CO1 :	Apply various software testing methods

Software testing:

- Software testing is defined as performing Verification and Validation of the Software Product for its correctness and accuracy of working.
- Software Testing is the process of executing a program with the intent of finding errors.
- A successful test is one that uncovers an as-yet-undiscovered error.
- Testing can show the presence of bugs but never their absence.
- Testing is a support function that helps developers look good by finding their mistakes before anyone else does.

Role of testing / Objectives of testing:

1. Finding defects which may get created by the programmer while developing the software.
2. Gaining confidence in and providing information about the level of quality.
3. To prevent defects.
4. To make sure that the end result meets the business and user requirements.
5. To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
6. To gain the confidence of the customers by providing them a quality product

What is Software testing?

- Finding defects
- Trying to break the system
- Finding and reporting defects
- Demonstrating correct functionality
- Demonstrating incorrect functionality
- Demonstrating robustness, reliability, security, maintainability, …
- Measuring performance, reliability, …
- Evaluating and measuring quality
- Proving the software correct
- Executing pre-defined test cases
- Automatic error detection

Skills Required for Tester

- Communication skills
- Domain knowledge
- Desire to learn
- Technical skills
- Analytical skills
- Planning
- Integrity

- Curiosity
- Think from users perspective
- Be a good judge of your product

Bug, Fault & Failure

- A person makes an **Error**
- That creates a **fault** in software
- That can cause a **failure** in operation
- **Error: An error is a human action that produces the incorrect result that results in a fault.**
- **Bug: The presence of error at the time of execution of the software.**
- **Fault: State of software caused by an error.**
- **Failure: Deviation of the software from its expected result. It is an event.**
- **Defect: A defect is an error or a bug**, in the application which is created. A programmer while designing and building the software can make mistakes or error. These mistakes or errors mean that there are flaws in the software. These are called defects.

Why do defects occur in software?

Software is written by human beings

Who know something, but not everything
Who have skills, but aren't perfect
Who don't usually use rigorous methods
Who do make mistakes (errors)

Under increasing pressure to deliver to strict deadlines

No time to check, assumptions may be wrong
Systems may be incomplete

Software is complex, abstract and invisible

Hard to understand
Hard to see if it is complete or working correctly
No one person can fully understand large systems
Numerous external interfaces and dependencies

Sources of defects

Education

Developers does not understand well enough what he or she is doing
Lack of proper education leads to errors in specification, design, coding, and testing

Communication

Developers do not know enough
Information does not reach all stakeholders
Information is lost

Oversight

Omitting to do necessary things

Transcription

Developer knows what to do but simply makes a mistake

Process

Process is not applicable for the actual situation
Process places restrictions that cause errors

Test Plan

A test plan is a systematic approach to testing a system i.e. software. The plan typically contains a detailed understanding of what the eventual testing workflow will be.

Test Case

A test case is a specific procedure of testing a particular requirement.

It will include:

- Identification of specific requirement tested
- Test case success/failure criteria
- Specific steps to execute test
- Test Data

Entry and Exit Criteria for software testing

Process model is a way to represent any given phase of software development that prevent and minimize the delay between defect injection and defect detection/ correction.
Entry criteria, specifies when that phase can be started also included the inputs for the phase. Tasks or steps that need to be carried out in that phase along with measurements that characterize the tasks. Verification, which specifies methods of checking that tasks have been carried out correctly. Clear entry criteria make sure that a given phase does not start prematurely. The verification for each phase helps to prevent defects. At least defects can be minimized.

Exit criteria, which stipulate the conditions under which one can consider the phases as done and included are the outputs for the phase.
Exit criteria may include:

1. All test plans have been run
2. All requirements coverage has been achieved.
3. All severe bugs are resolved.

ENTRY CRITERIA

Entry Criteria for QA testing is defined as "Specific conditions or on-going activities that must be present before a process can begin". In the Systems Development Life Cycle it also specifies which entry criteria are required at each phase. Additionally, it is also important to define the time interval or required amount of lead time that an entry criteria item is available to the process. Input can be divided into two categories. The first is what we receive from development. The second is what we produce that acts as input to later test process steps.

The type of required input from development includes:

1. Technical Requirements/Statement of Need
2. Design Document
3. Change Control
4. Turnover Document

The type of required input from test includes:

1. Evaluation of available software test tools
2. Test Strategy
3. Test Plan
4. Test Incident Reports

By referencing the Entry Exit Criteria matrix, we get the clarity of the deliverables expected from each phase. The matrix should contain "date required" and should be modified to meet the specific goals and requirements of each test effort based on size and complexity.

EXIT CRITERIA

Exit Criteria is often viewed as a single document concluding the end of a life cycle phase. Exit Criteria is defined as "The specific conditions or on-going activities that must be present before a life cycle phase can be considered complete. The life cycle specifies which exit criteria are required at each phase". This definition identifies the intermediate deliverables, and allows us to track them as independent events.

The type of output from test includes:

1. Test Strategy
2. Test Plan
3. Test Scripts/Test Case Specifications
4. Test Logs
5. Test Incident Report Log
6. Test Summary Report/Findings Report

By identifying the specific Exit criteria, we are able to identify and plan how these steps and processes fit into the life cycle. All of the Exit Criteria listed above, less the Test Summary/Findings Report; act as Entry Criteria to alter process.

Verification & Validation

- **Verification**
    - Are you building the product right?
    - Software must conform to its specification
- **Validation**
    - Are you building the right product?
    - Software should do what the user really requires

What is Verification?

Definition: The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

- Verification is a static practice of verifying documents, design, code and program. It includes all the activities associated with producing high quality software: inspection, design analysis and specification analysis. It is a relatively objective process.
- Verification will help to determine whether the software is of high quality, but it will not ensure that the system is useful. Verification is concerned with whether the system is well-engineered and error-free.
- *Methods of Verification : Static Testing*
- *Walkthrough*
- *Inspection*
- *Review*

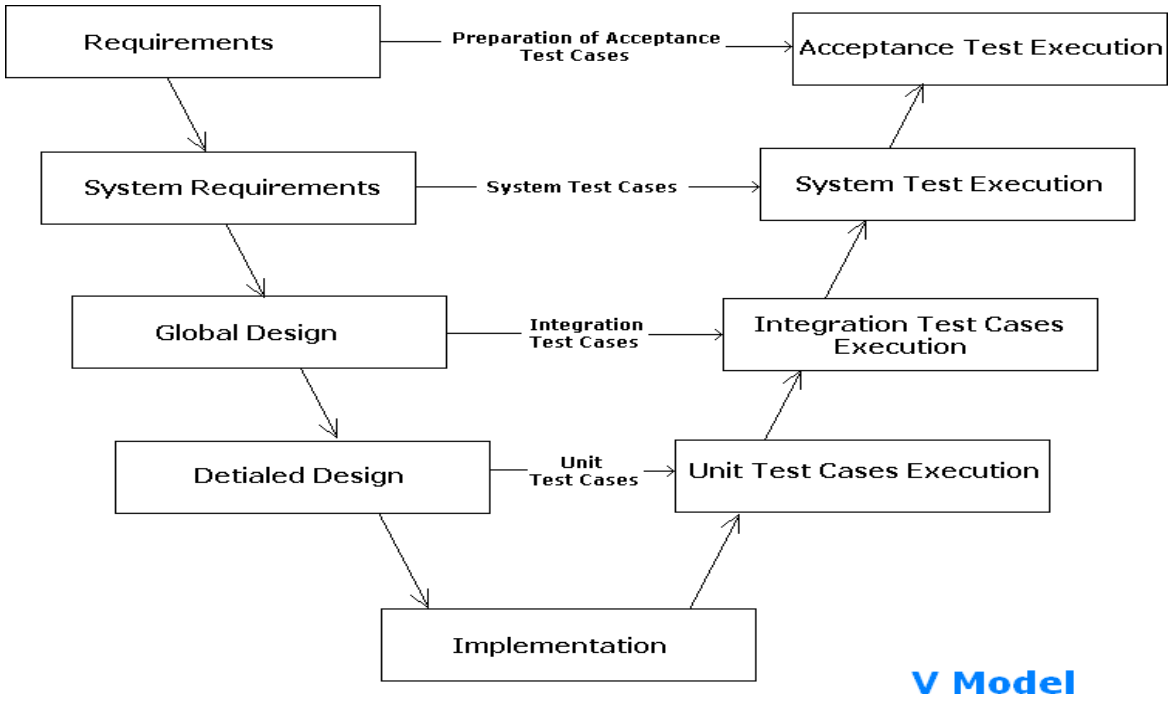
What is Validation?

Definition: *The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements.*

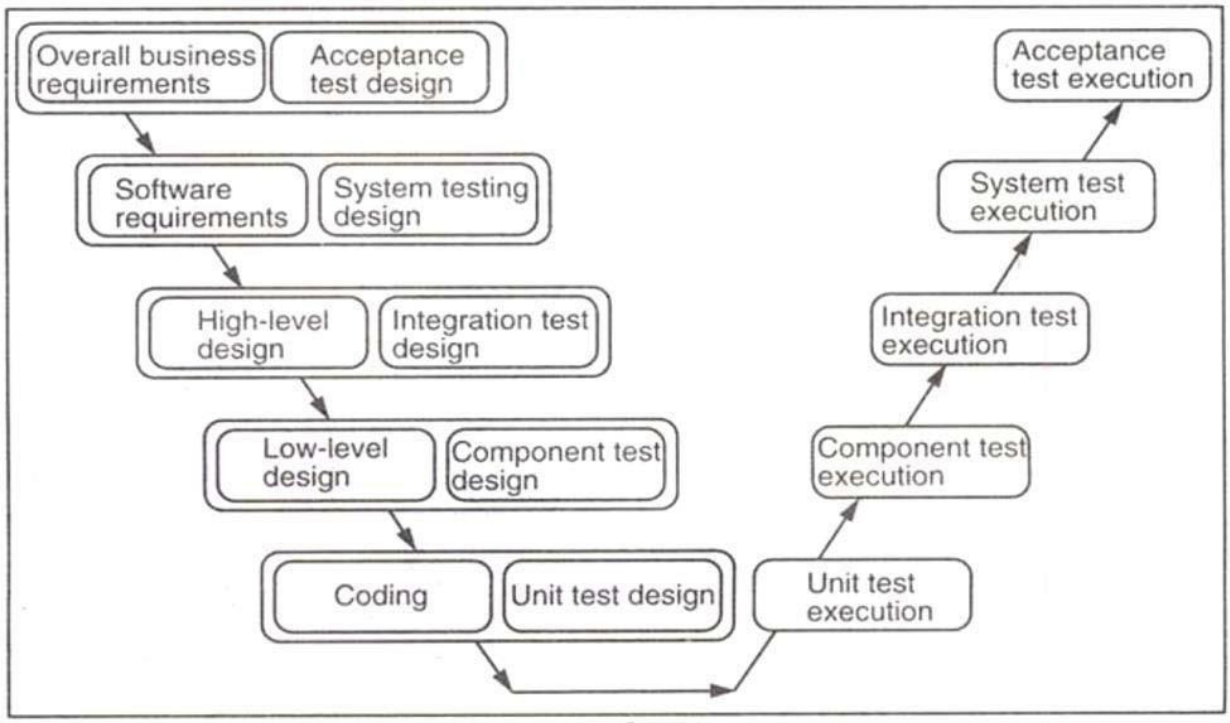
- Validation is the process of evaluating the final product to check whether the software meets the customer expectations and requirements. It is a dynamic mechanism of validating and testing the actual product.
- *Methods of Validation : Dynamic Testing*
- *Testing*
- *End Users*

Verification	***Validation***
1. **Verification is a static practice of verifying documents, design, code and program.**	**1. Validation is a dynamic mechanism of validating and testing the actual product.**
2. **It does not involve executing the code.**	**2. It always involves executing the code.**
3. **It is human based checking of documents and files.**	***3.*** **It is computer based execution of program.**
4. **Verification uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.**	***4.*** **Validation uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.**
5. Verification is to check whether the software conforms to specifications.	**5. Validation is to check whether software meets the customer expectations and requirements.**
6. **It can catch errors that validation cannot catch. It is low level exercise.**	***6. It can catch errors that verification cannot catch. It is High Level Exercise.***
7. Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	***7.*** **Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.**
8. Verification is done by QA team to ensure that the software is as per the specifications in the SRS document.	***8. Validation is carried out with the involvement of testing team.***

9. It generally comes first-done before validation.	***9.* It generally follows after verification.**



V Model



Verification and validation model makes the V-model. It is sequential path of execution of processes. Each phase must be completed before the next phase begins. Under V-model, the corresponding testing phase of the development phase is planned in parallel. So there is verification on one side of V & validation phase on the other side of V.

Verification Phase:

1. **Overall Business Requirement:** In this first phase of the development cycle, the product requirements are understood from customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirements. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.
2. **Software Requirement**: Once the product requirements are clearly known, the system can be designed. The system design comprises of understanding & detailing the complete hardware, software & communication set up for the product under development. System test plan is designed based on system design. Doing this at earlier stage leaves more time for actual test execution later.
3. **High level design:** High level specification are understood & designed in this phase. Usually more than one technical approach is proposed & based on the technical & financial feasibility, the final decision is taken. System design is broken down further into modules taking up different functionality.
4. **Low level design:** In this phase the detailed integral design for all the system modules is specified. It is important that the design is compatible with the other modules in the system & other external system. Components tests can be designed at this stage based on the internal module design,
5. **Coding:** The actual coding of the system modules designed in the design phase is taken up in the coding phase. The base suitable programming language is decided base on requirements. Coding is done based on the coding guidelines & standards.

Validation:

1. Unit Testing: Unit testing designed in coding are executed on the code during this validation phase. This helps to eliminate bugs at an early stage.

2. Components testing: This is associated with module design helps to eliminate defects in individual modules.

3. Integration Testing: It is associated with high level design phase & it is performed to test the coexistence & communication of the internal modules within the system

5. System Testing: It is associated with system design phase. It checks the entire system functionality & the communication of the system under development with external systems. Most of the software & hardware compatibility issues can be uncovered using system test execution.

6. Acceptance Testing: It is associated with overall & involves testing the product in user environment. These tests uncover the compatibility issues with the other systems available in the user environment. It also uncovers the non-functional issues such as load & performance defects in the actual user environment.

Quality Assurance:

i. It is Process oriented activities.

ii. A part of quality management focused on providing confidence that quality requirements will be fulfilled.

iii. All the planned and systematic activities implemented within the quality system that can be demonstrated to provide confidence that a product or service will fulfill requirements for quality

iv. Quality Assurance is fundamentally focused on planning and documenting those processes to assure quality including things such as quality plans and inspection and test plans.

v. Quality Assurance is a system for evaluating performance, service, of the quality of a product against a system, standard or specified requirement for customers.

vi. Quality Assurance is a complete system to assure the quality of products or services. It is not only a process, but a complete system including also control. It is a way of management.

- **Standards:** Standards are the criteria's to which the s/w product is compared.
- **Documentations Standards**: Specify form and Contents for planning, analysis and product documentation and consistency throughout a project.
- **Design Standards**: Specify forms and contents of design product. They provide rules and methods for translating the s/w requirements into the s/w design.
- **Code Standards**: Specify the language in which code is to written and define any restrictions on use of language features.
- They define legal language structures, style conversions, rules for data structure and interface.
- **Procedure:** Expected steps to be followed in carrying out a process.

Quality Control:

i. It is Product oriented activities.

ii. A part of quality management focused on fulfilling quality requirements.

iii. The operational techniques and activities used to fulfill requirements for quality.

iv. Quality Control on the other hand is the physical verification that the product conforms to these planned arrangements by inspection, measurement etc.

v. Quality Control is the process involved within the system to ensure job management, competence and performance during the manufacturing of the product or service to ensure it meets the quality plan as designed.
vi. Quality Control just measures and determines the quality level of products or services.

Methods of testing:

1. **Static Testing :**

- Static testing is the testing of the software work products manually, or with a set of tools, but they are not executed.
- It starts early in the Life cycle and so it is done during the verification process.
- It does not need computer as the testing of program is done without executing the program. For example: reviewing, walk through, inspection, etc.
- Static testing consists of following methods
  1) Walkthrough
  2) Inspection
  3) Technical Review

Advantages of Static Testing

- Since static testing can start early in the life cycle, early feedback on quality issues can be established.
- By detecting defects at an early stage, rework costs are most often relatively low.
- Since rework effort is substantially reduced, development productivity figures are likely to increase.
- The evaluation by a team has the additional advantage that there is an exchange of information between the participants.
- Static tests contribute to an increased awareness of quality issues.

Disadvantages of Static Testing

- Demand great amount of time when done manually
- Automated tools works with only few programming languages
- Automated tools may provide false positives and false negatives
- Automated tools only scan the code
- Automated tools cannot pinpoint weak points that may create troubles in run-time

2. **Dynamic Testing**

- Dynamic testing (or dynamic analysis) is a term used in software engineering to describe the testing of the dynamic behavior of code.
- That is, dynamic analysis refers to the examination of the physical response from the system to variables that are not constant and change with time.
- In dynamic testing the software must actually be compiled and run.

- It involves working with the software, giving input values and checking if the output is as expected by executing specific test cases which can be done manually or with the use of an automated process
- The process and function of dynamic testing in software development, dynamic testing can be divided into unit testing, integration testing, system testing, acceptance testing and finally regression testing.
- Unit testing is a test that focuses on the correctness of the basic components of software. Unit testing falls into the category of white-box testing. In the entire quality inspection system, unit testing needs to be completed by the product group, and then the software is handed over to the testing department.
- Integration testing is used to detect if the interfaces between the various units are properly connected during the integration process of the entire software.
- Testing a software system that has completed integration is called a system test, and the purpose of the test is to verify that the correctness and performance of the software system meet the requirements specified in its specifications. Testers should follow the established test plan. When testing the robustness and ease of use of the software, its input, output, and other dynamic operational behaviour should be compared to the software specifications. If the software specification is incomplete, the system test is more dependent on the tester's work experience and judgment, such a test is not sufficient. The system test is Black-box testing.
- This is the final test before the software is put into use. It is the buyer's trial process of the software. In the actual work of the company, it is usually implemented by asking the customer to try or release the Beta version of the software. The acceptance test is Black-box testing.
- The purpose of regression testing is to verify and modify the acceptance test results in the software maintenance phase. In practical applications, the handling of customer complaints is an embodiment of regression testing.

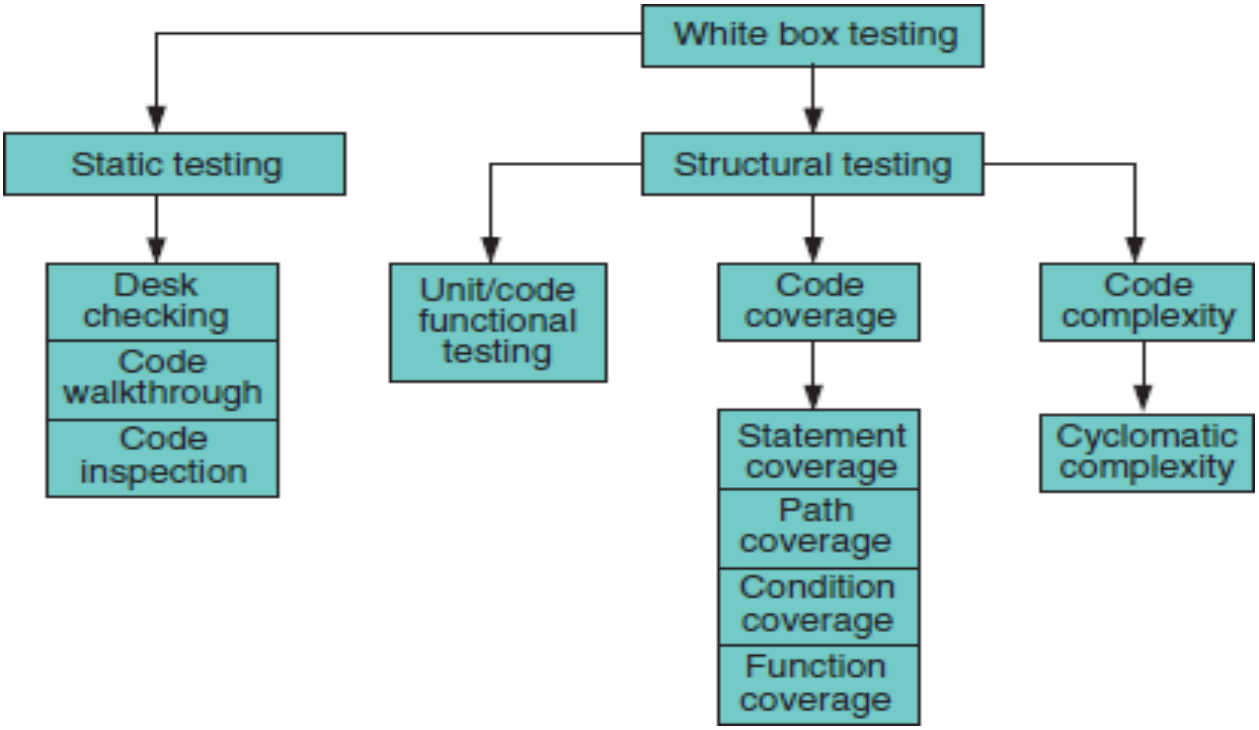
Advantages of Dynamic Testing

- Dynamic testing could identify the weak areas in the runtime environment.
- Dynamic testing supports application analysis even if the tester does not have an actual code.
- Dynamic testing could identify some vulnerabilities that are difficult to find by static testing.
- Dynamic testing also could verify the correctness of static testing results.
- Dynamic testing could be applied to any application.

Disadvantages of Dynamic Testing

- Automated tools may give the wrong security, such as check everything.
- Automated tools can generate false positives and false negatives.
- Finding trained dynamic test professionals is not easy.
- Dynamic testing is hard to track down the vulnerabilities in the code, and it takes longer to fix the problem. Therefore, fixing bugs becomes expensive.

White box testing



1) Walkthrough

- In walkthrough, author guides the review team via the document to fulfill the common understanding and collecting the feedback.
- Walkthrough is not a formal process.
- In walkthrough, a review team does not require to do detailed study before meeting while authors are already in the scope of preparation.
- Walkthrough is useful for higher-level documents i.e requirement specification and architectural documents.

Goals of Walkthrough

- Make the document available for the stakeholders both outside and inside the software discipline for collecting the information about the topic under documentation.
- Describe and evaluate the content of the document.
- Study and discuss the validity of possible alternatives and proposed solutions.

Participants of Structured Walkthrough

- **Author -** The Author of the document under review.
- **Presenter -** The presenter usually develops the agenda for the walkthrough and presents the output being reviewed.
- **Moderator -** The moderator facilitates the walkthrough session, ensures the walkthrough agenda is followed, and encourages all the reviewers to participate.

- **Reviewers -** The reviewers evaluate the document under test to determine if it is technically accurate.
- **Scribe -** The scribe is the recorder of the structured walkthrough outcomes who records the issues identified and any other technical comments, suggestions, and unresolved questions.

Benefits of Structured Walkthrough

- Saves time and money as defects are found and rectified very early in the lifecycle.
- This provides value-added comments from reviewers with different technical backgrounds and experience.
- It notifies the project management team about the progress of the development process.
- It creates awareness about different development or maintenance methodologies which can provide a professional growth to participants.

2) Inspection

- The trained moderator guides the Inspection. It is most formal type of review.
- The reviewers are prepared and check the documents before the meeting.
- In Inspection, a separate preparation is achieved when the product is examined and defects are found. These defects are documented in issue log.
- In Inspection, moderator performs a formal follow-up by applying exit criteria.

Goals of Inspection

- Assist the author to improve the quality of the document under inspection.
- Efficiently and rapidly remove the defects.
- Generating the documents with higher level of quality and it helps to improve the product quality.
- It learns from the previous defects found and prohibits the occurrence of similar defects.
- Generate common understanding by interchanging information.

Difference between Inspection and Walkthrough

Inspection	Walkthrough
Formal	Informal
Initiated by the project team	Initiated by the author
Planned meeting with fixed roles assigned to all the members involved	Unplanned.
Reader reads the product code. Everyone inspects it and comes up with defects.	Author reads the product code and his team mate comes up with defects or suggestions

Recorder records the defects	Author makes a note of defects and suggestions offered by team mate
Moderator has a role in making sure that the discussions proceed on the productive lines	Informal, so there is no moderator

3) Technical Review

- Technical review is a discussion meeting that focuses on technical content of the document. **It is a less formal review.**
- It is guided by a trained moderator or a technical expert.

Goals of Technical Review

- The goal is to evaluate the value of technical concept in the project environment.
- Build the consistency in the use and representation of the technical concepts.
- In early stages it ensures that the technical concepts are used correctly.
- Notify the participants regarding the technical content of the document.

Code Functional Testing:

i. Code Functional Testing involves tracking a piece of data completely through the software.
ii. At the unit test level this would just be through an individual module or function.
iii. The same tracking could be done through several integrated modules or even through the entire software product although it would be more time consuming to do so.
iv. During data flow, the check is made for the proper declaration of variables declared and the loops used are declared and used properly.

For example

```
1. #include<stdio.h>
2. void main()
3. {
4. int i , fact= 1, n;
6. scan
7. for(i =1; i<=n; i++)
8. fact = fact * i;
9.printf("\n Factorial of number is %d",fact);
10. }
```

Code Coverage Testing:

i. The logical approach is to divide the code just as you did in black-box testing into its data and its states (or program flow).
ii. By looking at the software from the same perspective, you can more easily map the white-box information you gain to the black-box case you have already written.

iii. Consider the data first. Data includes all the variables, constants, arrays, data structures, keyboard and mouse input, files and screen input and output, and I/O to other devices such as modems, networks, and so on.

For example

```
1. #include<stdio.h>
2. void main()
3. {
4. int i , fact= 1, n;
5. printf("Enter the number");
6. scanf("%d",&n);
7. for(i =1; i<=n; i++)
8. fact = fact * i;
9.printf("\n Factorial of number is %d",fact);
10. }
```

The declaration of data is complete with the assignment statement and the variable declaration statements. All the variable declared are properly utilized.

Program Statements and Line Coverage (Code Complexity Testing)

i. The most straightforward form of code coverage is called statement coverage or line coverage.

ii. If you're monitoring statement coverage while you test your software, your goal is to make sure that you execute every statement in the program at least once.

iii. With line coverage the tester tests the code line by line giving the relevant output.

For example

```
1. #include
2. void main()
3. {
4. int i , fact= 1, n;
5. printf(—enter the number —);
6. scanf(—%d‖, &n);
7. for(i =1 ;i <=n; i++)
8. fact = fact * i;
9. printf("\n Factorial of number is %d",fact);
10. }
```

Branch Coverage (Code Complexity Testing)

i. Attempting to cover all the paths in the software is called path testing.

ii. The simplest form of path testing is called branch coverage testing.

iii. To check all the possibilities of the boundary and the sub boundary conditions and it's branching on those values.

iv. Test coverage criteria requires enough test cases such that each condition in a decision takes on all possible outcomes at least once, and each point of entry to a program or subroutine is invoked at least once.

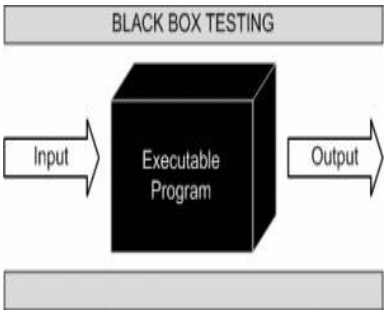
v. Every branch (decision) taken each way, true and false. vi. It helps in validating all the branches in the code making sure that no branch leads to abnormal behavior of the application.

Condition Coverage (Code Complexity Testing)

i. Just when you thought you had it all figured out, there's yet another Complication to path testing.
ii. Condition coverage testing takes the extra conditions on the branch statements into account.

Black Box Testing

- **Black Box Testing**, also known as Behavioral Testing, is a software testing method in which the internal structure/ design/ implementation of the item being tested is not known to the tester. These tests can be functional or non-functional, though usually functional



This method attempts to find errors in the following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors
- **EXAMPLE :** A tester, without knowledge of the internal structures of a website, tests the web pages by using a browser; providing inputs (clicks, keystrokes) and verifying the outputs against the expected outcome.

Advantages of black box testing

- Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.
- Tester need not know programming languages or how the software has been implemented.

- Tests can be conducted by a body independent from the developers, allowing for an objective perspective and the avoidance of developer-bias.
- Test cases can be designed as soon as the specifications are complete.

Disadvantages of black box testing

- Only a small number of possible inputs can be tested and many program paths will be left untested.
- Without clear specifications, which is the situation in many projects, test cases will be difficult to design.
- Tests can be redundant if the software designer/ developer has already run a test case.
- Ever wondered why a soothsayer closes the eyes when foretelling events? So is almost the case in Black Box Testing.

Techniques for black box testing

1. Requirement Based Testing
2. Boundary Value Analysis
3. Equivalence Partitioning

1) Requirement based testing

- Requirements-based testing is a testing approach in which test cases, conditions and data are derived from requirements. It includes functional tests and also non-functional attributes such as performance, reliability or usability.

Stages in Requirements based Testing:

- **Defining Test Completion Criteria -** Testing is completed only when all the functional and non-functional testing is complete.
- **Design Test Cases -** A Test case has five parameters namely the initial state or precondition, data setup, the inputs, expected outcomes and actual outcomes.
- **Execute Tests -** Execute the test cases against the system under test and document the results.
- **Verify Test Results -** Verify if the expected and actual results match each other.
- **Verify Test Coverage -** Verify if the tests cover both functional and non-functional aspects of the requirement.

- **Track and Manage Defects -** Any defects detected during the testing process goes through the defect life cycle and are tracked to resolution. Defect Statistics are maintained which will give us the overall status of the project.

2) Boundary Value Analysis

- For the most part, errors are observed in the extreme ends of the input values, so these extreme values like start/end or lower/upper values are called Boundary values and analysis of these Boundary values is called "Boundary value analysis". It is also sometimes known as 'range checking'.
- Boundary value analysis is used to find the errors at boundaries of input domain rather than finding those errors in the center of input.
- This is one of the software testing technique in which the test cases are designed to include values at the boundary. If the input data is used within the boundary value limits, then it is said to be Positive Testing. If the input data is picked outside the boundary value limits, then it is said to be Negative Testing.
- Boundary value analysis is another black box test design technique and it is used to find the errors at boundaries of input domain rather than finding those errors in the center of input.
- Each boundary has a valid boundary value and an invalid boundary value. Test cases are designed based on the both valid and invalid boundary values. Typically, we choose one test case from each boundary.

  Boundary value analysis is a black box testing and is also applies to white box testing. Internal data structures like arrays, stacks and queues need to be checked for boundary or limit conditions; when there are linked lists used as internal structures, the behavior of the list at the beginning and end have to be tested thoroughly.
- Boundary value analysis help identify the test cases that are most likely to uncover defects

 For example : Suppose you have very important tool at office, accepts valid User Name and Password field to work on that tool, and accepts minimum 8 characters and maximum 12 characters. Valid range 8-12, Invalid range 7 or less than 7 and Invalid range 13 or more than 13.

- Test Cases 1: Consider password length less than 8.
- Test Cases 2: Consider password of length exactly 8.
- Test Cases 3: Consider password of length between 9 and 11.
- Test Cases 4: Consider password of length exactly 12.
- Test Cases 5: Consider password of length more than 12.

Test cases for the application whose input box accepts numbers between 1-1000. Valid range 1-1000, Invalid range 0 and Invalid range 1001 or more.

- Test Cases 1: Consider test data exactly as the input boundaries of input domain i.e. values 1 and 1000.
- Test Cases 2: Consider test data with values just below the extreme edges of input domains i.e. values 0 and 999.
- Test Cases 3: Consider test data with values just above the extreme edges of input domain i.e. values 2 and 1001.

3) Equivalence Partitioning

- Equivalence partitioning is a software technique that involves identifying a small set of representative input values that produce as much different output condition as possible.
- This reduces the number of permutation & combination of input, output values used for testing, thereby increasing the coverage and reducing the effort involved in testing.
- The set of input values that generate one single expected output is called a partition.
- When the behavior of the software is the same for a set of values, then the set is termed as equivalence class or partition.
- Example: An insurance company that has the following premium rates based on the age group. A life insurance company has base premium of $0.50 for all ages. Based on the

age group, an additional monthly premium has to pay that is as listed in the table below. For example, a person aged 34 has to pay a premium=$0.50 +$ 1.65=$2.15

Age Group	Additional Premium
Under 35	$1.65
35-59	$2.87
60+	$6.00

- Based on the equivalence portioning technique, the equivalence partitions that are based on age are given below:

1. Below 35 years of age (valid input)
2. Between 35 and 59 years of age (valid input)
3. Above 6 years of age (valid input)
4. Negative age (invalid input)
5. Age as 0(invalid input)
6. Age as any three-digit number (valid input)