Chapter1

What is Testing?

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

According to ANSI/IEEE 1059 standard, Testing can be defined as - A process of analysing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

Who does Testing?

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, the following professionals are involved in testing a system within their respective capacities −

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have different designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, QA Analyst, etc.

When to Start Testing?

An early start to testing reduces the cost and time to rework and produce error-free software that is delivered to the client. However in Software Development Life Cycle (SDLC), testing can be started from the Requirements Gathering phase and continued till the deployment of the software.

It also depends on the development model that is being used. For example, in the Waterfall model, formal testing is conducted in the testing phase; but in the incremental model, testing is performed at the end of every increment/iteration and the whole application is tested at the end.

Testing is done in different forms at every phase of SDLC −

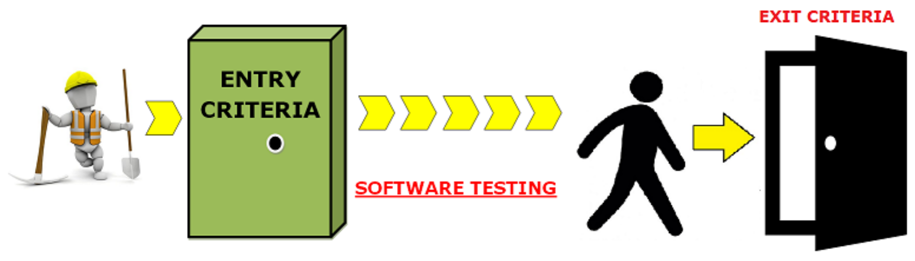
- During the requirement gathering phase, the analysis and verification of requirements are also considered as testing.
- Reviewing the design in the design phase with the intent to improve the design is also considered as testing.
- Testing performed by a developer on completion of the code is also categorized as testing.

When to Stop Testing?

It is difficult to determine when to stop testing, as testing is a never-ending process and no one can claim that a software is 100% tested. The following aspects are to be considered for stopping the testing process −

- Testing Deadlines
- Completion of test case execution
- Completion of functional and code coverage to a certain point
- Bug rate falls below a certain level and no high-priority bugs are identified
- Management decision

What is Entry and Exit Criteria?



What is An Entry Criteria in Software Testing?

Entry Criteria: Entry Criteria gives the prerequisite items that must be completed before testing can begin.

As the name specifies, entry criteria is a set of conditions or requirements, which are required to be fulfilled or achieved to create a suitable and favourable condition for testing. Finalized and decided upon after a thorough analysis of software and business requirements, entry criteria ensures the accuracy of the testing process and neglecting it can impact its quality. Some of the entry criteria, which are generally used to mark the beginning of the testing, are:

- Complete or partially testable code is available.
- Requirements are defined and approved.
- Availability of sufficient and desired test data.
- Test cases are developed and ready.
- Test environment has been set-up and all other necessary resources such as tools and devices are available.

Both, development and testing phases are used as a source to define the entry criteria for software testing process, like:

- Development phase/process provides useful information pertaining to software, its design, functionalities, structure, and other relevant features, which offer assistance in deciding the accurate entry criteria like functional and technical requirement, system design, etc.
- From testing phase, following inputs are considered:
    - Test Plan.
    - Test Strategy.
    - Test data and testing tools.
    - Test Environment.

The entry criteria is mainly determined for four specific test levels i.e., unit testing, integration testing, system testing and acceptance testing. Each of these test levels requires distinct entry criteria to validate the objective of test strategy and to ensure fulfilment of product requirements.

Unit Testing:

- Business requirements are present.
- Availability of functional requirements and specification.
- High level design document is available.
- Planning phase has been successfully completed.

Integration Testing

- All modules are unit tested and available for the integration.
- Code is completed.
- All priority bugs are fixed and closed.
- Availability integration **test plans**, **scenarios** and **cases**.

System Testing

- Integration of the module has been done successfully and passed the exit criteria of Integration testing.
- All the priority bugs have been fixed and closed.
- **System Testing** environment has been set up.
- **Test cases**/scripts are ready.

Acceptance Testing

- Successfully passing the exit criteria of the system testing.
- Business requirements had been met.
- **Test Environment** has been set up.
- Test cases/scripts are ready.

What are Exit Criteria in Software Testing?

Exit Criteria: Exit Criteria defines the items that must be completed before testing can be concluded

An exit criterion decides the completion or termination of the testing task. Exit Criteria is the condition of the set of conditions which imparts the completion of an activity or meeting of the targets and goals. Similar to entry criteria, **exit criteria** is also defined and outlined during the test planning phase. However, **exit criteria** are more difficult to define in comparison to the entry criteria.

Following are some of the possible **exit criteria** which may be used to mark the end of the testing activities.

- All **test cases** have been executed.
- Sufficient coverage of the requirements and functionalities has been achieved.

- All **high severity or top priority** bug has been fixed.
- High risk identified area has been taken up and tested.
- Budget gets depleted.
- Deadlines reached.

List of Some Exit Criteria in Software Testing

Unit Testing

- Successfully executed the unit tests.
- Priority bugs are fixed and closed.

Integration Testing

- No Outstanding critical defects.
- All priority bugs are fixed and closed.
- Integration system has passed the all functional and performance requirements.
- Install-ability has been verified.

System Testing

- All system test cases have been successfully executed.
- Priority bugs have been fixed and closed.
- System is meeting all sorts of business and functional requirements.
- System is supporting all its intended hardware and software.

Acceptance Testing

- Completed and approved by the **Test Management** team.
- Essential business process has been covered.
- No critical defect is left out.

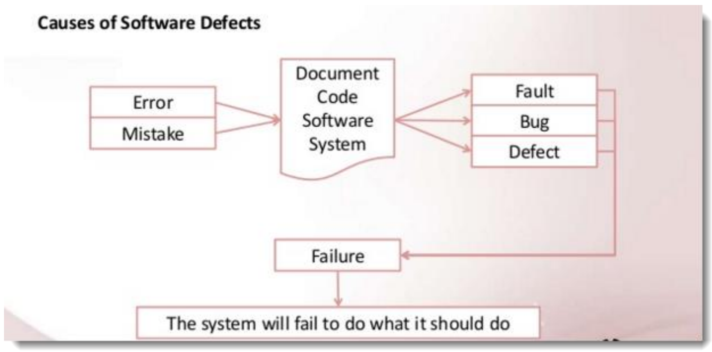
"Similar to entry criteria, the above defined exit criteria list may be further increased or decreased to meet the business needs and requirements within the stipulated deadlines"

Objectives of Software testing :

- **Finding defects** which may get created by the programmer while developing the software.
- Gaining confidence in and providing information about the level of **quality**.
- To prevent defects.
- To make sure that the end result meets the business and user requirements.
- To ensure that it satisfies the BRS that is Business Requirement Specification and SRS that is System Requirement Specifications.
- To gain the confidence of the customers by providing them a quality product.
- Ensures the quality of product
- Verify and validate the user requirement
- Focus on accurate and reliable result
- Discuss on generating test cases test cases
- Provide information to take decision for next phase
- Gain confidence of work
- Evaluates the capabilities of a system and system performance

Causes of software defects:

"A mistake in coding is called error ,error found by tester is called defect, defect accepted by development team then it is called bug ,build does not meet the requirements then it Is failure."


Causes of Software Defects
Error
Mistake
Document
Code
Software
System
Fault
Bug
Defect
Failure
The system will fail to do what it should do

If someone makes an **error** or ***mistake*** in using the software, this may lead directly to a problem – the software is used incorrectly and so does not behave as we expected. However, people also design and build the software and they can make mistakes during the design and build. These mistakes mean that there are ***flaws*** in the software itself. These are called ***defects*** or sometimes ***bugs*** or ***faults***.

When the software code has been built, it is executed and then any defects may cause the system to fail to do what it should do (or do something it shouldn't), causing a ***failure.*** Not all defects result in failures; some stay dormant in the code and we may never notice them.

What is an Error or Mistake?

Error is a human action that produces an incorrect result. **It is deviation from actual and expected value.** The mistakes made by programmer are known as an 'Error'. This could happen because of the following reasons

- Some confusion in understanding the requirement of the software
- Some miscalculation of the values
- Or/And Misinterpretation of any value, etc.

It represents mistake made by people and Mistake in the program leads to error.

What is a Bug?

A Bug is the result of a coding Error or Fault in the program which causes the program to behave in an unintended or unanticipated manner. It is an evidence of fault in the program. Bugs arise from mistakes and errors, made by people, in either a program's source code or its design. Normally, there are bugs in all useful computer programs, but well-written programs contain relatively few bugs, and these bugs typically do not prevent the program from performing its task.

What is a Defect or Fault?

A Defect is a deviation from the Requirements. A Software Defect is a condition in a software product which does not meet a software requirement (as stated in the requirement specifications) or end-user expectations. In other words, a defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected result. This could be hardware, software, network, performance, format, or functionality.

What is a Failure?

Failure is a deviation of the software from its intended purpose. It is the inability of a system or a component to perform its required functions within specified performance requirements. Failure occurs when fault executes.

Test Case

A **TEST CASE** is a set of conditions or variables under which a tester will determine whether a system under test satisfies requirements or works correctly.

The process of developing test cases can also help find problems in the requirements or design of an application.

Test Case Template

A test case can have the following elements. Note, however, that a test management tool is normally used by companies and the format is determined by the tool used.

Test Suite ID	The ID of the test suite to which this test case belongs.
Test Case ID	The ID of the test case.
Test Case Summary	The summary / objective of the test case.
Related Requirement	The ID of the requirement this test case relates/traces to.

Prerequisites	Any prerequisites or preconditions that must be fulfilled prior to executing the test.
Test Procedure	Step-by-step procedure to execute the test.
Test Data	The test data, or links to the test data, that are to be used while conducting the test.
Expected Result	The expected result of the test.
Actual Result	The actual result of the test; to be filled after executing the test.
Status	Pass or Fail. Other statuses can be 'Not Executed' if testing is not performed and 'Blocked' if testing is blocked.
Remarks	Any comments on the test case or test execution.
Created By	The name of the author of the test case.
Date of Creation	The date of creation of the test case.
Executed By	The name of the person who executed the test.
Date of Execution	The date of execution of the test.
Test Environment	The environment (Hardware/Software/Network) in which the test was executed.

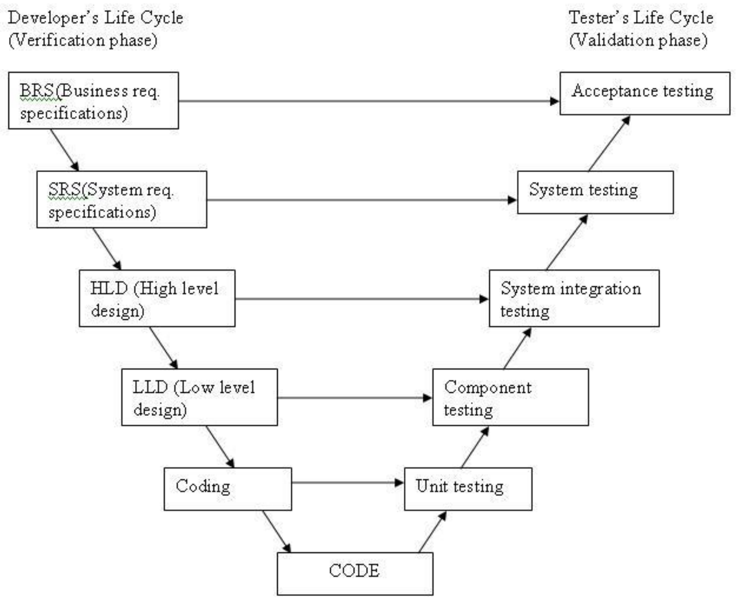
Test Case Example / Test Case Sample

Test Suite ID	TS001
Test Case ID	TC001
Test Case Summary	To verify that clicking the Generate Coin button generates coins.
Related Requirement	RS001
Prerequisites	1. User is authorized. 2. Coin balance is available.
Test Procedure	1. Select the coin denomination in the Denomination field. 2. Enter the number of coins in the Quantity field. 3. Click Generate Coin.
Test Data	1. Denominations: 0.05, 0.10, 0.25, 0.50, 1, 2, 5 2. Quantities: 0, 1, 5, 10, 20
Expected Result	1. Coin of the specified denomination should be produced if the specified Quantity is valid (1, 5) 2. A message 'Please enter a valid quantity between 1 and 10' should be displayed if the specified quantity is invalid.
Actual Result	1. If the specified quantity is valid, the result is as expected. 2. If the specified quantity is invalid, nothing happens; the expected message is not displayed
Status	Fail
Remarks	This is a sample test case.
Created By	John Doe

Date of Creation	01/14/2020
Executed By	Jane Roe
Date of Execution	02/16/2020
Test Environment	• OS: Windows Y • Browser: Chrome N

Verification and Validation

V- model means Verification and Validation model. Just like the **waterfall model**, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. **V-Model** is one of the **many software development models**.
Testing of the product is planned in parallel with a corresponding phase of development in **V-model**.
Diagram of V-model:



The various phases of the V-model are as follows:
Requirements like BRS and SRS begin the life cycle model just like the waterfall model. But, in this model before development is started, a **system test** plan is created. The **test plan** focuses on meeting the functionality specified in the requirements gathering.
The high-level design (HLD) phase focuses on system architecture and design. It provides overview of solution, platform, system, product and service/process. An **integration test** plan is created in this phase as well in order to test the pieces of the software systems ability to work together.
The low-level design (LLD) phase is where the actual software components are designed. It defines the actual logic for each and every component of the system. Class diagram with all the methods and relation between classes comes under LLD. **Component tests** are created in this phase as well.
The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.
Coding: This is at the bottom of the V-Shape model. Module design is converted into code by developers. **Unit Testing** is performed by the developers on the code written by them.

Advantages of V-model:

- Simple and easy to use.
- Testing activities like planning, **test designing** happens well before coding. This saves a lot of time. Hence higher chance of success over the waterfall model.
- Proactive defect tracking – that is defects are found at early stage.
- Avoids the downward flow of the defects.

- Works well for small projects where requirements are easily understood.

Disadvantages of V-model:

- Very rigid and least flexible.
- Software is developed during the implementation phase, so no early prototypes of the software are produced.
- If any changes happen in midway, then the test documents along with requirement documents has to be updated.

When to use the V-model:

- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-Shaped model should be chosen when ample technical resources are available with needed technical expertise.
- These two terms are very confusing for most people, who use them interchangeably. The following table highlights the differences between verification and validation.

Sr.No.	Verification	Validation
1	Verification addresses the concern: "Are you building it right?"	Validation addresses the concern: "Are you building the right thing?"
2	Ensures that the software system meets all the functionality.	Ensures that the functionalities meet the intended behaviour.
3	Verification takes place first and includes the checking for documentation, code, etc.	Validation occurs after verification and mainly involves the checking of the overall product.
4	Done by developers.	Done by testers.
5	It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify software.	It has dynamic activities, as it includes executing the software against the requirements.
6	It is an objective process and no subjective decision should be needed to verify software.	It is a subjective process and involves subjective decisions on how well a software works.

Testing, Quality Assurance, and Quality Control

Most people get confused when it comes to pin down the differences among Quality Assurance, Quality Control, and Testing. Although they are interrelated and to some extent, they can be considered as same activities, but there exist distinguishing points that set them apart. The following table lists the points that differentiate QA, QC, and Testing.

Quality Assurance	Quality Control	Testing
QA includes activities that ensure the implementation of processes, procedures and standards in context to verification of developed software and intended requirements.	It includes activities that ensure the verification of a developed software with respect to documented (or not in some cases) requirements.	It includes activities that ensure the identification of bugs/error/defects in software.
Focuses on processes and procedures rather than conducting actual testing on the system.	Focuses on actual testing by executing the software with an aim to identify bug/defect through implementation of procedures and process.	Focuses on actual testing.
Process-oriented activities.	Product-oriented activities.	Product-oriented activities.
Preventive activities.	It is a corrective process.	It is a preventive process.
It is a subset of Software Test Life Cycle (STLC).	QC can be considered as the subset of Quality Assurance.	Testing is the subset of Quality Control.

What is Quality?

Quality is extremely hard to define, and it is simply stated: "Fit for use or purpose." It is all about meeting the needs and expectations of customers with respect to functionality, design, reliability, durability, & price of the product.

What is Assurance?

Assurance is nothing but a positive declaration on a product or service, which gives confidence. It is certainty of a product or a service, which it will work well. It provides a guarantee that the product will work without any problems as per the expectations or requirements.

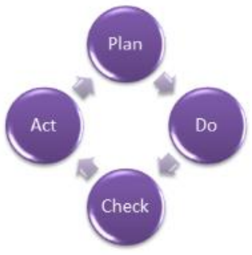
What is Quality Assurance?

Quality Assurance is popularly known as QA Testing, is defined as an activity to ensure that an organization is providing the best possible product or service to customers. QA focuses on improving the processes to deliver Quality Products to the customer. An organization has to ensure, that processes are efficient and effective as per the quality standards defined for software products.

How to do Quality Assurance: Complete Process

Quality assurance has a defined cycle called PDCA cycle or Deming cycle. The phases of this cycle are:

- Plan
- Do
- Check
- Act

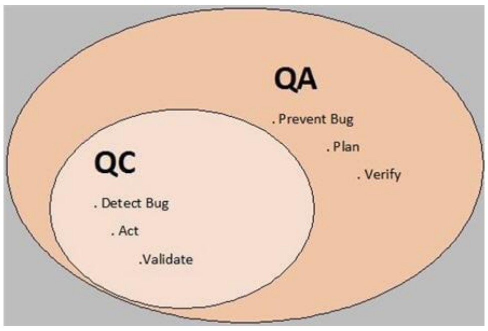


These above steps are repeated to ensure that processes followed in the organization are evaluated and improved on a periodic basis. Let's look into the above steps in detail -

- Plan - Organization should plan and establish the process related objectives and determine the processes that are required to deliver a high-Quality end product.
- Do - Development and testing of Processes and also "do" changes in the processes
- Check - Monitoring of processes, modify the processes, and check whether it meets the predetermined objectives
- Act - Implement actions that are necessary to achieve improvements in the processes

An organization must use Quality Assurance to ensure that the product is designed and implemented with correct procedures. This helps reduce problems and errors, in the final product.

What is Quality Control?



Quality control popularly abbreviated as QC. It is a Software Engineering process used to ensure quality in a product or a service. It does not deal with the processes used to create a product; rather it examines the quality of the "end products" and the final outcome.

The main aim of Quality control is to check whether the products meet the specifications and requirements of the customer. If an issue or problem is identified, it needs to be fixed before delivery to the customer.

QC also evaluates people on their quality level skill sets and imparts training and certifications. This evaluation is required for the service based organization and helps provide "perfect" service to the customers.

Difference between Quality Control and Quality Assurance?

Sometimes, QC is confused with the QA. Quality control is to examine the product or service and check for the result. Quality assurance is to examine the processes and make changes to the processes which led to the end-product.
Examples of QC and QA activities are as follows:

Quality Control Activities	**Quality Assurance Activities**
Walkthrough	Quality Audit
Testing	Defining Process
Inspection	Tool Identification and selection
Checkpoint review	Training of Quality Standards and Processes

Differences between SQA and Software Testing

Following table explains on differences between SQA and Software Testing:

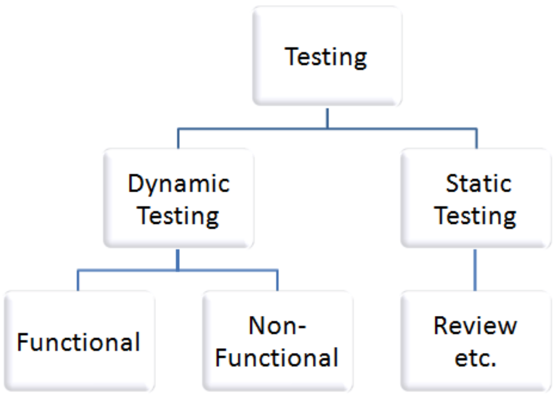
SQA	**Software Testing**
Software Quality Assurance is about engineering process that ensures quality	Software Testing is to test a product for problems before the product goes live
Involves activities related to the implementation of processes, procedures, and standards. Example - Audits Training	Involves actives concerning verification of product Example - Review Testing
Process focused	Product focused
Preventive technique	Corrective technique
Proactive measure	Reactive measure
The scope of SQA applied to all products that will be created by the organization	The scope of Software Testing applies to a particular product being tested.

Difference between Quality Assurance (QA) and Quality Control (QC)

Quality Assurance (QA)	**Quality Control (QC)**
• It is a procedure that focuses on providing assurance that quality requested will be achieved	It is a procedure that focuses on fulfilling the quality requested.
• QA aims to prevent the defect	• QC aims to identify and fix defects
• It is a method to manage the quality- Verification	• It is a method to verify the quality-Validation
• It does not involve executing the program	• It always involves executing a program
• It's a Preventive technique	• It's a Corrective technique

• It's a Proactive measure	• It's a Reactive measure
• It is the procedure to create the deliverables	• It is the procedure to verify that deliverables
• QA involves in full software development life cycle	• QC involves in full software testing life cycle
• In order to meet the customer requirements, QA defines standards and methodologies	• QC confirms that the standards are followed while working on the product
• It is performed before Quality Control	• It is performed only after QA activity is done
• It is a Low-Level Activity, it can identify an error and mistakes which QC cannot	• It is a High-Level Activity, it can identify an error that QA cannot
• Its main motive is to prevent defects in the system. It is a less time-consuming activity	• Its main motive is to identify defects or bugs in the system. It is a more time-consuming activity
• QA ensures that everything is executed in the right way, and that is why it falls under verification activity	• QC ensures that whatever we have done is as per the requirement, and that is why it falls under validation activity
• It requires the involvement of the whole team	• It requires the involvement of the Testing team
• The statistical technique applied on QA is known as SPC or Statistical Process Control (SPC)	• The statistical technique applied to QC is known as SQC or Statistical Quality Control

Methods of testing:



What is Static Testing?

Under **Static Testing**, code is not executed. Rather it manually checks the code, requirement documents, and design documents to find errors, hence, the name "static".

Static Testing is type of testing in which the code is not executed. It can be done manually or by a set of tools. This type of testing checks the code, requirement documents and design documents and puts review comments on the work document .With static testing, we try to find out the errors, code flaws and potentially malicious code in the software application. It starts earlier in development life cycle and hence it is also called verification testing. Static testing can be done on work documents like requirement specifications, design documents, source code, test plans, test scripts and test cases, web page content.

The main objective of this testing is to improve the quality of software products by finding errors in the early stages of the development cycle. This testing is also called a Non-execution technique or verification testing.

Static testing involves manual or automated reviews of the documents. This review is done during an initial phase of testing to catch Defect early in STLC. It examines work documents and provides review comments

Examples of Work documents-

- Requirement specifications

- Design document
- Source Code
- Test Plans
- Test Cases
- Test Scripts
- Help or User document
- Web Page content

Static Testing Techniques:

Types of Reviews

Main types of review are as follows:

1) Walkthrough
2) Inspection
3) Technical Review

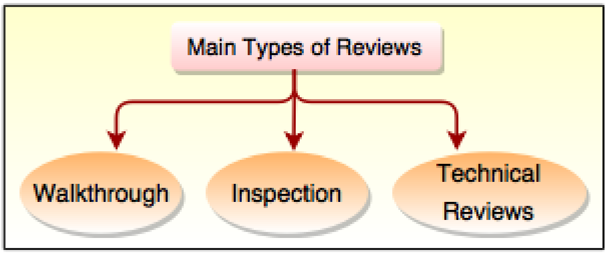


Fig. - Main Types of Review

These types of review comes under static testing technique.

1) Walkthrough

- In walkthrough, author guides the review team via the document to fulfil the common understanding and collecting the feedback.
- Walkthrough is **not a formal process.**
- The author of the work product explains the product to his team. Participants can ask questions if any. A meeting is led by the author. Scribe makes note of review comments
- 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.

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.
- The main purpose is to find defects and meeting is led by a trained moderator. This review is a formal type of review where it follows a strict process to find the defects. Reviewers have a checklist to review the work products. They record the defect and inform the participants to rectify those errors.
- 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.

3) Technical Review

- Technical review is a discussion meeting that focuses on technical content of the document. **It is a less formal review.**
- A team consisting of your peers review the technical specification of the software product and checks whether it is suitable for the project. They try to find any discrepancies in the specifications and standards followed. This review concentrates mainly on the technical documentation related to the software such as Test Strategy, Test Plan and requirement specification documents.
- 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.

4) **Informal Reviews:** This is one of the types of review which doesn't follow any process to find errors in the document. Under this technique, you just review the document and give informal comments on it.
   - Informal reviews are applied in the early stages of the life cycle of the document.
   - These reviews are conducted between two person team. In later stages more people are involved.
   - The aim of informal reviews is to improve the quality of the document and help the authors.
   - These reviews are not based on the procedure and not documented.

Advantages of Static Testing:

1. Helps in identifying the flaws in code
2. The testing is conducted by trained software developers with good knowledge of coding
3. It is fast and easy way to find and fix the errors
4. With automated tools, it becomes quite fast to scan and review the software
5. The use of Automated tools provides mitigation recommendations
6. With static testing it is possible to find errors at an early stage of development life cycle, thus, in turn, reduces the cost of fixing.

Disadvantages of Static Testing:

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

What is Dynamic Testing?

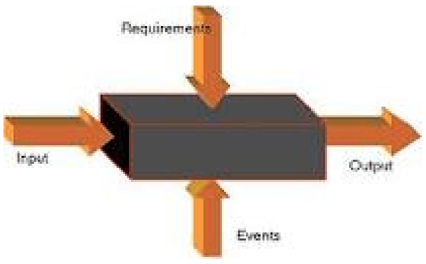
Under Dynamic Testing, a code is executed. It checks for functional behaviour of software system, memory/cpu usage and overall performance of the system. Hence the name "Dynamic"

The main objective of this testing is to confirm that the software product works in conformance with the business requirements. This testing is also called an Execution technique or validation testing.

Dynamic testing executes the software and validates the output with the expected outcome. Dynamic testing is performed at all levels of testing and it can be either black or white box testing.

Dynamic testing is done when the code is in operation mode. Dynamic testing is performed in runtime environment. When the code being executed is input with a value, the result or the output of the code is checked and compared with the expected output. With this we can observe the functional behaviour of the software, monitor the system memory, CPU response time, performance of the system. Dynamic testing is also known as validation testing, evaluating the finished product. Dynamic testing is of two types: Functional Testing and Non-functional testing.

Dynamic Testing Techniques:



- **Unit Testing:** Under Unit Testing, individual units or modules are tested by the developers. It involves testing of source code by developers.
- **Integration Testing:** Individual modules are grouped together and tested by the developers. The purpose is to determine what modules are working as expected once they are integrated.
- **System Testing:** System Testing is performed on the whole system by checking whether the system or application meets the requirement specification document.

Advantages of Dynamic code analysis

1. Dynamic coding helps in identifying weak areas in a run-time environment
2. Dynamic testing supports analysis of applications even if the tester does not have the actual code.
3. It identifies weak areas that are hard to be found with static code analysis
4. It allows validating static code analysis findings
5. It can be applied with any application

Dynamic code analysis limitations:

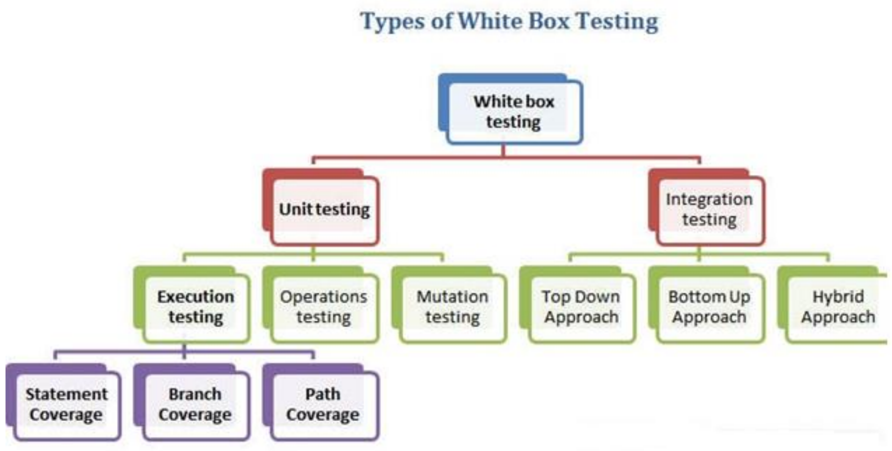
1. Automated tools may give a false security that everything is checked
2. Automated tools can generate false positives and false negatives
3. It is not easy to find a trained professional for dynamic testing
4. It is difficult to trace the vulnerability in the code, and it takes longer to fix the problem. Thus, it becomes costly to fix the errors

Difference between Static Testing and Dynamic Testing

Static Testing	Dynamic Testing
1. Static Testing is white box testing which is done at early stage if development life cycle. It is more cost effective than dynamic testing	1. Dynamic Testing on the other hand is done at the later stage of development lifecycle.
2. Static testing has more statement coverage than dynamic testing in shorter time	2. Dynamic Testing has less statement stage because it is covers limited area of code
3. It is done before code deployment	3. It is done after code deployment
4. It is performed in Verification Stage	4. It is done in Validation Stage
5. This type of testing is done without the execution of code.	5. This type of execution is done with the execution of code.
6. Static testing gives assessment of code as well as documentation.	6. Dynamic Testing gives bottlenecks of the software system.
7. In Static Testing techniques a checklist is prepared for testing process	7. In Dynamic Testing technique the test cases are executed.
8. Static Testing Methods include Walkthroughs, code review.	8. Dynamic testing involves functional and non-functional testing

The Box approach:

- It is also called as Glass Box, Clear Box, and Structural Testing.
- White Box Testing is based on applications internal code structure. In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. This testing usually done at the unit level.
- White box testing strategy deals with the internal logic and structure of the code. White box testing is also called as glass, structural, open box or clear box testing. The tests written based on the white box testing strategy incorporate coverage of the code written, branches, paths, statements and internal logic of the code etc.
- In order to implement white box testing, the tester has to deal with the code and hence is needed to possess knowledge of coding and logic i.e. internal working of the code. White box test also needs the tester to look into the code and find out which unit/statement/chunk of the code is malfunctioning.



Unit Testing:
The developer carries out unit testing in order to check if the particular module or unit of code is working fine. The Unit Testing comes at the very basic level as it is carried out as and when the unit of the code is developed or a particular functionality is built.

Static and dynamic Analysis:
Static analysis involves going through the code in order to find out any possible defect in the code. Dynamic analysis involves executing the code and analysing the output.

Statement Coverage:
In this type of testing the code is executed in such a manner that every statement of the application is executed at least once. It helps in assuring that all the statements execute without any side effect.

Branch Coverage:
No software application can be written in a continuous mode of coding, at some point we need to branch out the code in order to perform a particular functionality. Branch coverage testing helps in validating of all the branches in the code and making sure that no branching leads to abnormal behavior of the application.

Security Testing:
Security Testing is carried out in order to find out how well the system can protect itself from unauthorized access, hacking – cracking, any code damage etc. which deals with the code of application. This type of testing needs sophisticated testing techniques.

Mutation Testing:
A kind of testing in which, the application is tested for the code that was modified after fixing a particular bug/defect. It also helps in finding out which code and which strategy of coding can help in developing the functionality effectively.
Besides all the testing types given above, there are some more types which fall under both Black box and White box testing strategies such as: Functional testing (which deals with the code in order to check its functional performance), Incremental integration testing (which deals with the testing of newly added code in the application), Performance and Load testing (which helps in finding out how the particular code manages resources and give performance etc.) etc.
Advantages of White box testing are:

- As the knowledge of internal coding structure is prerequisite, it becomes very easy to find out which type of input/data can help in testing the application effectively.
- The other advantage of white box testing is that it helps in optimizing the code
- It helps in removing the extra lines of code, which can bring in hidden defects.

Disadvantages of white box testing are:

- As knowledge of code and internal structure is a prerequisite, a skilled tester is needed to carry out this type of testing, which increases the cost.
- And it is nearly impossible to look into every bit of code to find out hidden errors, which may create problems, resulting in failure of the application.

1. **INSPECTION**

- It is the most formal review type
- It is led by the trained moderators
- During inspection the documents are prepared and checked thoroughly by the reviewers before the meeting
- It involves peers to examine the product
- A separate preparation is carried out during which the product is examined and the **defects are found**
- The defects found are documented in a logging list or issue log
- A formal follow-up is carried out by the moderator applying exit criteria

goals of inspection are:

1. It helps the author to improve the quality of the document under inspection
2. It removes defects efficiently and as early as possible
3. It improve product quality
4. It create common understanding by exchanging information
5. It learn from defects found and prevent the occurrence of similar defects

Advantages of Inspection:

- It helps author to improve the quality of artefacts
- Help to remove defects in early stage
- Improves product quality
- It brings all team members at one page
- It helps to prevent the appearance of same kind of defeats again and again.

2. **WALKTHROUGH**

- Walkthrough in software testing is used to review documents with peers, managers, and fellow team members who are guided by the author of the document to gather feedback and reach a consensus.
- A walkthrough can be pre-planned or organised based on the needs. Generally people working on the same work product are involved in the walkthrough process.
- This review becomes more beneficial for those people who are away from software sphere and through this meeting they get a good insight about the product that is to be developed. The content is being explained by the author step by step to reach a common objective.
- The audience is selected from different backgrounds in order to have a diverse point of view and thus, provide different dimensions to a common objective. This is not a formal process but is specifically used for high level documents like requirement specifications or functional specifications, etc.

Goals of walkthrough are:

• Gather information regarding the topic in the document by involving stakeholders, both within and outside the software discipline.

• Describe and justify the contents of the document.
• Reach a common consensus on the document.
• Check and discuss the different solutions to a problem and different suggested alternatives.

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

- It is less formal review
- It is led by the trained moderator but can also be led by a technical expert
- It is often performed as a peer review without management participation
- **Defects** are found by the experts (such as architects, designers, key users) who focus on the content of the document.
- In practice, technical reviews vary from quite informal to very formal

The goals of the technical review are:

- To ensure that an early stage the technical concepts are used correctly
- To access the value of technical concepts and alternatives in the product
- To have consistency in the use and representation of technical concepts
- To inform participants about the technical content of the document

4. FUNCTIONAL TESTING

Functional Testing is defined as a type of testing which verifies that each **function** of the software application operates in conformance with the requirement specification. This testing mainly involves black box testing and it is not concerned about the source code of the application.
Each and every functionality of the system is tested by providing appropriate input, verifying the output and comparing the actual results with the expected results.
This testing involves checking of User Interface, APIs, Database, security, client/ server applications and functionality of the Application Under Test. The testing can be done either manually or using automation

What do you test in Functional Testing?

The prime objective of Functional testing is checking the functionalities of the software system. It mainly concentrates on -

- **Mainline functions**: Testing the main functions of an application
- **Basic Usability**: It involves basic usability testing of the system. It checks whether a user can freely navigate through the screens without any difficulties.
- **Accessibility**: Checks the accessibility of the system for the user
- **Error Conditions**: Usage of testing techniques to check for error conditions. It checks whether suitable error messages are displayed.

Functional testing involves the following steps:

1. Identify functions that the software is expected to perform.
2. The creation of input data based on the function's specifications.

3. To determine the output based on the function's specifications.
4. Execute the test case.
5. Compare the actual and expected outputs.

Types of Functional Testing:

- ***Unit Testing:*** Individually and independently testing of smallest testable parts of an application.
- ***Integration Testing:*** When individual software modules are combined together and tested as a group than it is known as Integration Testing.
- ***Smoke Testing:*** Preliminary testing to reveal simple failures severe enough to (for example) reject a prospective software release or build.
- ***Sanity Testing:*** Very brief run-through of the functionalities to assure that part of the system or methodology works roughly as expected.
- ***System Testing:*** Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.
- ***Regression Testing:*** Retesting of a software system to confirm that changes made to few parts of the codes has not any side effects on existing system functionalities
- ***Acceptance Testing:***

Advantages

- It simulates actual system usage.
- It does not make any system structure assumptions.

Disadvantages

- It has a potential of missing logical errors in software.
- It has a high possibility of redundant testing.

.

Functional Vs Non-Functional Testing:

Functional Testing	**Non-Functional Testing**
Functional testing is performed using the functional specification provided by the client and verifies the system against the functional requirements.	Non-Functional testing checks the Performance, reliability, scalability and other non-functional aspects of the software system.
Functional testing is executed first	Non-functional testing should be performed after functional testing
Manual Testing or automation tools can be used for functional testing	Using tools will be effective for this testing
Business requirements are the inputs to functional testing	Performance parameters like speed, scalability are inputs to non-functional testing.
Functional testing describes what the product does	Non-functional testing describes how good the product works
Easy to do Manual Testing	Tough to do Manual Testing
Examples of Functional testing are • Unit Testing • Smoke Testing • Sanity Testing • Integration Testing • White box testing • Black Box testing • User Acceptance testing • Regression Testing	Examples of Non-functional testing are • Performance Testing • Load Testing • Volume Testing • Stress Testing • Security Testing • Installation Testing • Penetration Testing • Compatibility Testing • Migration Testing

Functional Testing Tools

There are several tools available in the marker to perform functional testing. They are explained as follows:

- Selenium - Popular Open Source Functional Testing Tool
- QTP - Very user-friendly Functional Test tool by HP
- JUnit- Used mainly for Java applications and this can be used in Unit and System Testing
- soapUI - This is an open source functional testing tool, mainly used for Web service testing. It supports multiple protocols such as HTTP, SOAP, and JDBC.
- Watir - This is a functional testing tool for web applications. It supports tests executed at the web browser and uses a ruby scripting language

.

What is Code coverage?

Code coverage is a measure which describes the degree of which the source code of the program has been tested. It is one form of white box testing which finds the areas of the program not exercised by a set of test cases. It also creates some test cases to increase coverage and determining a quantitative measure of code coverage.

In most cases, code coverage system gathers information about the running program. It also combines that with source code information to generate a report about the test suite's code coverage.

Why use Code Coverage?

Reasons for using code coverage:

- It helps you to measure the efficiency of test implementation
- It offers a quantitative measurement.
- It defines the degree to which the source code has been tested.

Code Coverage Methods

Following are major code coverage methods

- Statement Coverage
- Condition Coverage
- Branch Coverage
- Toggle Coverage
- FSM Coverage

Statement Coverage

Statement coverage is a white box test design technique which involves execution of all the executable statements in the source code at least once. It is used to calculate and measure the number of statements in the source code which can be executed given the requirements.

Statement coverage is used to derive scenario based upon the structure of the code under test.

$$\text{Statement Coverage} = \frac{\text{Number of executed statements}}{\text{Total number of statements}} \times 100$$

In White Box Testing, the tester is concentrating on how the software works. In other words, the tester will be concentrating on the internal working of source code concerning control flow graphs or flow charts.

Generally in any software, if we look at the source code, there will be a wide variety of elements like operators, functions, looping, exceptional handlers, etc. Based on the input to the program, some of the code statements may not be executed. The goal of Statement coverage is to cover all the possible path's, line, and statement in the code.

Let's understand this with an example, how to calculate statement coverage.

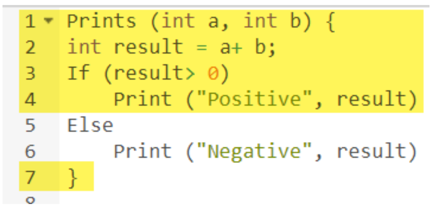
Scenario to calculate Statement Coverage for given source code. Here we are taking two different scenarios to check the percentage of statement coverage for each scenario.

Source Code:

```
prints (int a, int b) {                ------------  Prints is a function
   int result = a+ b;
   if (result> 0)
              print ("Positive", result)
   else
              print ("Negative", result)
   }                            -----------   End of the source code
```

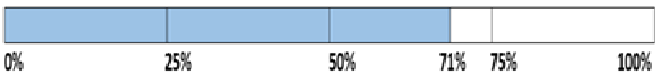
Scenario 1:

If A = 3, B = 9



```
Prints (int a, int b) {
int result = a+ b;
If (result> 0)
    Print ("Positive", result)
Else
    Print ("Negative", result)
}
```

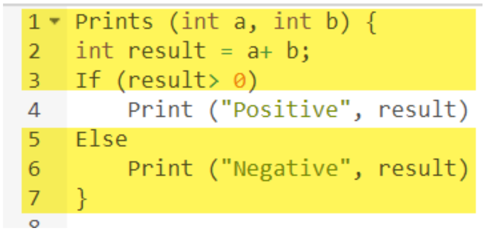
The statements marked in yellow colour are those which are executed as per the scenario
Number of executed statements = 5, Total number of statements = 7
Statement Coverage: 5/7 = 71%



Likewise we will see scenario 2,

Scenario 2:

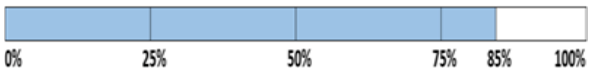
If A = -3, B = -9



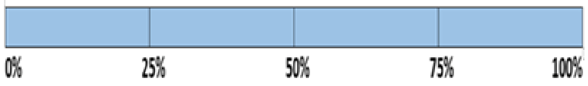
```
Prints (int a, int b) {
int result = a+ b;
If (result> 0)
    Print ("Positive", result)
Else
    Print ("Negative", result)
}
```

The statements marked in yellow color are those which are executed as per the scenario.
Number of executed statements = 6
Total number of statements = 7
Statement Coverage: 6/7 = 85%

$$\text{Statement Coverage} = \frac{\text{Number of executed statments}}{\text{Total number of statments}}$$



But overall if you see, all the statements are being covered by 2nd scenario is considered. So we can conclude that overall statement coverage is 100%.



What is covered by Statement Coverage?

1. Unused Statements
2. Dead Code
3. Unused Branches

Decision Coverage

Decision coverage reports the true or false outcomes of each Boolean expression. In this coverage, expressions can sometimes get complicated. Therefore, it is very hard to achieve 100% coverage.
That's why there are many different methods of reporting this metric. All these methods focus on covering the most important combinations. It is very much similar to decision coverage, but it offers better sensitivity to control flow.

$$Decision\ Coverage = \frac{Number\ of\ Decision\ Outcomes\ Excercised}{Total\ Number\ of\ Decision\ Outcomes}$$

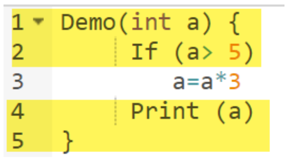
Example of decision coverage

Consider the following code-

```
Demo(int a) {
   If (a> 5)
           a=a*3
   Print (a)
}
```

Scenario 1:

Value of a is 2



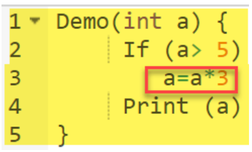
```
Demo(int a) {
    If (a> 5)
        a=a*3
    Print (a)
}
```

The code highlighted in yellow will be executed. Here the "No" outcome of the decision If (a>5) is checked.

Decision Coverage = 50%

Scenario 2:

Value of a is 6



```
Demo(int a) {
    If (a> 5)
        a=a*3
    Print (a)
}
```

The code highlighted in yellow will be executed. Here the "Yes" outcome of the decision If (a>5) is checked.

Decision Coverage = 50%

Test Case	Value of A	Output	Decision Coverage
1	2	2	50%
2	6	18	50%

Branch Coverage

In the branch coverage, every outcome from a code module is tested. For example, if the outcomes are binary, you need to test both True and False outcomes.

It helps you to ensure that every possible branch from each decision condition is executed at least a single time.

By using Branch coverage method, you can also measure the fraction of independent code segments. It also helps you to find out which is sections of code don't have any branches.

The formula to calculate Branch Coverage:

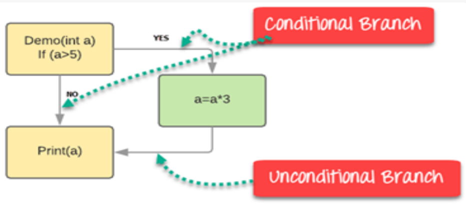
$$Branch\ Coverage = \frac{Number\ of\ Executed\ Branches}{Total\ Number\ of\ Branches}$$

Example of Branch Coverage

To learn branch coverage, let's consider the same example used earlier

Consider the following code

```
Demo(int a) {
   If (a> 5)
             a=a*3
   Print (a)
  }
```



Branch Coverage will consider unconditional branch as well

Test Case	Value of A	Output	Decision Coverage	Branch Coverage
1	2	2	50%	**33%**

2	6	18	50%	**67%**

Advantages of Branch coverage:

Branch coverage Testing offers the following advantages:

- Allows you to validate-all the branches in the code
- Helps you to ensure that no branched lead to any abnormality of the program's operation
- Branch coverage method removes issues which happen because of statement coverage testing
- Allows you to find those areas which are not tested by other testing methods
- It allows you to find a quantitative measure of code coverage
- Branch coverage ignores branches inside the Boolean expressions

Condition Coverage

Conditional coverage or expression coverage will reveal how the variables or sub expressions in the conditional statement are evaluated. In this coverage expressions with logical operands are only considered.

For example, if an expression has Boolean operations like AND, OR, XOR, which indicated total possibilities.

Conditional coverage offers better sensitivity to the control flow than decision coverage. Condition coverage does not give a guarantee about full decision coverage

The formula to calculate Condition Coverage:

$$Condition\ Coverage = \frac{Number\ of\ Executed\ Operands}{Total\ Number\ of\ Operands}$$

Example:

```
IF (x < y) AND (a>b) THEN
```

For the above expression, we have 4 possible combinations

- TT
- FF
- TF
- FT

Consider the following input

X=3
Y=4
	(x<y)	TRUE	Condition Coverage is ¼ = 25%

A=3
B=4
	(a>b)	FALSE	

Finite State Machine Coverage

Finite state machine coverage is certainly the most complex type of code coverage method. This is because it works on the behaviour of the design. In this coverage method, you need to look for how many time-specific states are visited, transited. It also checks how many sequences are included in a finite state machine.

Which Type of Code Coverage to Choose

This is certainly the most difficult answer to give. In order to select a coverage method, the tester needs to check that the

- code under test has single or multiple undiscovered defects
- cost of the potential penalty
- cost of lost reputation
- cost of lost sale, etc.

The higher the probability that defects will cause costly production failures, the more severe the level of coverage you need to choose.

Code Coverage vs. Functional Coverage

Code Coverage	**Functional Coverage**
Code coverage tells you how well the source code has been exercised by your test bench.	Functional coverage measures how well the functionality of the design has been covered by your test bench.

Never use a design specification	Use design specification
Done by developers	Done by Testers

Code Coverage Tools

Here, is a list of Important code coverage Tools:

Tool Name	**Description**
Coco	It is an Cross-platform and cross-compiler code coverage analysis for C, C++, SystemC, C#, Tcl and QML code. Automated measurement of test coverage of statements, branches and conditions. No changes to the application are necessary Learn more about coco
Parasoft Jtest	It accelerates Java software development by providing a set of tools to maximize quality and minimize business risks. Results from static analysis, JUnit tests, and code coverage are efficiently integrated with functional and manual testing results. Learn more about Parasoft Jtest
Cobertura	It is an open source code coverage tool. It measures test coverage by instrumenting a code base and analyze which lines of code are executing and which are not executed when the test suite runs.
Clover	Clover also reduces testng time by only running the tests which cover the application code which was modified since the previous build.
DevPartner	DevPartner enables developers to analyze Java code for Code Quality and Complexity.
Emma	EMMA supports class, method, line, and base block coverage, aggregated source file, class, and method levels.
JTest	JT's tool helps you to check functionalities like unit test-case generation, static analysis, regression testing, and code review.
Kalistick	Kalistick is a third party application which analyzes the codes with different perspectives.
CoView and CoAnt	Coding Software is a code coverage tool for metrics, mock object creation, code testability, path & branch coverage, etc.
Bullseye for C++	BulseyeCoverage is a code coverage tool for C++and C.
Sonar	Sonar is an open code coverage tool which helps you to manage code quality.

Advantages of Using Code Coverage

- Helpful to evaluate a quantitative measure of code coverage
- It allows you to create extra test cases to increase coverage
- It allows you to find the areas of a program which is not exercised by a set of test cases

Disadvantages of Using Code Coverage

- Even when any specific feature is not implemented in design, code coverage still report 100% coverage.
- It is not possible to determine whether we tested all possible values of a feature with the help of code coverage
- Code coverage is also not telling how much and how well you have covered your logic
- In the case when the specified function hasn't implemented, or a not included from the specification, then structure-based techniques cannot find that issue.

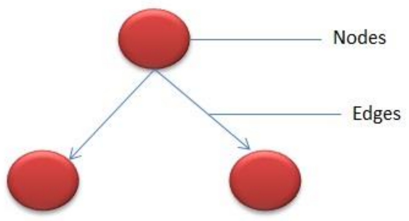
Code complexity testing:

What Is Cyclomatic Complexity?

Cyclomatic complexity (CYC) is a metric for software quality. It was developed by Thomas J. McCabe Sr. in 1976. In its simplest form, CYC is a count of the number of decisions in the source code. The higher the count, the more complex the code.

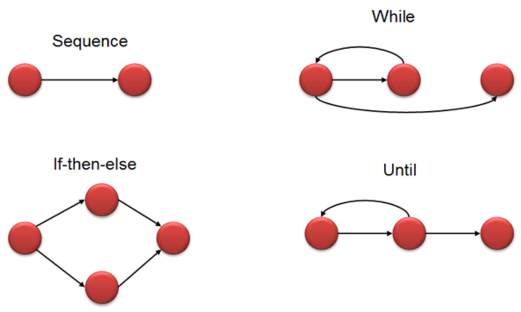
Cyclomatic complexity is a software metric used to measure the complexity of a program. These metric, measures independent paths through program source code. Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths. Cyclomatic complexity can be calculated with respect to functions, modules, methods or classes within a program.

In the graph, Nodes represent processing tasks while edges represent control flow between the nodes.



Flow graph notation for a program:

Flow Graph notation for a program is defines. several nodes connected through the edges. Below are Flow diagrams for statements like if-else, While, until and normal sequence of flow.



How to Calculate Cyclomatic Complexity

Mathematical representation:

Mathematically, it is set of independent paths through the graph diagram. The Code complexity of the program can be defined using the formula –

$$V(G) = E - N + 2P$$

In this equation:

- P = number of disconnected parts of the flow graph (e.g. a calling program and a subroutine)
- E = number of edges (transfers of control)
- N = number of nodes (sequential group of statements containing only one transfer of control)

This translates to the number of decisions + one.

Binary decisions — such as "if" and "while" statements — add one to complexity.

Boolean operators can add either one or nothing to complexity. For instance, one may be added if a Boolean operator is found within a conditional statement.

Properties of Cyclomatic complexity:

Following are the properties of Cyclomatic complexity:

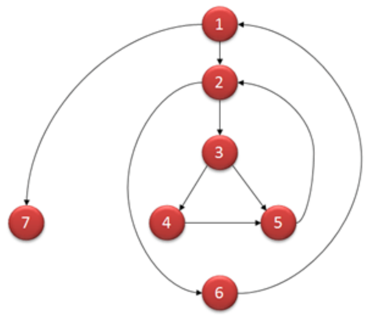
1. V (G) is the maximum number of independent paths in the graph
2. V (G) >=1
3. G will have one path if V (G) = 1
4. Minimize complexity to 10

Example -

```
i = 0;
n=4; //N-Number of nodes present in the graph
```

```
while (i<n-1) do
j = i + 1;
while (j<n) do
if A[i]<A[j] then
swap(A[i], A[j]);
end do;
i=i+1;
end do;
```

Flow graph for this program will be



Computing mathematically,

- V(G) = 9 - 7 + 2 = 4
- V(G) = 3 + 1 = 4 (Condition nodes are 1,2 and 3 nodes)
- Basis Set - A set of possible execution path of a program
- 1, 7
- 1, 2, 6, 1, 7
- 1, 2, 3, 4, 5, 2, 6, 1, 7
- 1, 2, 3, 5, 2, 6, 1, 7

Why Is It Important?

It can be used in two ways, to:

- Limit code complexity.
- Determine the number of test cases required.

Cyclomatic complexity can be one of the most difficult code metrics to understand. And that makes it difficult to calculate.

How this metric is useful for software testing?

Basis Path testing is one of White box technique and it guarantees to execute atleast one statement during testing. It checks each linearly independent path through the program, which **means number test cases, will be equivalent to the cyclomatic complexity of the program.**

This metric is useful because of properties of Cyclomatic complexity (M) -

1. M can be number of test cases to achieve branch coverage (Upper Bound)
2. M can be number of paths through the graphs. (Lower Bound)

Consider this example -

```
If (Condition 1)
Statement 1
Else
Statement 2
If (Condition 2)
Statement 3
Else
Statement 4
```

Cyclomatic Complexity for this program will be 9-7+2=4.

As complexity has calculated as 4, four test cases are necessary to the complete path coverage for the above example.

Steps to be followed:

The following steps should be followed for computing Cyclomatic complexity and test cases design.

Step 1 - Construction of graph with nodes and edges from the code
Step 2 - Identification of independent paths
Step 3 - Cyclomatic Complexity Calculation
Step 4 - Design of Test Cases
Once the basic set is formed, TEST CASES should be written to execute all the paths.
More on V (G):
Cyclomatic complexity can be calculated manually if the program is small. Automated tools need to be used if the program is very complex as this involves more flow graphs. Based on complexity number, team can conclude on the actions that need to be taken for measure.
Following table gives overview on the complexity number and corresponding meaning of v (G):

Complexity Number	**Meaning**
1-10	Structured and well written code High Testability Cost and Effort is less
10-20	Complex Code Medium Testability Cost and effort is Medium
20-40	Very complex Code Low Testability Cost and Effort are high
>40	Not at all testable Very high Cost and Effort

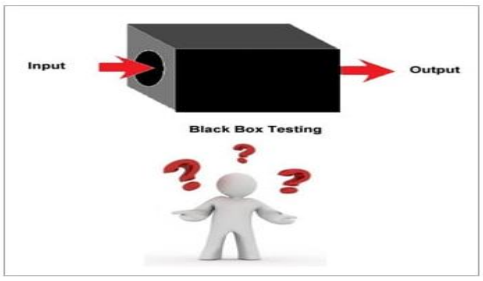
Tools for Cyclomatic Complexity calculation:
Many tools are available for determining the complexity of the application. Some complexity calculation tools are used for specific technologies. Complexity can be found by the number of decision points in a program. The decision points are if, for, for-each, while, do, catch, case statements in a source code.
Examples of tools are

- OCLint - Static code analyzer for C and Related Languages
- devMetrics - Analyzing metrics for C# projects
- Reflector Add In - Code metrics for .NET assemblies
- GMetrics - Find metrics in Java related applications
- NDepends - Metrics in Java applications

Uses of Cyclomatic Complexity:
Cyclomatic Complexity can prove to be very helpful in

- Helps developers and testers to determine independent path executions
- Developers can assure that all the paths have been tested at least once
- Helps us to focus more on the uncovered paths
- Improve code coverage in Software Engineering
- Evaluate the risk associated with the application or program
- Using these metrics early in the cycle reduces more risk of the program

Black box testing:



In Black Box Testing, the tester tests an application without knowledge of the internal workings of the application being tested. Data are entered into the application and the outcome is compared with the expected results; what the program does with the input data or how the program arrives at the output data is not a concern for the tester performing black box testing. All that is tested is the behaviour of the functions being tested.
This is why black box testing is also known as functional testing which tests the functionality of a program. Note we can also have non-functional black box testing, such as performance testing which is a type of black box testing but instead of verifying the behaviour of the system, it tests how long it takes for a function to respond to user's inputs and how long it takes to process the data and generate outputs.
Because black box testing is not concerned with the underlying code, then the techniques can be derived from the requirement documents or design specifications and hence testing can start as soon as the requirements are written.
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.

Types Of Black Box Testing

Practically, there are several types of black box testing that are possible but if we consider the major variant of it then below mentioned are the two fundamental ones.
1) Functional Testing
This type deals with the functional requirements or specifications of an application. Here, different actions or functions of the system are being tested by providing the input and comparing the actual output with the expected output.
For Example, when we test a Dropdown list, we click on it and verify that it expands and all the expected values are showing in the list.
Few major types of Functional Testing are:

- Smoke Testing
- Sanity Testing
- Integration Testing
- System Testing
- Regression Testing
- User Acceptance Testing

2) Non-Functional Testing
Apart from the functionalities of the requirements, there are several non-functional aspects as well that are required to be tested to improve the quality and performance of the application.
Few major types of Non-functional testing include:

- Usability Testing
- Load Testing
- Performance Testing
- Compatibility Testing
- Stress Testing
- Scalability Testing

Black Box Testing Techniques
In order to systematically test a set of functions, it is necessary to design test cases. Testers can create test cases from the requirement specification document using the following black box testing techniques.

- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table Testing
- State Transition Testing
- Error Guessing
- Graph-Based Testing Methods
- Comparison Testing

Advantages of Black Box Testing are:

- The test is unbiased because the designer and the tester are independent of each other
- The tester does not need knowledge of any specific programming languages
- The test is done from the point of view of the user, not the designer
- Test cases can be designed as soon as the specifications are complete

Disadvantages of Black Box Testing are:

- The test can be redundant if the software designer has already run a test case
- The test cases are difficult to design
- Testing every possible input stream is unrealistic because it would take an inordinate amount of time; therefore, many program paths will go untested

What is Requirements Based Testing?

The process of requirements based testing deals with validating whether the requirements are complete, consistent, unambiguous, complete and logically connected. With such requirements, we can proceed to develop test cases to ensure that the test cases fulfil all the requirements. Testing in this technique revolves around requirements. The strategy of Requirement based testing is to integrate testing throughout the life cycle of the software development process, to assure quality of the Requirement Specification. The aim is defect prevention than defect detection.

Taking Requirement Based testing into account, testing is divided into the following types of activity:

- **Define Test Completion Criteria:** Testing should be defined in quantifiable terms. The goal is considered to be achieved only when test coverage is 100%.
- **Design Test Cases:** Test cases must be in accordance with requirements specification.
- **Build Test Cases:** Join the logical parts together to form/build test cases.
- **Execute Test Cases :**
  Execute the test cases to evaluate the results.
- **Verify Test Results:** Check whether actual results deviate from the expected ones.
- **Verify Test Coverage:** Check for functional test coverage.
- **Manage Test Library: Test** manager is responsible for monitoring the test case executions, that is, the tests passed or failed, or to ascertain whether all tests have been successfully performed.

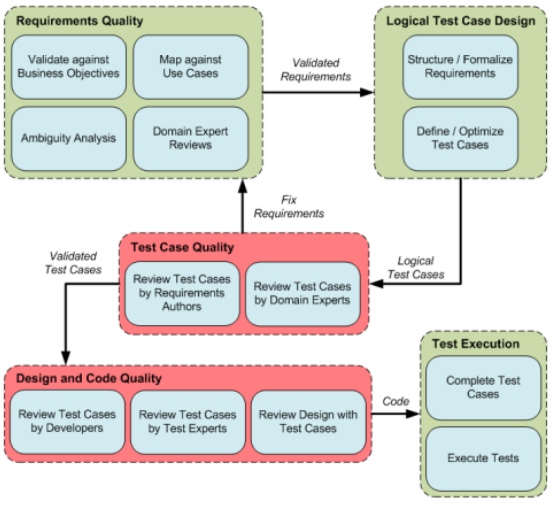
Why Requirements are Critical:

Various studies have shown that software projects fail due to the following reasons:

- Incomplete requirements and specifications
- Frequent changes in requirements and specifications
- When there is lack of user input to requirements

So the requirements based testing process addresses each of the above issues as follows:

- The Requirements based testing process starts at the very early phase of the software development, as correcting issues/errors is easier at this phase.
- It begins at the requirements phase as the chances of occurrence of bugs have its roots here.
- It aims at quality improvement of requirements. An insufficient requirement leads to failed projects.



Requirements-based testing process flow

What is Boundary Testing?

Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.

- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
- The basic idea in boundary value testing is to select input variable values at their:

1. Minimum
2. Just above the minimum
3. A nominal value
4. Just below the maximum
5. Maximum

- In Boundary Testing, Equivalence Class Partitioning plays a good role
- Boundary Testing comes after the Equivalence Class Partitioning.

What is Equivalent Class Partitioning?

Equivalent Class Partitioning is a black box technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system, etc. In this technique, you divide the set of test condition into a partition that can be considered the same.

- It divides the input data of software into different equivalence data classes.
- You can apply this technique, where there is a range in the input field.

Example 1: Equivalence and Boundary Value

- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.
- While value 11 to 99 are considered invalid for order and an error message will appear, **"Only 10 Pizza can be ordered"**

Order Pizza: Submit

Here is the test condition

1. Any Number greater than 10 entered in the Order Pizza field(let say 11) is considered invalid.
2. Any Number less than 1 that is 0 or below, then it is considered invalid.
3. Numbers 1 to 10 are considered valid
4. Any 3 Digit Number say -100 is invalid.

We cannot test all the possible values because if done, the number of test cases will be more than 100. To address this problem, we use equivalence partitioning hypothesis where we divide the possible values of tickets into groups or sets as shown below where the system behaviour can be considered the same.

The divided sets are called Equivalence Partitions or Equivalence Classes. Then we pick only one value from each partition for testing. The hypothesis behind this technique is **that if one condition/value in a partition passes all others will also pass**. Likewise**, if one condition in a partition fails, all other conditions in that partition will fail**.

Boundary Value Analysis- in Boundary Value Analysis, you test boundaries between equivalence partitions
In our earlier example instead of checking, one value for each partitions you will check the values at the partitions like 0, 1, 10, 11 and so on. As you may observe, you test values at **both valid and invalid boundaries**. Boundary Value Analysis is also called **range checking**.
Equivalence partitioning and boundary value analysis (BVA) are closely related and can be used together at all levels of testing.
Example 2: Equivalence and Boundary Value
Following password field accepts minimum 6 characters and maximum 10 characters
That means results for values in partitions 0-5, 6-10, 11-14 should be equivalent

Enter Password: Submit

Test Scenario #	**Test Scenario Description**	**Expected Outcome**
1	Enter 0 to 5 characters in password field	System should not accept
2	Enter 6 to 10 characters in password field	System should accept
3	Enter 11 to 14 character in password field	System should not accept

Examples 3: Input Box should accept the Number 1 to 10
Here we will see the Boundary Value Test Cases

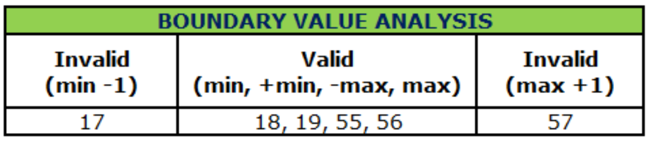
Test Scenario Description	**Expected Outcome**
Boundary Value = 0	System should NOT accept
Boundary Value = 1	System should accept
Boundary Value = 2	System should accept
Boundary Value = 9	System should accept
Boundary Value = 10	System should accept
Boundary Value = 11	System should NOT accept

Why Equivalence & Boundary Analysis Testing

1. This testing is used to reduce a very large number of test cases to manageable chunks.
2. Very clear guidelines on determining test cases without compromising on the effectiveness of testing.
3. Appropriate for calculation-intensive applications with a large number of variables/inputs

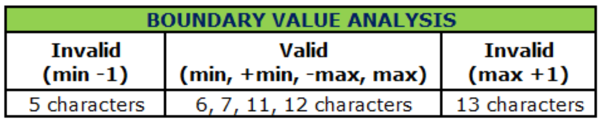
Boundary Value Analysis Test Case Design Technique
Boundary Value Analysis Test case design technique is one of the testing techniques. You could find other testing techniques such as Equivalence Partitioning, Decision Table and State Transition Techniques by clicking on appropriate links.
Boundary value analysis (BVA) is based on testing the boundary values of valid and invalid partitions. The Behaviour at the edge of each equivalence partition is more likely to be incorrect than the behaviour within the partition, so boundaries are an area where testing is likely to yield defects.
Check below video to see “Boundary Value Analysis In Software Testing”
Every partition has its maximum and minimum values and these maximum and minimum values are the boundary values of a partition.
A boundary value for a valid partition is a valid boundary value. Similarly a boundary value for an invalid partition is an invalid boundary value.

Tests can be designed to cover both valid and invalid boundary values. When designing test cases, a test for each boundary value is chosen.
For each boundary, we test +/-1 in the least significant digit of either side of the boundary.
Boundary value analysis can be applied at all test levels.
Example on Boundary Value Analysis Test Case Design Technique:
Assume, we have to test a field which accepts Age 18 – 56



BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	Valid (min, +min, -max, max)	Invalid (max +1)
17	18, 19, 55, 56	57

Minimum boundary value is 18
Maximum boundary value is 56
Valid Inputs: 18,19,55,56
Invalid Inputs: 17 and 57
Test case 1: Enter the value 17 (18-1) = Invalid
Test case 2: Enter the value 18 = Valid
Test case 3: Enter the value 19 (18+1) = Valid
Test case 4: Enter the value 55 (56-1) = Valid
Test case 5: Enter the value 56 = Valid
Test case 6: Enter the value 57 (56+1) =Invalid
Example 2:
Assume we have to test a text field (Name) which accepts the length between 6-12 characters.



BOUNDARY VALUE ANALYSIS		
Invalid (min -1)	Valid (min, +min, -max, max)	Invalid (max +1)
5 characters	6, 7, 11, 12 characters	13 characters

Minimum boundary value is 6
Maximum boundary value is 12
Valid text length is 6, 7, 11, 12
Invalid text length is 5, 13
Test case 1: Text length of 5 (min-1) = Invalid
Test case 2: Text length of exactly 6 (min) = Valid
Test case 3: Text length of 7 (min+1) = Valid
Test case 4: Text length of 11 (max-1) = Valid
Test case 5: Text length of exactly 12 (max) = Valid
Test case 6: Text length of 13 (max+1) = Invalid

Boundary Value Analysis Advantages:

- The BVA technique of testing is quite easy to use and remember because of the uniformity of identified tests and the automated nature of this technique.
- One can easily control the expenses made on the testing by controlling the number of identified test cases. This can be done with respect to the demand of the software that needs to be tested.
- BVA is the best approach in cases where the functionality of a software is based on numerous variables representing physical quantities.
- The technique is best at revealing any potential UI or user input troubles in the software.
- The procedure and guidelines are crystal clear and easy when it comes to determining the test cases through BVA.
- The test cases generated through BVA are very small.

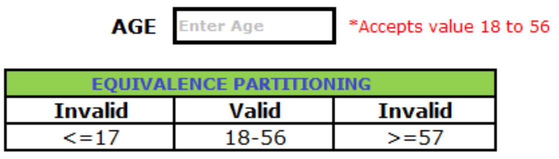
Boundary Value Analysis Disadvantages:

- This technique sometimes fails to test all the potential input values. And so, the results are unsure.

- The dependencies with BVA are not tested between two inputs.
- This technique doesn't fit well when it comes to Boolean Variables.
- It only works well with independent variables that depict quantity.

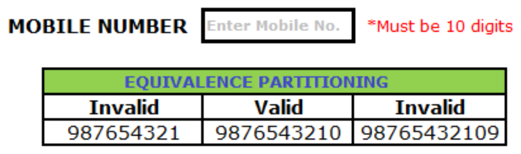
Equivalence Partitioning Test Case Design Technique

Equivalence Partitioning Test case design technique is one of the testing techniques. You could find other testing techniques such as Boundary Value Analysis, Decision Table and State Transition Techniques by clicking on appropriate links. Equivalence Partitioning is also known as Equivalence Class Partitioning. In equivalence partitioning, inputs to the software or system are divided into groups that are expected to exhibit similar behaviour, so they are likely to be proposed in the same way. Hence selecting one input from each group to design the test cases.
Each and every condition of particular partition (group) works as same as other. If a condition in a partition is valid, other conditions are valid too. If a condition in a partition is invalid, other conditions are invalid too.
It helps to reduce the total number of test cases from infinite to finite. The selected test cases from these groups ensure coverage of all possible scenarios.
Equivalence partitioning is applicable at all levels of testing.
Example on Equivalence Partitioning Test Case Design Technique:
Example 1:
Assume, we have to test a field which accepts Age 18 – 56



EQUIVALENCE PARTITIONING		
Invalid	Valid	Invalid
<=17	18-56	>=57

Valid Input: 18 – 56
Invalid Input: less than or equal to 17 (<=17), greater than or equal to 57 (>=57)
Valid Class: 18 – 56 = Pick any one input test data from 18 – 56
Invalid Class 1: <=17 = Pick any one input test data less than or equal to 17
Invalid Class 2: >=57 = Pick any one input test data greater than or equal to 57
We have one valid and two invalid conditions here.
Example 2:
Assume, we have to test a filed which accepts a Mobile Number of ten digits.



EQUIVALENCE PARTITIONING		
Invalid	Valid	Invalid
987654321	9876543210	98765432109

Valid input: 10 digits
Invalid Input: 9 digits, 11 digits
Valid Class: Enter 10 digit mobile number = 9876543210
Invalid Class Enter mobile number which has less than 10 digits = 987654321
Invalid Class Enter mobile number which has more than 11 digits = 98765432109

Comparison of Black Box Testing and White Box Testing

	Black Box Testing	**White Box Testing**
1	Black box testing is the Software testing method which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.
2	This type of testing is carried out by testers.	Generally, this type of testing is carried out by software developers.
3	Implementation Knowledge is not required to carry out	Implementation Knowledge is required to carry out

	Black Box Testing.	White Box Testing.
4	Programming Knowledge is not required to carry out Black Box Testing.	Programming Knowledge is required to carry out White Box Testing.
5	Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.
6	Black box testing means functional test or external testing.	White box testing means structural test or interior testing.
7	In Black Box testing is primarily concentrate on the functionality of the system under test.	In White Box testing is primarily concentrate on the testing of program code of the system under test like code structure, branches, conditions, loops etc.
8	The main aim of this testing to check on what functionality is performing by the system under test.	The main aim of White Box testing to check on how System is performing.
9	Black Box testing can be started based on Requirement Specifications documents.	White Box testing can be started based on Detail Design documents.
10	The Functional testing, Behaviour testing, Close box testing is carried out under Black Box testing, so there is no required of the programming knowledge.	The Structural testing, Logic testing, Path testing, Loop testing, Code coverage testing, Open box testing is carried out under White Box testing, so there is compulsory to know about programming knowledge.