Unit III

Q1. State software testing principles.

Principle 1. *All tests should be traceable to customer requirements.* 11 The objective of software testing is to uncover errors. It follows that the most severe defects (from the customer‟s point of view) are those that cause the program to fail to meet its requirements.

Principle 2. *Tests should be planned long before testing begins.* Test planning can begin as soon as the requirements model is complete. Detailed definition of test cases can begin as soon as the design model has been solidified. Therefore, all tests can be planned and designed before any code has been generated.

Principle 3. *The Pareto principle applies to software testing.* In this context the Pareto principle implies that 80 percent of all errors uncovered during testing will likely be traceable to 20 percent of all program components. The problem, of course, is to isolate these suspect components and to thoroughly test them.

Principle 4. *Testing should begin "in the small" and progress toward testing "in the large."* The first tests planned and executed generally focus on individual components. As testing progresses, focus shifts in an attempt to find errors in integrated clusters of components and ultimately in the entire system.

Principle 5. *Exhaustive testing is not possible.* The number of path permutations for even a moderately sized program is exceptionally large. For this reason, it is impossible to execute every combination of paths during testing. It is possible, however, to adequately cover program logic and to ensure that all conditions in the component-level design have been exercised.

Principle 6. *Apply to each module in the system a testing effort commensurate with its expected fault density.* These are often the newest modules or the ones that are least understood by the developers.

Principle 7. *Static testing techniques can yield high results.* More than 85% of software defects originated in the software documentation (requirements, specifications, code walkthroughs, and user manuals).There may be value in testing the system documentation.

Principle 8. *Track defects and look for patterns in defects uncovered by testing.* The total defects uncovered is a good indicator of software quality. The types of defects uncovered can be a good measure of software stability. Patterns of defects found over time can forecast numbers of expected defects.

Principle 9. *Include test cases that demonstrate software is behaving correctly.* As software components are being maintained or adapted, unexpected interactions cause unintended side effects in other components.

Q. Explain any two types of system testing Recovery Testing

Many computer-based systems must recover from faults and resume processing with little or no downtime. In some cases, a system must be fault tolerant; that is, processing faults must not cause overall system function to cease. In other cases, a system failure must be corrected within a specified period of time or severe economic damage will occur. Recovery testing is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed.

Security Testing

Any computer-based system that manages sensitive information or causes actions that can improperly harm (or benefit) individuals is a target for improper or illegal penetration. Penetration spans a broad range of activities: hackers who attempt to penetrate systems for sport, disgruntled employees who attempt to penetrate for revenge, dishonest individuals who attempt to penetrate for illicit personal gain.

Stress testing executes a system in a manner that demands resources in abnormal quantity, frequency, or volume. For example, (1) special tests may be designed that generate 10 interrupts per second, when one or two is the average rate, (2) input data rates may be increased by an order of magnitude to determine how input functions will respond.

Performance Testing

For real-time and embedded systems, software that provides required function but does not conform to performance requirements is unacceptable. Performance testing is designed to test the run-time performance of software within the context of an integrated system. Performance testing occurs throughout all steps in the testing process. Even at the unit level, the performance of an individual module may be assessed as tests are conducted

Deployment Testing

Deployment testing, sometimes called configuration testing, exercises the software in each environment in which it is to operate. In addition, deployment testing examines all installation procedures and specialized installation software (e.g., “installers”) that will be used by customers, and all documentation that will be used to introduce the software to end users.

Q.Explain white box testing.

White-box testing, sometimes called *glass-box testing* or *structural testing,* is a test-case design philosophy that uses the control structure described as part of component-level design to derive test cases. Using white-box testing methods, you can derive test cases that (1) guarantee that all independent paths within a module have been exercised at least once, (2) exercise all logical decisions on their true and false sides, (3) execute all loops at their boundaries and within their operational bounds, and (4) exercise internal data structures to ensure their validity.

Advantages

- Code optimization by finding hidden errors.
- White box tests cases can be easily automated.
- Testing is more thorough as all code paths are usually covered.
- Testing can start early in SDLC even if GUI is not available.

Q.Explain how FTR is carried out

The FTR (Formal Technical Review) is a software quality assurance activity with the objectives to uncover errors in function, logic or implementation for any representation of the software; to verify that the software under review meets its requirements; to ensure that the software has been represented according to predefined standards; to achieve software that is developed in a uniform manner and to make projects more manageable.

FTR (Formal Technical Review) is also a learning ground for junior developers to know more about different approaches to software analysis, design and implementation. It also serves as a backup and continuity for the people who are not exposed to the software development so far. FTR (Formal Technical Review) activities include walkthroughs, inspection and round robin reviews and other technical assessments.

A formal technical review is a software quality assurance activity performed by software engineers (and others). The objectives of the FTR are

(1) to uncover errors in function, logic, or implementation for any representation of the software;
(2) to verify that the software under review meets its requirements;
(3) to ensure that the software has been represented according to predefined standards;
(4) to achieve software that is developed in a uniform manner; and
(5) to make projects more manageable.

The Review Meeting

Regardless of the FTR format that is chosen, every review meeting should abide by the following constraints:
• Between three and five people (typically) should be involved in the review.
• Advance preparation should occur but should require no more than two hours of work for each person.
• The duration of the review meeting should be less than two hours.

Review Reporting and Record Keeping

During the FTR, a reviewer (the recorder) actively records all issues that have been raised. These are summarized at the end of the review meeting and a review issues list is produced. In addition, a formal technical review summary report is completed.

A review summary report answers three questions:
1. What was reviewed?
2. Who reviewed it?
3. What were the findings and conclusions?

Review Guidelines

- *Review the product, not the producer.*
- *Set an agenda and maintain it.*
- *Limit debate and rebuttal.*
- *Enunciate problem areas, but don't attempt to solve every problem noted.*
- *Take written notes.*
- *Limit the number of participants and insist upon advance preparation.*
- *Develop a checklist for each product that is likely to be reviewed.*
- *Allocate resources and schedule time for FTRs.*
- *Conduct meaningful training for all reviewers.*
- *Review your early reviews.*

Q.what do you mean by software safety?

Software system safety optimizes system safety in the design, development, use, and maintenance of software systems and their integration with safety-critical hardware systems in an operational environment.

Software safety is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail. If hazards can be identified early in the software engineering process, software design features can be specified that will either eliminate or control potential hazards.

A modeling and analysis process is conducted as part of software safety. Initially, hazards are identified and categorized by criticality and risk. For example, some of the hazards associated with a computer- based cruise control for an automobile might be

• causes uncontrolled acceleration that cannot be stopped

• does not respond to depression of brake pedal (by turning off)

- Safety consistent with mission requirements, is designed into the software in a timely, cost effective manner.
- On complex systems involving many interactions safety-critical functionality should be identified and thoroughly analyzed before deriving hazards and design safeguards for mitigations.
- Safety-critical functions lists and preliminary hazards lists should be determined proactively and influence the requirements that will be implemented in software.
- Contributing factors and root causes of faults and resultant hazards associated with the system and its software are identified, evaluated and eliminated or the risk reduced to an acceptable level, throughout the lifecycle.
- Reliance on administrative procedures for hazard control is minimized.
- The number and complexity of safety critical interfaces is minimized.

Q. Write a short note on software quality

Quality is defined as “a characteristic or attribute of something.” As an attribute of an item, quality refers to measurable characteristics— things we are able to compare to known standards such as length, color, electrical properties, and malleability. However, software, largely an intellectual entity, is more challenging to characterize than physical objects. Nevertheless, measures of a program‟s characteristics do exist. These properties include cyclomatic complexity, cohesion, number of function points, lines of code, and many others.

When we examine an item based on its measurable characteristics, two kinds of quality may be encountered: quality of design and quality of conformance. Quality of design refers to the characteristics that designers specify for an item.

The grade of materials, tolerances, and performance specifications all contribute to the quality of design. As higher-grade materials are used, tighter tolerances and greater levels of performance are specified, the design quality of a product increases, if the product is manufactured according to specifications.

Quality of conformance is the degree to which the design specifications are followed during manufacturing. Again, the greater the degree of conformance, the higher is the level of quality of conformance.

Q.Short note on Defect removal

The data collected during a software process are used to compute a set of metrics that support evaluation and improvement of the process as well as planning and tracking quality. -Many different metrics can be calculated during an inspection process including the following: -

The metrics computed during such process should be defined by the requirements of your organization (typically in the quality manual).

•The number of major and minor defects found

•The number of major defects found to total found. (If the proportion of minor defects to major defects is too large a moderator may request that the reviewer repeat the review, focusing on major defects, prior to commencing the logging meeting)

•The size of the artifact (pages, LOC, ...)

• The rate of review - the size of the reviewed artifact divided by time (normally expressed in hours) (e.g., 15 pages /hour).

•The defect detection rate - the number of major defects found per review hour.

Total Number of Defects Found and Defects Density

-Total number of defects found is the sum of the total number of defect s found by each reviewer, minus the number of common defects found.

Total Defects Found = A + B - C Where A and B are the number found by reviewer A and B respectively and C is the number found by both A and B.

For instance, with 2 reviewers, the metric is computed by -Defect density is the ratio of the number of defects found to the size of the artifact. It is given by Defect Density = Total Defects Found / Size Where the size of the artifact is measured in number of pages, loc, or other size measure.

Q.What do you mean by integration testing. Explain methods of integration testing

In integration Testing, individual software modules are integrated logically and tested as a group.

A typical software project consists of multiple software modules, coded by different programmers. Integration Testing focuses on checking data communication amongst these modules.

Although each software module is unit tested, defects still exist for various reasons like

- A Module in general is designed by an individual software developer whose understanding and programming logic may differ from other programmers.

- Interfaces of the software modules with the database could be erroneous
- External Hardware interfaces, if any, could be erroneous
- Inadequate exception handling could cause issues.

  Big Bang Approach :

  Incremental Approach: which is further divided into following

  - Top Down Approach
  - Bottom Up Approach
  - Sandwich Approach - Combination of Top Down and Bottom Up

Big Bang Approach:

Here all component are integrated together at **once**, and then tested.

Advantages:

- Convenient for small systems.

Disadvantages:

- Fault Localization is difficult.
- Given the sheer number of interfaces that need to be tested in this approach, some

interfaces links to be tested could be missed easily.

- Since the integration testing can commence only after "all" the modules are designed, testing team will have less time for execution in the testing phase.
- Since all modules are tested at once, high risk critical modules are not isolated and tested on priority. Peripheral modules which deal with user interfaces are also not isolated and tested on priority.

Incremental Approach:

In this approach, testing is done by joining two or more modules that are ***logically related***. Then the other related modules are added and tested for the proper functioning. Process continues until all of the modules are joined and tested successfully.

This process is carried out by using dummy programs called **Stubs and Drivers**. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.

Stub: Is called by the Module under Test.

Driver: Calls the Module to be tested.

Incremental Approach in turn is carried out by two different Methods:

- **Bottom Up**
- **Top Down**

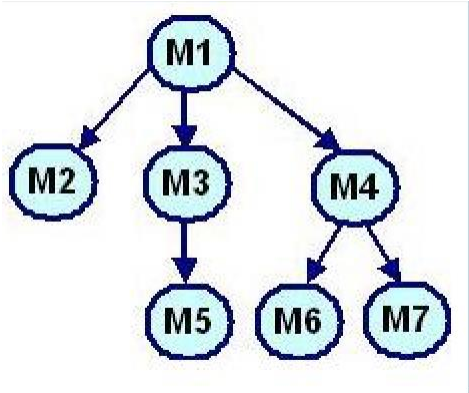
Bottom up Integration

In the bottom up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing

In Top to down approach, testing takes place from top to down following the control flow of the software system.

Top Down Approach (TDA):
Involves development of all the topmost / parent modules before hand followed by due integration with child modules.

This approach uses a Program called "Stub". While integrating the modules with this Top Down Approach, if at any stage it is found that some mandatory module is missing, then in such an event that particular module is replaced with a temporary program known as "Stub".



Advantages of Top down Approach:

a) This approach is advantageous if all major flaws are captured towards the top of the program

b) Early skeletal program allows demonstrations and boosts the morale

Disadvantages of Top down Approach:

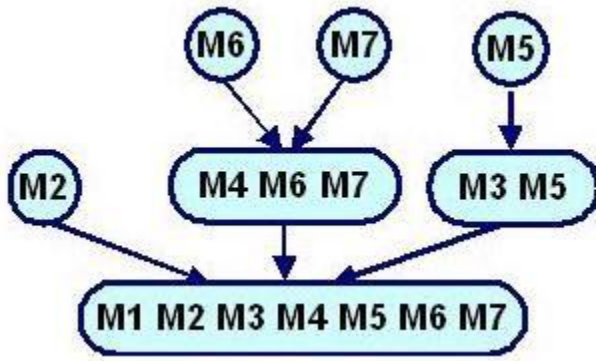
a) Stub modules are essential

b) Test conditions may be impossible, or very difficult, to create

c) Observation of test output is more difficult, as only simulated values will be used initially

Bottom Up Approach (BUA):
Involves development of all the elementary modules or child modules before hand followed by due integration with the corresponding parent modules.

This approach uses a Program called "Driver". While integrating the modules with this Bottom Up Approach, if at any stage it is found that some mandatory module is missing, then in such an event that particular module is replaced with a temporary program known as "Driver".



Advantages of Bottom Up Approach:

a) This approach is advantageous if all major flaws are captured towards the bottom of the program

b) Test conditions are easier to create

c) Observations of test results are easier since "live" data is used from the beginning.

Disadvantages of Bottom Up Approach:

a) Driver modules are essential>

b) The program as an entity does not exist until the last module is added

Q. Explain Control structure testing.

1.Condition Testing

Condition testing is a test case design method that exercises the logical conditions contained in a program module. A simple condition is a Boolean variable or a relational expression, possibly preceded with one NOT (¬) operator. A relational expression takes the form
E1 <relational-op>E2
Therefore, types of errors in a condition include the following:
• Boolean operator error (incorrect/missing/extra Boolean operators).
• Boolean variable error.
• Boolean parenthesis error.
• Relational operator error.
• Arithmetic expression error.

2) Data Flow Testing

The data flow testing method selects test paths of a program according to the locations of definitions and uses of variables in the program.

DEF(S) = {X | statement S contains a definition of X}

USE(S) = {X | statement S contains a use of X}

If statement S is an if or loop statement, its DEF set is empty and its USE set is based on the condition of statement S. The definition of variable X at statement S is said to be live at statement S' if there exists a path from statement S to statement S' that contains no other definition of X.

3) Loop Testing

Loops are the cornerstone for the vast majority of all algorithms implemented in software. And yet, we often pay them little heed while conducting software tests. Loop testing is a white-box testing technique that focuses exclusively on the validity of loop constructs. Four different classes of loops can be defined: simple loops, concatenated loops, nested loops, and unstructured loops.

Simple loops.

The following set of tests can be applied to simple loops, where n is the maximum number of allowable passes through the loop. 1. Skip the loop entirely. 2. Only one pass through the loop. 3. Two passes through the loop. 4. m passes through the loop where m < n. 5. n 1, n, n + 1 passes through the loop.

Nested loops:

If we were to extend the test approach for simple loops to nested loops, the number of possible tests would grow geometrically as the level of nesting increases. This would result in an impractical number of tests. Beizer [BEI90] suggests an approach that will help to reduce the number of tests:

1. Start at the innermost loop. Set all other loops to minimum values.

2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter (e.g., loop counter) values. Add other tests for out-of-range or excluded values.

Concatenated loops.

Concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other. However, if two loops are concatenated and the loop counter for loop 1 is used as the initial value for loop 2, then the loops are not independent. When the loops are not independent, the approach applied to nested loops is recommended.

Unstructured loops.

Whenever possible, this class of loops should be redesigned to reflect the use of the structured programming constructs

Q. Explain SQA plan

The SQA Plan provides a road map for instituting software quality assurance. Developed by the SQA group, the plan serves as a template for SQA activities that are instituted for each software project.

Initial sections describe the purpose and scope of the document and indicate those software process activities that are covered by quality assurance.

All documents noted in the SQA Plan are listed and all applicable standards are noted. The management section of the plan describes SQA"s place in the organizational structure, SQA tasks and activities and

their placement throughout the software process, and the organizational roles and responsibilities relative to product quality.

The standards, practices, and conventions section lists all applicable standards and practices that are applied during the software process (e.g., document standards, coding standards, and review guidelines).

The reviews and audits section of the plan identifies the reviews and audits to be conducted by the software engineering team, the SQA group, and the customer. It provides an overview of the approach for each review and audit.

Q.Discuss Quality concepts involved in software development

Quality of design refers to the characteristics that designers specify for an item. The grade of materials, tolerances, and performance specifications all contribute to the quality of design

Quality of conformance is the degree to which the design specifications are followed during manufacturing.

In software development, quality of design encompasses requirements, specifications, and the design of the system

Quality control

Quality control involves the series of inspections, reviews, and tests used throughout the software process to ensure each work product meets the requirements placed upon it. Quality control includes a feedback loop to the process that created the work product.

Quality Assurance

Quality assurance consists of the auditing and reporting functions of management. The goal of quality assurance is to provide management with the data necessary to be informed about product quality, thereby gaining insight

Cost of quality

The cost of quality includes all costs incurred in the pursuit of quality or in performing quality-related activities.

Quality costs may be divided into costs associated with prevention, appraisal, and failure. Prevention costs include • quality planning • formal technical reviews • test equipment

Q. Explain Software reliability

Software reliability is defined in statistical terms as "the probability of failure-free operation of a computer program in a specified environment for a specified time"

To illustrate, program X is estimated to have a reliability of 0.96 over eight elapsed processing hours. In other words, if program X were to be executed 100 times and require eight hours of elapsed processing time (execution time), it is likely to operate correctly (without failure) 96 times out of 100.

In the context of any discussion of software quality and reliability, failure is nonconformance to software requirements. Yet, even within this definition, there are gradations. Failures can be only annoying or catastrophic. One failure can be corrected within seconds while another requires weeks or even months to correct.

If we consider a computer-based system, a simple measure of reliability is

Meantime-between-failure (MTBF),

where MTBF = MTTF + MTTR The acronyms MTTF and MTTR are mean-time-to-failure and mean-time-to-repair, respectively.

In addition to a reliability measure, we must develop a measure of availability. Software availability is the probability that a program is operating according to requirements at a given point in time and is defined as

Availability = [MTTF/(MTTF + MTTR)] 100%

Q. Write a short note on software reviews

Software reviews are a "filter" for the software engineering process. That is, reviews are applied at various points during software development and serve to uncover errors and defects that can then be removed.

Software reviews "purify" the software engineering activities that we have called analysis, design, and coding]

A review—any review—is a way of using the diversity of a group of people to:

1. Point out needed improvements in the product of a single person or team;

2. Confirm those parts of a product in which improvement is either not desired or not needed;

3. Achieve technical work of more uniform, or at least more predictable, quality than can be achieved without reviews, in order to make technical work more manageable.

Many different types of reviews can be conducted as part of software engineering. Each has its place. An informal meeting around the coffee machine is a form of review, if technical problems are discussed.

Q. Explain approaches to object oriented testing

Unit Testing in the OO Context

When object-oriented software is considered, the concept of the unit changes. Encapsulation drives the definition of classes and objects. This means that each class and each instance of a class (object) packages attributes (data) and the operations (also known as methods or services) that manipulate these data.

Integration Testing in the OO Context

integrating operations one at a time into a class (the conventional incremental integration approach) is often impossible because of the "direct and indirect interactions of the components that make up the class"

Validation Testing in an OO Context

At the validation or system level, the details of class connections disappear. Like conventional validation, the validation of OO software focuses on user-visible actions and user-recognizable output from the system.

Q. Define SQA. Explain its role.

Software quality assurance (**SQA**) is a process that ensures that developed **software** meets and complies with defined or standardized quality specifications. **SQA** is an ongoing process within the **software** development life cycle (SDLC) that routinely checks the developed **software** to ensure it meets desired quality measures.

Prepares an SQA plan for a project.

Participates in the development of the project‟s software process description.
Reviews software engineering activities to verify compliance with the defined software process
Audits designated software work products to verify compliance with those defined as part of the software process.
Ensures that deviations in software work and work products are documented and handled according to a documented procedure.
Records any noncompliance and reports to senior management.

Q. Discuss features of smoke testing.

Smoke Testing, also known as "Build Verification Testing", is a type of software testing that comprises of a non-exhaustive set of tests that aim at ensuring that the most important functions work. The results of this testing is used to decide if a build is stable enough to proceed with further testing.

Features:

- Identifying the business critical functionalities that a product must satisfy.
- Designing and executing the basic functionalities of the application.
- Ensuring that the smoke test passes each and every build in order to proceed with the testing.
- Smoke Tests enables uncovering obvious errors which saves time and effort of test team.
- Smoke Tests can be manual or automated.

Advantages of Smoke testing:

- It helps to find issues introduced in integration of modules.
- It helps to find issues in the early phase of testing.
- It helps to get confidence to tester that fixes in the previous builds not breaking major features

Q. What do you mean by quality movement?

- **Total Quality Management** (TQM) is a popular approach for management practice.
- TQM stresses continuous process improvement and can be applied to software development.

Not much can be done to improve quality until a visible, repeatable, and measurable process is created.

Refers to a system of continuous process improvement.

- develop a process that is visible, repeatable, and measurable.

2. *This* step examines intangibles that affect the process and works to optimize their impact on the process.

3. This step concentrates on the user of the product. Examining the way the user applies the product. This step leads to improvement in the product itself and, potentially, to the process that created it.

4. This is a business-oriented step that looks for opportunity in related areas identified by observing the use of the product in the marketplace.

Q. Explain Statistical SQA

Each defect needs to be traced to its cause.

Defect causes having the greatest impact on the success of the project must be addressed first.

1. Information about software defects is collected and categorized.
2. An attempt is made to trace each defect to its underlying cause (e.g., nonconformance to specifications, design error, violation of standards, poor communication with the customer).
3. Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the "vital few").
4. Once the vital few causes have been identified, move to correct the problems that have caused the defects.

Q.List and explain features of BVA

Boundary value analysis is a test case design technique that complements equivalence Partitioning

Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well.

Guidelines for BVA

1. If an input condition specifies a range bounded by values a and b, test cases should be designed with values a and b and just above and just below a and b.

2. If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.

3. Apply guidelines 1 and 2 to output conditions. For example, assume that a temperature vs. pressure table is required as output from an engineering analysis program. Test cases should be designed to create an output report that produces the maximum (and minimum) allowable number of table entries.

4. If internal program data structures have prescribed boundaries (e.g., an array has a defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary.

Q.Difference between verification and validation

Criteria	Verification	Validation
Definition	The process of evaluating work-products (not the actual final product) of a development phase to determine whether they meet the specified requirements for that phase.	The process of evaluating software during or at the end of the development process to determine whether it satisfies specified business requirements.
Objective	To ensure that the product is being built according to the requirements and design specifications. In other words, to ensure that work products meet their specified requirements.	To ensure that the product actually meets the user's needs and that the specifications were correct in the first place. In other words, to demonstrate that the product fulfills its intended use when placed in its intended environment.
Question	Are we building the product *right*?	Are we building the *right* product?
Evaluation Items	Plans, Requirement Specs, Design Specs, Code, Test Cases	The actual product/software.
Activities	• Reviews • Walkthroughs • Inspections	• Testing

. Walkthrough

A walkthrough is conducted by the author of the „document under review" who takes the participants through the document and his or her thought processes, to achieve a common understanding and to gather feedback. This is especially useful if people from outside the software discipline are present, who are not used to, or cannot easily understand software development documents. The content of the document is explained step by step by the author, to reach consensus on changes or to gather information. The participants are selected from different departments and backgrounds If the audience represents a broad section of skills and disciplines, it can give assurance that no major defects are „missed" in the walk-through. A walkthrough is especially useful for higher-level documents, such as requirement specifications and architectural documents.

The specific goals of a walkthrough are:-
• to present the document to stakeholders both within and outside the software discipline, in order to gather information regarding the topic under documentation.
• to explain and evaluate the contents of the document.
• to establish a common understanding of the document.
• to examine and discuss the validity of proposed solutions and the possible alternatives.

Inspection is the most formal review type. It is usually led by a trained moderator. The document under inspection is prepared and checked thoroughly by the reviewers before the meeting, comparing the work product with its sources and other referenced documents, and using rules and checklists. In the inspection meeting the defects found are logged. Depending on the organization and the objectives of a project, inspections can be balanced to serve a number of goals.

The specific goals of Inspection are:
• help the author to improve the quality of the document under inspection.
• remove defects efficiently, as early as possible.
• improve product quality, by producing documents with a higher level of quality.
• create a common understanding by exchanging information among the inspection participants.