CHAPTER 13

Reviews and Testing

A review is an evaluation of a life-cycle work product(s) or project status to determine if there are any deviations from planned results and to recommend improvement. Peer reviews may be thought of as *human-based* testing as opposed to *computer-based* testing. An anomaly is any condition that deviates from expectations based on requirements specifications, design documents, standards, plans, and so on, or from someone's experiences. These anomalies are most often called *defects*.

Why Should We Conduct Reviews?

Reviews are conducted for these reasons:

- Detect defects.
- Remove defects as close to the point of insertion as possible.
- Determine product progress/status.
- Identify potential improvements.
- Produce technical work of a more uniform and predictable quality.
- Gain ownership by the project team.
- Assist employees with cross-training.
- Reduce costs to build and maintain better products.
- Reduce development time.
- Reduce testing cost and time.
- Reduce total system maintenance cost dramatically (as much as 10 to 1 according to recent statistics).

The sooner a defect is found, the cheaper it is to fix. Studies over the years have indicated that if a defect can be detected and fixed in the requirements phase, it might cost $1. If that defect moves through subsequent life-cycle phases until systems test, the cost to find and fix the defect might be as much as $500. If the defect continues further and reaches the customer before it is detected, the cost to fix may be nearer $1,000 or higher. The reasoning is simple if you look at the rework costs. If the defect makes it to the customer and comes back to the supplier organization to be resolved, the cost of rework includes the time required to analyze the problem, the time to fix the problem, the time to conduct a review, the time to conduct unit testing, the time to conduct regression testing, and the time to get the fixed system

back to the customer. This does not include the time supplied by the support groups such as quality assurance and configuration management that is necessary to support the defect finding and fixing process.

The ultimate objective of a peer review is to minimize the number of defects being passed along from one life-cycle work product to another and from one life-cycle phase to another by finding and fixing the defects at the point in the life cycle in which they were created. When one looks to peer reviews as a technique to support quality management [1], "…the inevitability of defect is not the true source of concern for quality management, but rather at what point in the product life cycle these defects are detected and corrected. If we recognize that development is an error-prone process, and gear our quality program to finding and correcting defects early in the process, we reduce the cost of defect correction and increase productivity—while we improve quality."

Reviews as a Management Control Tool

During any project, management requires a means of assessing and measuring progress. Questions such as these might be asked:

- Are we ahead or behind where we expected to be?
- Will we complete the work as planned?
- Do we require more computer or human resources to meet the planned schedule?
- Is the required functionality being implemented?
- What risks are we taking based on the project information that we have today?

We are accustomed to writing status reports and making project progress presentations. But how do successive levels of management and ultimately the customer know just what progress has been made? "True" progress cannot be measured by counting the completion of tasks unless there is a reliable way of measuring the quality of the work performed and knowing that it would not have to be redone or changed later (rework!). In other words, unless techniques such as reviews and testing are being applied to the life-cycle work products and the results analyzed, management progress reports could represent little more than wishful thinking. Figure 13.1 shows a representative sample of the different types of reviews and audits that can be applied for achieving quality purposes to support project progress reporting.

Management or Progress Reviews

A management review is a formal evaluation of a project-level plan or project *progress* relative to that plan by a designated review team. Management reviews are normally conducted for these reasons:

- Track progress according to plan, based on an evaluation of the product development status.

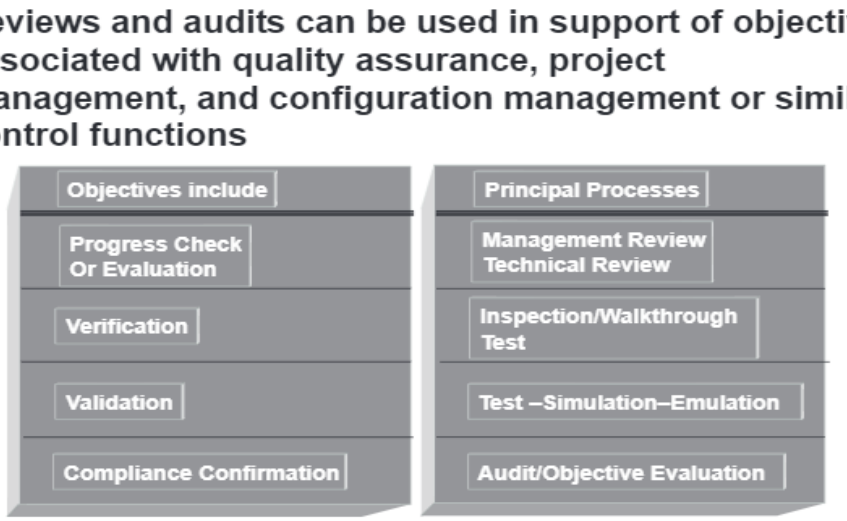

Figure 13.1 Principal processes for achieving quality objectives.

- Identify inconsistencies with and deviations from plans.
- Confirm requirements and their system allocation.
- Monitor risks.
- Evaluate the effectiveness of management approaches used to achieve fitness for purpose.
- Take corrective action and change project direction or identify the need for alternative planning.
- Maintain control of the project through adequate allocation and reallocation of resources.
- Change the scope of the project.

Three basic types of management reviews or progress reviews can be found in the CMMI®: project manager reviews, milestone reviews, and senior management oversight reviews. Let us take a brief look these types of management reviews.

Project Manager Review

The project manager review normally translates into the weekly project meeting that is called by the project manager. At a minimum, the project manager and development staff are in attendance. Representatives from systems engineering, QA, CM, and testing should also be in attendance. Items discussed at the weekly project meeting include tracking of the standard issues of technical activities, cost, and schedule performance against the plan. Staffing concerns may also be brought up. Resources planned for and consumed are reviewed. Risks are identified, prioritized, and contingency plans discussed. Necessary commitment changes are acknowledged, documented, and approved. Stakeholder involvement is discussed to ensure that the relevant stakeholders are involved at the time it was deemed necessary and the project is receiving the necessary results from their involvement. Corrective actions may be necessary and all planning documents are updated as necessary.

Milestone Reviews

Formal milestone dates are documented in the project management plan, tracked, and reviewed. Milestones normally represent meaningful points in the project's schedule. This might signify the end of a phase or a major activity. It might be linked to customer involvement and even contract payments such as in the defense world of conducting preliminary design reviews (PDRs) and critical design reviews (CDRs).

Typical milestone reviews include examination of technical progress, planned engineering activities, commitments, and plans. They normally address project and other engineering discipline risks. Action items from previous meetings are brought up to determine progress and closure and new action items are recorded, assigned, and reviewed as necessary. All decisions are documented with corresponding actions and resolutions.

Senior Management Oversight Reviews

Watts Humphrey's original intent for senior management oversight reviews was to provide higher level management with the appropriate visibility into the processes being used on the projects.

When discussions around senior management oversight reviews come up, many individuals and organizations believe that this strictly refers to a meeting with the senior management team to discuss the project progress including technical, cost, staffing, schedule, performance, risks, and other necessary functions. Clearly these senior management team progress reviews are necessary. But Mr. Humphrey wanted the senior management team to receive more than the normal project status. He wanted the senior managers to understand what processes were being followed on the projects, whether they efficient and effective, and whether they helping the project team members produce the required product and service quality.

It is the senior management team that owns the organization's process, and the senior management team is the only group that can reallocate resources to support increased process improvement and quality needs. Only the senior management team can authorize mass training and award the necessary authority to carry out these process improvement and quality management activities.

Management oversight did not and does not indicate that the senior management team missed something important. It was an indication that they could see over the projects, understand the processes and work environment provided to them, and decide on the priorities to support the improvement of the processes in order to assist the projects in accomplishing their business goals.

Peer Reviews

Although the SEI's Capability Maturity Model® defines peer reviews and refers to inspections and structured walkthroughs as examples of peer reviews at CMMI® ML 3, a number of peer reviews types can be used for different objectives and expected results including:

- Buddy checks;

- Circulation reviews;
- Technical reviews;
- Inspections;
- Walkthroughs;
- Structured walkthroughs.

These peer review types will be described in an overview fashion in this chapter but first it might be helpful to provide a historical look at how use of the term *peer reviews* came about in the CMM® and now the CMMI®.

Walkthroughs and structured walkthroughs had been introduced to the industry by leading figures such as Gerald Weinberg[1] and Ed Yourdon[2] in the late 1960s and early 1970s. In 1972, Michael Fagan[3] of IBM developed a more formal approach to defect detection for early-phase life-cycle work products called inspections. Inspections were a more formal approach to conducting reviews with an absolute focus on detecting major defects and eliminating them if possible.

As Mr. Watts Humphrey worked for IBM for more than 30 years, he naturally brought with him that notion of conducting a review on selected life-cycle work

1. For more than 45 years, Jerry Weinberg has worked on transforming software organizations, particularly emphasizing the interaction of technical and human issues. After spending between 1956 and 1969 as software developer, researcher, teacher, and designer of software curricula at IBM, he formed the consulting firm of Weinberg & Weinberg to help software engineering organizations manage the change process in a more fully human way. In June 1997, he was inducted into the Computer Hall of Fame, along with such notables as Charles Babbage, Semour Cray, James Martin, Grace Hopper, Gerald Weinberg, and Bill Gates. Jerry is author or coauthor of several hundred articles and more than 30 books. His earliest published work was on operating systems and programming languages, but the 1971 publication of *The Psychology of Computer Programming* is considered by many the beginning of the study of software engineering as human behavior. His subsequent works have been an elaboration of many of the software engineering topics raised in that book, through all phases of the software life cycle, including defining problems and requirements, analysis and design, testing and measurement, as well as management. He cowrote the *Handbook of Walkthroughs, Inspections, and Technical Reviews* with Daniel P. Freedman in 1982. (*From:* http://www.geraldmweinberg.com/BIOStuff/EachBIO/bio.Jerry.html.)
2. Ed Yourdon is an internationally recognized computer consultant who specializes in project management, software engineering methodologies, and Web 2.0 development. According to the December 1999 issue of *Crosstalk: The Journal of Defense Software Engineering,* Ed Yourdon is one of the 10 most influential men and women in the software field. In June 1997, he was inducted into the Computer Hall of Fame, along with such notables as Charles Babbage, Semour Cray, James Martin, Grace Hopper, Gerald Weinberg, and Bill Gates. Yourdon is widely known as the lead developer of the structured analysis/design methods of the 1970s. He was a codeveloper of the Yourdon/Whitehead method of object-oriented (OO) analysis/design and the popular Coad/Yourdon OO methodology. He wrote his book on *Structured Walkthroughs,* 4th ed., in 1989. Mr. Yourdon began his career in the computer industry at Digital Equipment Company more than 35 years ago. Mr. Yourdon is the author/coauthor of more than 25 computer books, as well as more than 525 technical articles since 1967. (*From:* http://www.cutter.com/meet-our-experts/ eybio.html.)
3. Michael Fagan is the founder and CEO of Michael Fagan Associates and the creator of Fagan Inspections and the Fagan Defect-Free Process™. As a product development manager at IBM, Michael created the Fagan Inspection Process for use on his own projects. Over the years, this has been enhanced and expanded into the Fagan Defect-Free Process™, incorporating formal process definition, and reinforcing the continuous process improvement aspect of the inspection process. The methodology developed by Michael Fagan is credited with dramatically reducing the number of defects in software and hardware products, increasing the feature content per release, shortening cycle time, increasing customer satisfaction, improving development processes, accelerating SEI/CMM® maturity in organizations, and significantly reducing costs!

   Michael Fagan has 20 years of experience in IBM as a line manager of software development, engineering development, and manufacturing. In addition, he was manager of programming methodology for IBM's DP Product Group (Worldwide); the first software senior technical staff member in IBM's T. J. Watson Research Laboratory; a member of the Corporate Technology Staff; and one of the founding members of the IBM Quality Institute. (*From:* http://www.mfagan.com/bio_frame.htm.)

products, especially in the early phases of the life cycle. As fate would have it, I wrote 50% of IEEE Standard 1028, *Reviews and Audits*, which was approved before I joined the SEI in 1988. I wrote the section on walkthroughs and Bob Ebenau wrote the section on inspections. As the CMM® started evolving, Mr. Humphrey insisted that inspections be required at Maturity Level 3. But in the intervening time before the CMM® was made public, many debates were held over what these required reviews would be called. Not all people appreciated the IBM way as Watts did, so in the end, the term *peer reviews* was used. However, even in the initial days of the CMM®, the intent was to apply the formal approach to life-cycle work products in the manner defined by Michael Fagan.

Let us now examine each of the peer reviews types listed earlier in an overview fashion.

Buddy Check

A buddy check is normally thought of as an informal verification technique in which the life-cycle work product is examined by the author and one other person. The buddy check operates on basically the same principles as a walkthrough. The "buddy" may or may not prereview the life-cycle work product prior to the peer review. The author walks the "buddy" through the life-cycle work product and describes what is intended in each section. The "buddy" offers comments for improvement on technical correctness, style, order of presentation, clarity, and understandability. Most importantly the buddy decides on the "fit" to the described intent by the author. The author may or may not accept the buddy's suggestions for improvement.

The objectives of buddy checks include:

- Improve the life-cycle work product;
- Consider alternative implementations;
- Exchange techniques and style variations;
- Point out problems with clarity and understandability;
- Allow the author to look at the life-cycle work product from a different "angle" or point of view;
- Mentoring of others in the concepts embedded in the life-cycle work product.

The second-to-last objective is often the greatest benefit of using the buddy check. It is not uncommon for the author to discover defects while describing the intent of the work product to the buddy.

There is another reason—a very powerful and very human reason—to incorporate buddy checks into a person's, project's, or organization's set of peer review definitions. When we ask our colleagues to give some of their very valuable time to review a work product of ours to help make it better, it is simply common courtesy for the author to make his/her life-cycle product as good as possible before giving it to a colleague or peer. No one wants to be asked to review a life-cycle work product that is incomplete or laced with small but annoying defects such as spelling errors or incorrect formatting, which distracts the reviewer from the real task of reviewing to detect defects.

Circulation Review

Circulation reviews take on attributes of both buddy checks and walkthroughs. Circulation reviews can be informal or follow strict rules. The life-cycle work product is circulated to each reviewer who reviews it and either attaches comments, questions, and recommendations directly on the life-cycle work product or places them into a separate document. Circulation reviews are dependent on the goodwill of the reviewer. Normally, the author has the authority to accept the reviewers' comments and suggestions or ignore them.

Circulation reviews are especially useful in these situations:

- Reviewers are geographically separated and face-to-face or teleconferencing is not possible.
- Individuals who are asked to be reviewers are willing to "work in" the review but not take the time out of their schedule to participate in a more formal review.
- The author is looking for a large cross section of opinions regardless of background or experience.
- A large population of stakeholders must be satisfied.
- The time during which the review must be completed is not a constraint, but the author needs as detailed a look at the life-cycle work product as possible.

The objectives of circulation reviews include:

- Improve the life-cycle work product;
- Consider alternative implementations;
- Point out problems with clarity and understandability;
- Point out areas of concern and offer comments and suggestions;
- Gain consensus from a large population of reviewers;
- Gain input from valuable contributors who cannot be present for a face-to-face review.

Technical Review

A technical review is a formal team evaluation of a life-cycle work product to:

- Identify any discrepancies from specifications and standards;
- Determine its suitability for use;
- Provide recommendations after the examination of various alternatives.

Technical reviews may be held at the request of functional management, project management, quality management, systems engineering, software engineering, or hardware engineering. Technical reviews may be required to evaluate the impacts of hardware anomalies or deficiencies of the software.

The objectives of technical reviews are to ensure that:

- The life-cycle work product conforms to its specifications;

- The development or maintenance of the life-cycle work product is being done according to plans, standards, and guidelines applicable to the project;
- Changes to the life-cycle work product are properly implemented and affect only those areas of the system identified by the change specification.

Technical reviews are typically used for the classic design reviews such as PDRs, CDRs, and test readiness reviews (TRRs). Technical reviews are also appropriate for life-cycle work products such as architectural specifications.

Next we describe inspections, then walkthroughs, and finish up with a discussion of structured walkthroughs and a comparison to inspections.

Inspections

An inspection is a formal verification technique in which life-cycle work products are examined in detail by a group of peers for the explicit purpose of detecting and identifying defects. The process is led by a moderator or facilitator or inspection leader who is not the author and is impartial to the life-cycle work product under review. The author is not allowed to act as the moderator. Written action on all major defects is mandatory. Rework due to corrections of major defects is formally verified. Defect data is systematically collected and stored in an inspection database. This defect data is analyzed to improve the product, the process, and the effectiveness of the inspection process. Individuals holding management positions over any member of the inspection team are normally not permitted to participate in the inspection.

The objective of an inspection is to detect and identify life-cycle work product defects in a rigorous, formal, peer examination that does the following:

- Verifies that the life-cycle work product satisfies both its specification and preceding intermediate work products.
- Verifies that the life-cycle work product conforms to applicable standards.
- Identifies real or potential deviations from standards and specifications.
- Collects engineering data (i.e., defect and effort data).
- Does not examine alternatives or stylistic issues.

Inspections are most effective if they are an integral part of the development process.

Inspector's Responsibilities

Inspectors/reviewers should be chosen to:

- Identify and describe the major and minor defects found in the product or product component (checking);
- Be chosen to represent different points of view (requirements, design, development, test, independent test, project management, quality management, and so on);

- Be assigned specific review topics to ensure effective coverage:
  - Conformance with a specific standard;
  - Overall coherence.

Inspection Minimum Entry Criteria

An inspection shall not be conducted until all of the following events have occurred:

- The product or product component is complete and conforms to project standards for content and format.
- Any automated error-detecting tools required for the inspection are available.
- Required supporting documentation is available.
- For a reinspection, all items noted on the defect list are resolved.
- The author is available.
- A trained moderator and sufficient number of skilled inspectors are available.

Inspection Preparedness

The inspection leader or moderator is required verify that the inspectors are prepared for the inspection. The inspection leader may reschedule the inspection (logging) if the inspectors are not adequately prepared.

Inspection Defect List

The recorder is responsible for entering each major and minor defect, location, description, and classification on the defect list. The author should be ready to answer specific questions and contribute personally to the defect detection process. If there is disagreement about a defect, the potential defect shall be logged and marked for resolution at the end of the meeting. The moderator is responsible for having the defect list reviewed with the team to ensure it completeness and accuracy.

Walkthroughs

Walkthroughs were designed to be a less formal verification technique in which life-cycle work products are examined by a group of peers for the purpose of finding defects, omissions, and contradictions. The walkthrough is normally led by the author or the producer of the material being reviewed. As the walkthrough progresses, errors, suggested changes, and improvement suggestions are noted and documented. The consolidated notes are taken by the author for review and revision as the author sees fit.

The objectives of walkthroughs in addition to detecting defects are to:

- Improve the life-cycle work product;
- Consider alternative implementations;

- Point out efficiency and readability problems, or modularity problems if the life-cycle work product is code;
- Exchange techniques and style variations;
- Educate the participants.

The walkthrough leader/author normally distributes the product or product component and conducts the walkthrough meeting. Each team member prepares by examining the life-cycle product in advance and prepares a list of items for discussion during the meeting itself. The author presents an overview of the life-cycle product under review, facilitates the general discussion, and serially presents the life-cycle work product in detail, accepting team members' comments as the author proceeds.

Walkthroughs should provide data for the analysis of:

- The quality of the product;
- The effectiveness of the acquisition, supply, development, operation, and maintenance processes;
- Efficiency of the walkthrough itself.

Structured Walkthroughs

Structured walkthroughs are more closely aligned with inspections than the informal walkthrough. One can think of doing a walkthrough while using many of the inspection requirements such as:

- Ensuring that each participant has individually prepared for the structured walkthrough and is prepared with a listing of major and minor defects before coming to the face-to-face part of the structured walkthrough;
- Having roles and responsibilities preplanned such as recorder, reviewer, and any special roles;
- Controlling the checking and logging rate;
- Concentrating on major defects;
- Restricting the structured walkthrough to presenting and consolidating a list of major and minor defects and not allowing solution discussions;
- Collecting the defect data and analyzing it for trends and process improvement.

In fact, structured walkthroughs can be made to be very close in nature to the demands of inspections. The one significant difference is that the structured walkthrough is led by the author. There are possible positive and negative reasons for the author to lead the structured walkthrough. The positive reason is that the author understands the intent of his/her work and can provide that guidance to the reviewers to help them gain greater insight and possibly find more major defects. One negative aspect of the author leading the structured walkthrough is that he/she can lead the reviewers around or away from areas that might contain significant major defects.

I have found reasons to use both inspections and structured walkthroughs for artifacts that are critical, complex, or that have a history of being found with major defects. Although many arguments have been put forth over the years, historical data collected by IBM, TRW, and MITRE Corporation indicates formal inspections are the most efficient form of defect removal.

Steps in the Inspections Process

The inspection process is shown in Figure 13.2. An abbreviated description of each inspection block is offered here.

Entry

The inspection process is only entered when a specified set of entry criteria have been met:

- Is the author of the life-cycle work product ready for inspection?
- Is the life-cycle work product ready for inspection?
- Are all of the source and kin documents identified and available against which the life-cycle work product will be evaluated?
- Are the "exit" criteria for this inspection documented and available?

Planning

The moderator is responsible for leading the inspection planning effort, which includes these tasks:

- Obtain and distribute supporting documents:
  - Life-cycle work product;

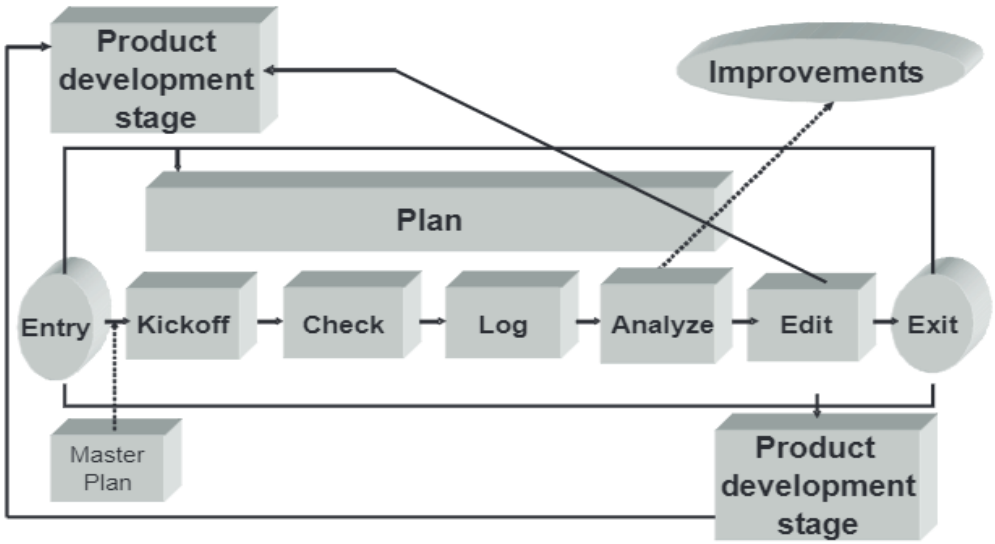

Figure 13.2 Steps in the inspection process.

    - Source/rule/kin documentation;
    - Checklists.
- Establish optimum checking rate.
- Decide format for life-cycle work product to be inspected (full, sample, chunk).
- Select potential review team members.
- Determine inspection roles, including any special roles.
- Set logging meeting time and date.
- Schedule kickoff meeting, including meeting room, and notifies applicable parties.
- Ensure that all necessary forms are available.

Kickoff

The moderator conducts a "kickoff" meeting prior to checking by completing these tasks:

- Set ground rules or expectations for the inspection.
- Review the inspection process as a refresher to those who have previously participated in inspections or to those who are participating for the first time.
- Get agreement with the team goal from all of the reviewers:
    - Finding the most major defects in the time given (efficiency);
    - Finding the most major defects (effectiveness);
    - Finding the most defects (major and minor).
- Get agreement on the strategy to meet the team goal:
    - Full life-cycle work product inspection;
    - Systematic sampling of life-cycle work product;
    - Chunking.
- Assign standard roles and special roles.
- Review past examples of defects for that type of life-cycle work product from historical data.
- Agree to approach to logging.
- Provide educational overview as needed.
- Make sure that all participants are focused on the correct life-cycle work product and the appropriate rules and checklists.
- Ensure that the reviewers have all necessary forms to use during the checking.

Checking

Each inspection team member inspects the life-cycle work product and logs possible major and minor defects. Note that checking is completed prior to the logging meeting. Reviewers must be given the necessary source/rule/kin documents to review the life-cycle work product against.

Logging

This is the part of the inspection where the reviewers meet face to face to log and consolidate the defects found during their individual checking period. The inspection team concentrates on reporting orally and logging, in writing, major issues, at a rate of at least one per minute. The logging meeting should last a maximum of 2 hours; 1½ hours is an excellent target duration.

Analyzing

The analysis subprocess of the inspection process is a phase I created by separating it from the logging phase. It was done more for psychological reasons, but in many cultures, it works and makes reviewers more comfortable. The author reviews the recorded major and minor defects and answers these questions:

- Are all recorded defects understandable?
- Are there any disagreements with the defect as a defect that can be resolved with a short statement?
- Can the inspection team come to consensus with the author within an agreed-on short time period?

Process Improvement Brainstorming

Immediately, if possible, or shortly after the logging and analysis meeting, time may be allotted to brainstorm the defect causes and process improvements to prevent these causes. Improvements may be made to the inspection process, the source and kin documents, and/or the checklists.

Edit

Some written action, including corrective action, must be carried out by the author or an editor.

- Some written action must be taken on all logged issues—if necessary by sending change requests to other authors.
- The author may request a solution brainstorming session with the other reviewers or anyone else that he/she believes can help to determine the best solution for the documented defects.
- The moderator may review that the defects have been corrected and spot-check the solution on a sampling basis.

Exit

The moderator determines whether the formal exit criteria have been met before signing off on completion of the inspection process. The moderator must determine the following:

- Is the life-cycle work product ready for promotion to the next phase or activity in the life cycle?
- Are additional inspections on this life-cycle work product necessary?

Testing

Testing is a quality control function in that it is used to verify the functionality and performance of life-cycle work products or product components as they move through the product life cycle. Why test? The expected answer would be to find and eliminate defects, but there are other objectives for testing that are significant and worthy of consideration.

Purpose of Testing

The purpose of testing is to:

- Establish confidence that a program or system does what it is supposed to do.
- Make lack of quality visible.
- Execute a program with the intent of finding errors.
- Exercise a component to verify that it satisfied a specific requirement.
- Provide continual assessment of whether the software being produced will meet the needs of the user.

Thought Process Involved in Designing Tests

David Gelprin of SQE fame together with his partner, William Hetzel, came up with the slogan "Test then code." The first time I saw that slogan, I had to think about the message being offered. If an engineer, especially a software engineer, thinks about what it will take to test his/her program, stating the expected testing results helps spot system errors, often without ever running the test.

Key Testing Principles

A number of testing principles should be taken into account before testing activities are assigned:

- *Complete testing is not possible.* It has been stated and agreed to for decades that complete testing of a system is not always possible. If we accept this premise, then what are the guidelines to help determine what should be tested? It is not adequate for any individual involved in the various stages of testing to simply decide to test or not test a component, an interface, or a scenario based on their experience and feelings. What should be tested and to what extent should it be documented according to project/organizational guidelines? What account factors, such as criticality, complexity, interfaces, and past problems with the component if it is a reuse component for this project, should be considered?

- *Testing is creative and difficult.* Testing is not an activity that is given to an engineer simply because he/she is new to the organization, is expendable, or management cannot decide what else to do with the person. In my past management years, I had the privilege of managing a gentleman who had more than 40 years of experience building microprocessors, developing the software to be run on them, and testing the software and hardware together as a system. He was briefly put into a management position, but it was quickly realized that his systems testing skills were so great that it was best to make him a technical fellow, have him continue with his testing activities, and be responsible for mentoring and coaching the next generations of testers.
- *Testing must be planned.* Testing is an intricate set of activities that must be planned just like any other project activity. Planning for testing, including the testing environment, is not a trivial task and should not be minimized. The following aspects should be considered for every test:
  - Test philosophy and criticality;
  - Objectives and completion criteria;
  - Methods;
  - Responsibilities and people involved;
  - Resources and test tools;
  - Budget;
  - Schedule;
  - Documentation;
  - Problem recording;
  - Problem fixing.
- *Testing requires independence.* Although unit testing or "white box" testing is normally carried out by the developers, integration and systems testing requires individuals who are independent from the project developers. It is not human nature for a developer to conduct unit testing and show his boss the defects that were found during that exercise, and then put on a systems test hat and show his boss all of the defects that he found on his components during systems testing. Objective evaluations are required for effective systems testing.
- *Expected results.* A necessary part of a test case is a definition of the expected output or result.
- *Invalid and unexpected input conditions.* Test cases must be written for invalid and unexpected input conditions as well as for valid and expected input conditions.
- *What is the program expected to do?* Examining a program to see if it does what it is supposed to do is only half of the battle. The other half is seeing whether the program does what it is not supposed to do. In other words, we do not want programs to have functionality for which there were no agreed-on requirements.
- *Probability of more errors.* The probability of the existence of more errors in a section of a program is proportional to the number of errors already found in that section. This tautology is often perplexing for developers. If their testing

efforts result in a large number of defects found, they want to be rewarded—and they should be rewarded. But then, they must think about their design and perhaps design additional tests because there are at least as many defects left in the program as they found.

- *Good test case versus successful test case.* A good test case is one that has a high probability of detecting an error that has not been discovered yet. A successful test case is one that detects an error that has not been discovered yet.

Different Levels of Testing

Many different levels or types of testing can be performed throughout the development life cycle. The more standard types are briefly described here.

Unit Testing

Unit testing is a process of testing the individual components, subsystems, hardware components such as programmable logic arrays, and software components such as subprograms, subroutines, or procedures. Unit testing for software focuses on white box or glass box testing. It focuses on test statements, branches, and paths through discrete pieces of code.

The objectives of unit testing may be stated as basic questions:

- Does the logic work properly?
  - Does the component do what was intended?
  - Can the program or component/subsystem fail?
- Is all of the necessary logic present?
  - Are any functions missing?
  - Does the module do everything specified?
- Has any unplanned functionality been added?

Integration Testing

The objective of integration testing is to test component interfaces and confirm that the components meet the interface requirements and that the components can indeed be assembled. The product components need to be checked for quantity, obvious damage to hardware components, and consistency between the product component and the interface descriptions. Integrations and systems test engineers want to obtain a working skeleton as rapidly as possible and establish the confidence that the skeleton parts interface correctly. Integration testing also normally includes demonstrating that simple test cases and transactions are being handled properly by the system.

Systems Testing

Systems testing is the first time at which the entire system can be tested against the systems requirements specification. Systems testing measures and determines what

the system capabilities are and ends when the system capabilities have been measured and enough of the problems have been corrected to have confidence that the acceptance is ready to be executed. Systems test planning covers types of testing to be performed, test strategies, test coverage approaches, methods for tracing requirements to test cases, and reliability metrics.

Acceptance Testing

The purpose of acceptance testing is to confirm that a system is ready for operational use and that the confidence built up in systems testing is justified. The primary issue in acceptance testing is usability and reliability—will the system support operational use? Acceptance criteria should be discussed and agreed on in advance of the actual acceptance testing. The focus should be on functional requirements, performance requirements, operational use profile, and recovery requirements.

Acceptance test objectives must be clearly stated together with "success" criteria. Quality criteria such as reliability and maintainability must be acceptance tested. Installation documentation and user documentation must be included in the acceptance testing.

Regression Testing

Regression testing is the execution of a series of tests to check that modifications to parts of an existing system will not negatively affect other working components or subsystems. Regression testing should be applied when a baselined system is being enhanced or repaired. Regression testing criteria should be developed to establish what part of the regression test suite to run and the adequacy of the regression testing.

Test Readiness Criteria

For any level of testing, a component or components must have satisfied its/their readiness criteria. For example:

- Test plans, test procedures, and test cases are reviewed and documented by peers of the developers of the plans and procedures before they are considered ready for use.
- Components have successfully passed a peer review and unit testing before they enter integration testing.
- Components have successfully passed integration testing before they enter system testing.
- A test readiness review is held before the product or system enters acceptance testing.

Configuration Management of Test Items

Key engineering work products should be maintained under configuration management such as:

- Test plans;
- Test cases;
- Test procedures;
- Test scenarios;
- Test scripts;
- Test data.

Test Coverage for Software

Test coverage analysis is the process of finding areas of a program not exercised by a set of test cases, creating additional test cases to increase coverage, and determining a quantitative measure of code coverage that serves an indirect measure of quality.

Statement Coverage

Statement coverage measures whether each executable statement is encountered. Block coverage is the same as statement coverage except that the unit of code measured is each sequence of nonbranching statements. Do–while loops are considered the same as nonbranching statements.

Although statement coverage is a good start on test coverage, statement coverage is completely insensitive to the logical operators (|| and &&) and it cannot distinguish consecutive "switch" labels.

Decision Coverage

Decision coverage measures whether Boolean expressions tested in control structures (such as if-statements or while-statements) are evaluated to both true and false. The entire Boolean expression is considered one true-or-false predicate regardless of whether it contains logical "and" or logical "or" operators.

A disadvantage of decision coverage is that this measure branches within Boolean expressions that occur due to short-circuit operators:

```
If (condition1 && (condition2 | | function1()))
  statement1;
Else
  statement2;
```

This measure could consider the control structure completely exercised without a call to function1. The test expression is true when condition1 is true and condition2 is true. The test expression is false when condition1 is false. The short-circuit operators preclude a call to function1.

Condition Coverage

Condition coverage measures the true or false outcome of each Boolean subexpression. Condition coverage is similar to decision coverage but has better sensitivity to the control flow. Full condition coverage, however, does not guarantee full decision coverage. An example follows:

```
Bool f(bool e) {return false;}
Bool a[2] = {false, false};
If (f(a && b)) ….
If (a [int (a && b) ] ) …
If ((a && b) ? False : False) …
```

All three of the if-statements shown here branch false regardless of the values of *a* and *b*; however, if you exercise this code with *a* and *b* having all possible combinations of values, condition coverage reports full coverage.

Multiple Condition Coverage

Multiple condition coverage measures whether every possible combination of Boolean subexpression occurs. For languages with short-circuit operators (such as C, C++, and Java), an advantage of multiple condition coverage is that it requires very thorough testing. Multiple condition testing is similar to condition testing.

A disadvantage of this measure is that is can be difficult to determine the minimum set of test cases required. This measure could also require a varied number of test cases among conditions that have similar complexity. For example:

```
a && b && (c | | (d && e))
((a | | b) && (c | | d) ) && e
```

To achieve full multiple condition coverage, the first condition requires 6 test cases, whereas the second condition requires 11 test cases.

Path Coverage

Path coverage measures whether each of the possible paths in each function has been followed. Path coverage has the advantage of requiring very thorough testing but it also has two disadvantages:

- The number of paths is exponential to the number of branches. A function containing 10 if-statements has 1,024 paths to test—adding one more doubles the count to 2,048.
- Many paths are impossible to exercise due to relationships of data.

General Observations on Testing

I have had the privilege of conducting appraisals in more than 15 countries and working in 25 countries. Here are some general observations on testing from those experiences:

- Testing still gets squeezed at the end.
- Test results, particularly negative ones, are not respected and are oftentimes disregarded by developers.
- Test data and test case sets are frequently not planned or established.
- The integration test strategy is often left up to personal choice with minimal evidence of any documented standards, guidelines, or procedures.

- System tests are often no more than a collection of developers' unit and integration tests.
- Independent test groups are becoming more prevalent, but they are not staffed by senior software developers with in-depth application and technical knowledge. They are not staffed by engineers with a "testing mentality." Systems testers are not included in up-front planning with other engineering disciplines (e.g., systems engineering, software development, QA, and CM).
- Training for the testing function is frequently minimal or nonexistent.
- Most orgranizations do not have testing methodologies, resulting in unit testing being performed inadequately.
- There is often a lack of test cases, expected results of tests are not established, and integration and systems testing is left solely up to developers.
- Regression testing is seldom performed and does not typically take the complexity and criticality of the system into consideration. There is inadequate understanding of adequacy of regression tests being applied.

Unit Testing and the CMMI®

The reference to unit testing can be found in the Technical solution Process area. Specifically, it is Specific Practice 3.1, Subpractice 4 [2]. Unit testing involves the testing of individual hardware or software units or groups of related items prior to integration of those items. Frequently unit testing for software is left up to the developer. This may or may not be a good idea depending on the complexity or criticality of the module or unit that is being tested. An example unit test procedure is offered here as a "starter kit" on which the reader can base his/her unit tests.

Software Test Planning

The master software test plan contains general, high-level information regarding unit testing, integration testing, and hardware/software system testing. The topics addressed in the master software test plan are listed in Table 13.1. Specific test procedures, descriptions, and so on, are contained in supporting documentation (e.g., integration test specification).

Unit Test Specification/Procedure

The unit test specification/procedure is intended to guide the testing and verification of all functions within each coded program module and subsystem before it is integrated and system tested. Table 13.2 lists the minimum amount of information contained in the unit test specification/procedure. Specific unit test specification procedures and results of unit tests are contained in the software development file and/or technical data package.

The objectives of module testing may be stated as basic questions:

- Does the logic work properly?

Table 13.1 Master Software Test Plan

Unit Test Planning	Describe the requirements, responsibilities, and schedules for their testing. Provide general project standards for unit test thoroughness. Describe how the developer will conduct and monitor unit testing to ensure compliance with this plan. Identify unit test input data that must be supplied by external sources and the plan for obtaining these data.
Integration Test Planning	Describe the integration and test requirements. Identify the organizations responsible for the preparation, execution, reporting, control, review, and audit of the integration and testing. Describe the various types or classes of tests that will be executed during the integration and testing period (e.g., timing tests, erroneous input tests, maximum capacity tests). Describe an overall integration and test schedule that is consistent with overall development plans.
System Testing Planning	Describe the general requirements that apply to all formal tests and unique requirements that apply to selected formal tests. Identify the organizations responsible for the preparation, execution, reporting, review, audits, and control of formal tests. Describe the various classes of formal tests to be executed (e.g., timing tests, maximum capacity tests). For each class of test, describe: Test purpose; Software requirements to be demonstrated; Special software, hardware, and facility configurations to be used; Generic test input environment and output conditions. Identify each systems test and state its objectives. Identify the support software to be used in testing and the plan for obtaining and validating this software prior to the start of formal testing. Identify the computer hardware and interfacing equipment that will be required to conduct the formal testing, including any simulations needed before actual hardware is available. Define the criteria to be used for determining acceptable system performance. Define the protocol(s) to be followed by the developer in reviewing, reporting, and accepting test executions and results. Define the criteria and procedures for incorporating software modifications and performing retest. Describe any assumptions that were made in test planning and test conditions that must be observed during testing to ensure valid results.

- Does the code do what was intended?
- Can the program fail?

- Is all the necessary logic present?
  - Are any functions missing?
  - Does the module do everything specified?
- Has any unplanned functionality been added?

Evolutionary Path

The evolutionary path for implementing inspections, testing, and improving on defect prevention—not just defect detection—is described in the steps listed next:

Table 13.2 Minimum Amount of Information Contained in the Unit Test Specification/Procedure

Name of Person	Persons conducting/observing unit test.
Overview	A general description of the approach that will be used to actually perform the unit testing.
Unit Testing Schedule and Budget	Dates unit testing took place or retesting took place.
Roles and Responsibilities	Developer Unit tester (may be the same as the developer). Quality assurance (spot-checks actual results or unit tests against the expected results). Configuration management (project-level CM for updating modules, design, and unit tests, based on unit testing results). Project manager. Hardware or simulation support.
Assumptions	Any assumptions that will be used in performing the actual unit tests (e.g., the assumed state the system will be in when this module or unit is invoked).
Test Techniques/ Approach	A description of the testing techniques that will be used during unit testing. Recommended testing techniques include: Execution of all paths through the unit (if all possible paths cannot be executed, execution coverage criteria should be listed, i.e., statement coverage, branch coverage, or multiple condition coverage); Verification of the correctness of the function; Verification of the computational accuracy of the function; Testing for the function's response to illegal input; Testing for the function's response to saturation conditions; Testing to detect boundary condition errors; Verification of all data output options and formats, including error and information messages.
Unit Test Environment	The hardware and software configuration under which this unit will be tested.
Test Data Required	Type, format, and quantity of test data necessary.
Test Tools	Test tools and test harnesses that are required for successful execution of the unit tests.
Success Criteria	The *expected results* that indicate the module successfully passed the module or unit test.
Support Requirements	Any necessary software or hardware support necessary to conduct this unit test (e.g., test data generators, test drivers, prototype hardware).
Traceability to Requirements	Identifies which requirement or requirements this unit test is supposed to satisfy.
Test Procedure	The actual scenario or test script to be followed in executing the test.
Unit Interfaces	What interfaces must this unit or module be compatible with?
Performance Test	What are the performance requirements that can or must be shown with this unit?
Test Coverage	What test coverage is being achieved by the unit test? Example: Statement coverage; Decision coverage; Condition coverage; Multiple condition coverage; Path coverage; Function coverage; Loop coverage.
Test Reports	What is to be reported and in what format for each unit test? What is the scope of the corrective action required based on the unit test results? Was retesting required following corrective action?
Unit Test Defects	Categorization of unit test defects that can be compared with peer review results as well as the results of subsequent test efforts.

1. Train the development staff on the process of conducting inspections.
2. Provide special training for a group of moderators whose responsibility will be to lead the inspections.
3. Develop the high-level categories of major and minor defects.
4. Ensure that all developers and any other inspection participants follow the inspection process strictly to ingrain the process as it was intended to be followed. Allow no deviations until most developers have exhibited a willingness and a competence in conducting the inspection.
5. Develop finer grained categories of major defects for tracking, analyzing for trends, and process improvement.
6. Provide feedback to the participants of the inspections on the effectiveness of inspections in which they have participated.
7. Control the formality or informality of the inspections through the checking and logging rates.
8. Develop organizational optimal rates for checking and logging depending on life-cycle work product and life-cycle phase and capabilities of the reviewers.
9. Ensure that the testing methodology is documented and audited by quality assurance for compliance.
10. Develop readiness criteria for movement between component development (code for software) and unit testing, unit testing and integration testing, and integration testing and systems testing, and ensure their enforcement.
11. Develop equivalent categories of major defects for the testing phases.
12. Start comparing the types of defects found in the inspection of life-cycle work products in the early phases of the product life cycle with those found in the respective testing phases.
13. Perform causal analysis activities to determine targets of improvement on suspected weak processes.
14. Collect and analyze the defects found after the product has been placed into production for 1 to 2 years.
15. Start to compare the after-production defects found by the customer with those categories of major defects found in the testing phases and through inspections of the life-cycle work products in the early phases of product development.
16. Perform formal causal analysis on the defect types that make it through all of the inspections and testing and appear during production use.
17. Improve processes that contribute the most toward finding and eliminating the defects that affect cost, schedule, performance, quality, risk, and customer satisfaction when the product is placed into production.
18. Track the improved processes to determine their impact and cost effectiveness.
19. Improve the processes at the organizational level.
20. Train, mentor, and coach the implementation of those improved processes on all projects throughout the organization.

Summary

A review is an evaluation of a life-cycle work product(s) or project status to determine if there are any deviations from planned results and to recommend improvement. We conduct reviews to detect and remove defects as close to the point of insertion as possible, to identify potential improvement in the work product and the process used to develop it, and to reduce costs to build and maintain better products. We can also conduct management or progress reviews that help us answer these questions:

- Are we ahead or behind where we expected to be?
- Will we complete the work as planned?
- Is the required functionality being implemented?
- What risks are we taking based on the project information that we have today?

Peer review types that can be used for different objectives and expected results include buddy checks, circulation reviews, technical reviews, walkthroughs, and inspections.

An inspection is a formal verification technique in which life-cycle work products are examined in detail by a group of peers for the explicit purpose of detecting and identifying defects and is the most effective peer review technique.

Testing is a quality control function as it is verifying the functionality and performance of life-cycle work products or product components as they move through the product life cycle. The purpose of testing is to:

- Establish confidence that a program or system does what it is supposed to do;
- Make lack of quality visible;
- Execute a program with the intent of finding errors;
- Exercise a component to verify that it satisfied a specific requirement;
- Provide continual assessment of whether the software being produced will meet the needs of the user.

Test coverage analysis is the process of finding areas of a program not exercised by a set of test cases, creating additional test cases to increase coverage, and determining a quantitative measure of code coverage that serves an indirect measure of quality.

Peer reviews (inspections) combined with effective testing can find and detect most of the major defects before a product is put into production without being required to do 100% software inspections or 100% path coverage in testing.

Peer reviews and unit testing are the product component verification techniques suggested in the Technical Solution process area that prepare the product component for integration and systems testing. Without applying these verification techniques judiciously to the product components, a project loses control of its integration process with the result being late deliveries, poor quality, nonworking functionality, and rework!