**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.

•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).

**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 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.

**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.

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, and provide recommendations after the examination of various alternatives.

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.  
  
  
**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 effective-ness of the inspection process.

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.

**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.

**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;

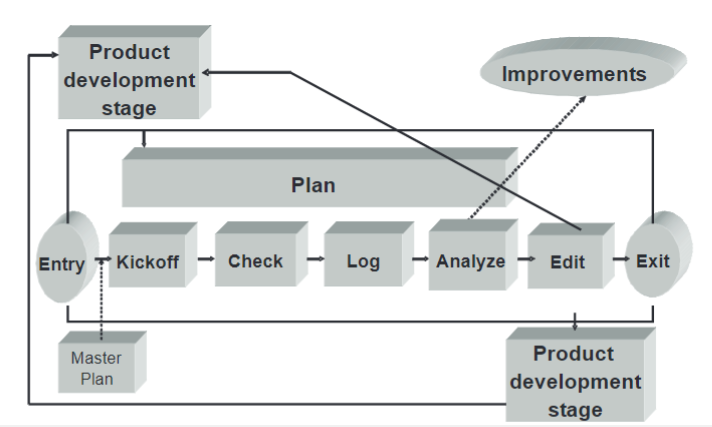
•Having roles and responsibilities preplanned such as recorder, reviewer, and any special roles;

•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.



**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.

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.

**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.

•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.

•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.

**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 whitebox 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.

**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 non-branching statements. Do–while loops are considered the same as non-branching 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.

-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.

-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 mini-mum set of test cases required. This measure could also require a varied number of test cases among conditions that have similar complexity.

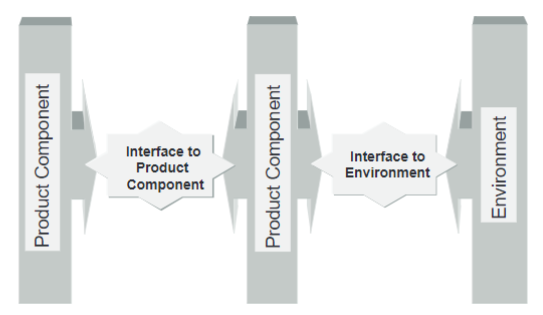
-Path Coverage

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.

**Product Integration**



**Assembly of Product Components**

Once the readiness criteria of the product components have been established, the actual process of integrating these product components can commence. This is normally expected to be done in an iterative fashion from the initial product components, through the interim assembly of product components, to the product as a whole. The readiness of the product integration environment should be ensured and the product components assembled according to the product integration sequence and integration procedures.

**Evaluation of Assembled Product Components**

Throughout the evolution of the product or product components, each interim state or final product state must be evaluated to show that they satisfy the functional, performance, and quality requirements. Actual product evaluation results should be compared against expected results and spot-checked by an independent party, such as quality assurance. The assembled product components must, therefore, be verified and validated according to the integration sequence and the verification and validation strategies.

**Verification**

Verification focuses not only on the final assembled product, but also includes verification of the intermediate work products against all selected requirements, including customer, product, and product component requirements. Verification methods address the technical approach to work product verification and the specifications, resources, and environments that will be used to verify specific work products. Verification typically begins with involvement in the definition of product and product component requirements to ensure that these requirements are verifiable. The verification environment must be appropriate to support the verification method and may be shared with product integration and validation.

**Verification Techniques and Methods**

Verification techniques ensure that the integrated product meets the specified requirements. In other words, it ensures that the project built the product right. Methods of verification include, but are not limited to:

•Inspections;

•Peer reviews;

•Audits;

•Walkthroughs;

•Analysis;

•Demonstrations;

•Comparisons;

•Path coverage testing;

•Simulations;

•Functional testing;

•Qualification testing;

•Factory testing;

•First article qualification;

•Bench testing;

•Load, stress, and performance testing;

•Operational scenario testing;

•Observations and demonstrations.

**Systems Testing Versus Bench Testing**

System testing can be defined as measuring a product component’s performance while installed in a real or reasonable approximation of a functional system. Bench testing, in contrast, is defined as the insertion of a product component into a test loop where all of the variables can be independently controlled, measured, and recorded.

**Validation**

System validation is an end-to-end process that is needed to ensure that the completed and integrated system will operate as needed in the environment for which it was intended. It can be defined as a measure of customer satisfaction, given the customer’s operational need and profile. System validation must always take into account the customer/end user during testing.

**Acceptance Testing**

Acceptance testing may satisfy the final verification and validation criteria. The purpose of acceptance testing is to confirm that a product or product component is ready for operational use. The acceptance test is performed for, or in conjunction with, someone else to demonstrate that the confidence is justified. The primary quality factors addressed are usability and reliability in order to answer the question “Will the product or product component support operational use?” Acceptance testing should also include the review of user documentation to verify its technical correctness compared to the system functionality, its completeness, and its ease of use.

**Packaging and Delivery**

Prior to packaging and delivering the product, the requirements, design, product, test results, and documentation are reviewed again to ensure that all issues affecting the packaging and delivery of the product are identified and resolved.