaluation of the software product.

Management staff may participate for the purpose of identifying issues that require management resolution.

Customer or user representatives may fill roles determined by the Review Leader prior to the review.

A single participant may fill more than one role, as appropriate.