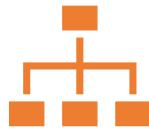


Basics of Software Testing and Testing Methods (14 Marks)



Unit Outcomes



Prepare test cases
and identify bug in
application or
program



Describe Entry and
exit criteria for the
test application



Validation of given
application using V
model and describes
feature of various
testing method

Definition Of Software Testing



Testing is the **execution** of programs **with an intention of finding defects**.

OR

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce the quality product.

Objective of testing

Finding Errors

Testing is process of executing a program with an intention of finding an error.

Creating good test cases

A good test case is one that has a high probability of finding yet undiscovered error.

Quality Improvement

Defects are fixed by developer so quality is improved.

Satisfying customer requirements:

Testing demonstrates customer that software works properly as per specification.

Why testing is important?

Example of application failure due to software bug:

- In 2015 fighter plane F-35 fell victim to a software bug, making it unable to detect targets correctly.
- China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocents live.
- In 1985, Canada's Therac-25 radiation therapy machine malfunctioned due to software bug and delivered lethal radiation doses to patients, leaving 3 people dead and critically injuring 3 others.

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
- **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.
- **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 bug occur?



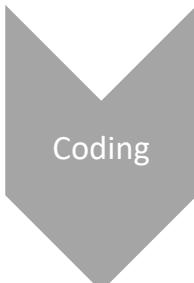
- Constantly changing requirements.
- Requirements not gathered properly



- Not written properly
- Constantly changing
- Not well communicated



- Not done properly
- Quickly done
- Not well communicated



- Inexperienced programmer/Programmers without proper domain knowledge
- Software Complexity
- Programmers are not following standards
- Schedule Pressure or plain dumb mistake

Entry criteria

Entry criteria are the condition or the set of conditions, which should exist or be met in order to start a process.

Let's see some of the conditions or situations which may be seen as an entry criteria for the initiation of testing activities.

- Requirements should be clearly defined and approved.
- **Test Design** and documentation plan is ready.

Entry criteria

- Availability of the **test environment** supporting necessary hardware, software, network configuration, settings and tools for the purpose of test execution.
- Testers are trained and necessary resources are available.
- Availability of proper and adequate **test data**(like **test cases**).
- It depends upon which software development model is used.

Exit criteria

Exit Criteria is often viewed as a single document concluding the end of a life cycle phase.

Let's see some of the conditions or situations which may be seen as an exit criteria for testing activities.

- Testing Deadline
- Completion of test case execution.

Exit criteria

- Completion of Functional and code coverage to a certain point.
- Bug rates fall below certain level and no high priority bugs are identified.
- Management decision.

Software Quality Concept

Quality : Quality means consistently meeting customer needs in terms of requirement, Cost and delivery schedule.

Quality of s/w is reasonably bug free, delivered on time and within budget, meets requirements and meeting customer exceptions.



Quality Assurance

- Quality assurance(QA) is set of activities for ensuring quality in the process by which products are developed.
- QA activities are work process oriented.
- They measure the process, identify deficiencies and suggest improvements.

Quality Control

- Quality control (QC) is a set of activities for ensuring quality in products. The activities focus on identifying defects in the actual product produced.
- QC activities are work product oriented.
- They measure the product, identify defect.

Difference between QA and QC

Quality Assurance	Quality Control
Process oriented activities.	Product oriented activities.
QA is the process of managing for quality.	QC is used to verify the quality of the output.
They measure the process, identify the deficiencies/weakness and suggest improvements.	They measure the product, identify the deficiencies/weakness and suggest improvements.
SQA is a set of activities for ensuring quality in software engineering processes (that ultimately result in quality in software products). The activities establish and evaluate the processes that produce products.	SQC is a set of activities for ensuring quality in software products. The activities focus on identifying defects in the actual products produced.
Activities of QA are Process Definition and Implementation, Audits and Training	Activities of QC are Reviews and Testing
It includes Prevention oriented activities.	It includes detection-oriented activities.
Verification is an example of QA	Validation/Software Testing is an example of QC
QA is a managerial tool	QC is a corrective tool

Verification & Validation

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

Verification

- Verification is a process of confirming whether the software meets its specifications.
- Verification is the process of examining a product to discover its defects.
- Verification is usually performed by static testing, or inspecting without execution on a computer.
- Verification is a “human” examination or review of the work product.

Validation

- Validation is a process of confirming whether the software meets user requirements.
- Validation is the process of executing a product to expose its defects.
- Validation is performed by dynamic testing or testing with execution on a computer.

Verification

1. **Verification** is a static practice of verifying documents, design, code and program.
2. It does not involve executing the code.
3. It is human based checking of documents and files.
4. **Verification** uses methods like inspections, reviews, walkthroughs, and Desk-checking etc.
5. **Verification** is to check whether the software conforms to specifications.
6. **Verification** is done by QA team to ensure that the software is as per the specifications in the SRS document.
7. It generally comes first-done before **validation**.

Validation

1. **Validation** is a dynamic mechanism of validating and testing the actual product.
2. It always involves executing the code.
3. It is computer-based execution of program.
4. **Validation** uses methods like black box (functional) testing, gray box testing, and white box (structural) testing etc.
5. **Validation** is to check whether software meets the customer expectations and requirements.
6. **Validation** is carried out with the involvement of testing team.
7. It generally follows after **verification**.

Who is a Software Tester??..

- Software Tester is the one who performs testing and find bugs, if they exist in the tested application.

Who Should Test?

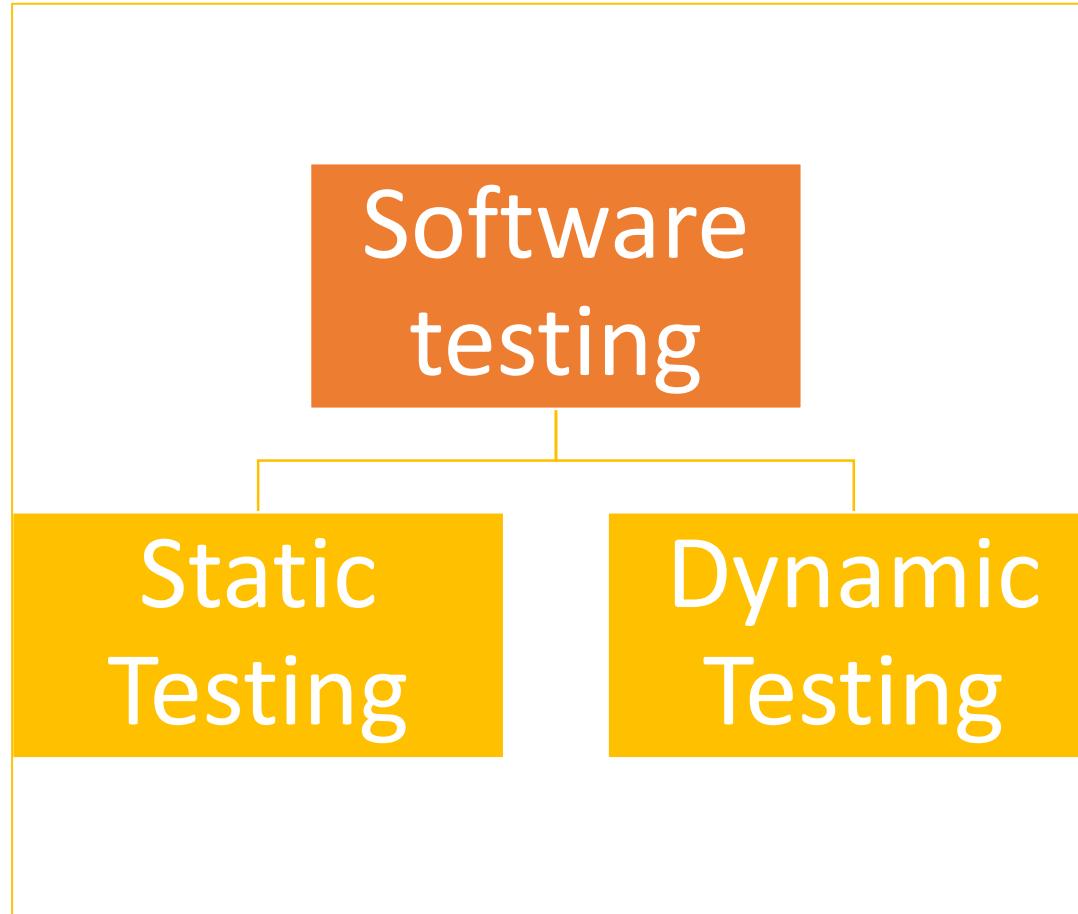


- **Developer**
 - Understands the system
 - But, will test gently
 - And, is driven by deadlines

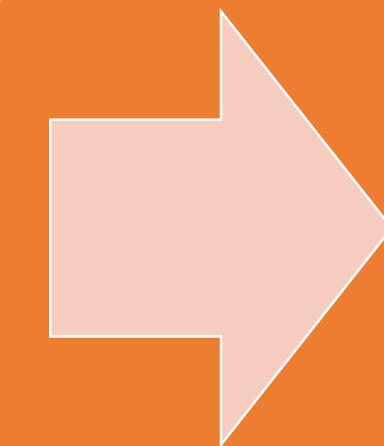


- **Independent tester**
 - Must learn system
 - But, will attempt to break it
 - And, is driven by “quality”

Methods of testing



Static Testing



Non execution based
Static testing is non-execution based technique.



Conducts review
Review requirements,coding,design documents.

Dynamic Testing



Execution based

Dynamic testing is execution based technique.



Entire system will undergo testing process to check the actual and expected result are meeting or not

Difference between Static Testing and Dynamic testing

Static Testing	Dynamic Testing
Testing is done without executing the program	Testing is done by executing the program
This testing does verification process	Dynamic testing does validation process
Static testing gives assessment of code and documentation	Dynamic testing finds bugs in the software application
Static testing involves checklist and process to be followed.	Dynamic testing involves test cases for execution.
It is performed in the early stage of the software development.	It is performed at the later stage of the software development.
Cost of fixing bug is low	Cost of fixing is high
Example : Inspection, walkthrough	Example : Boundary value analysis ,Equivalence Partitioning

What is test case

Test case is a well-documented procedure designed to test the functionality of the system.

The primary purpose of designing a test case is to find errors in the system

For designing the test case, it needs to provide a set of inputs and its corresponding expected outputs.

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.

Test case template

[Template of Test Case](#)

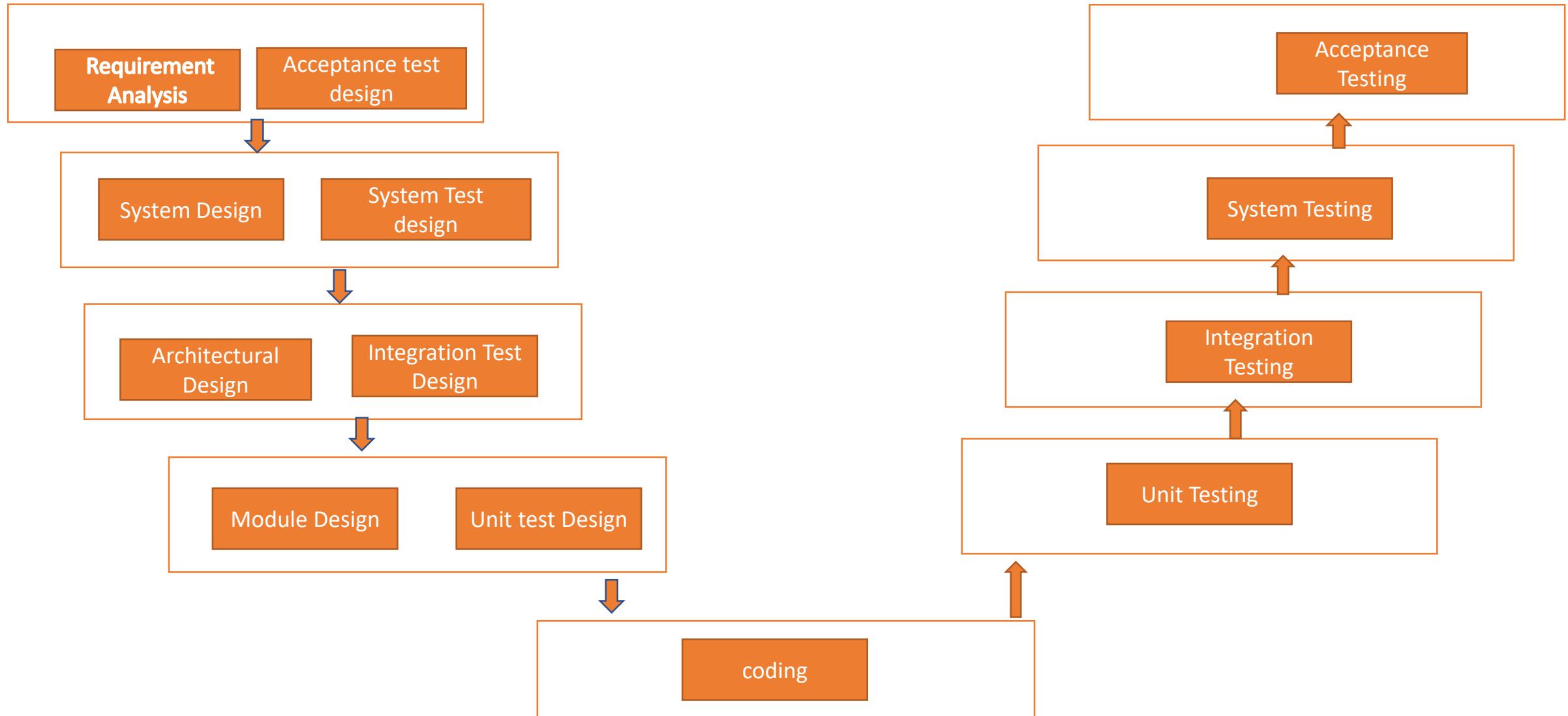
Example of Test case

[Test case for login screen](#)

V Model

- Extension of waterfall model
- Demonstrates the relationship between each phase of Development Life cycle and corresponding phase of Testing.
- Testing of the software is planned in Parallel with the corresponding phase of the development.

V Model



Verification Phase

Requirement Analysis

- Communicate with customer.
- Management tries to understand requirement from customer point of view.

Testing Activity

- Acceptance test designing planning is done.

Verification Phase

System Design

- Complete study of Requirement
- Detailing about user, hardware, software, databases and communication setup for the product.

Testing Activity

- System test designing is done

Verification Phase

Architectural Design

- Architect analyze system design document .
- Architectural specification are understood and designed .
- High level decision are taken place like programming language, memory ,communication protocols, controller and many more..
- System design broken into modules(High Level Design)

Testing Activity

- Integration test designing is done.

Verification Phase

Module Design

- Developer refers blueprint of software.
- Detail Internal design for all module is done(Low level design).

Testing Activity

Unit test designing is done.

Coding

- The best suitable programming language is decided based on the system and architectural requirements.
- System modules designed in the design phase is taken up in the Coding phase.
- The coding is performed based on the coding guidelines and standards.

Validation Phase

Unit Testing

- Unit tests designed in the module design phase are executed on the code during this validation phase.
- Unit testing is the testing at code level and helps eliminate bugs at an early stage.

Validation Phase

Integration Testing

- Integration tests are performed to test the coexistence and communication of the internal modules within the system.

Validation Phase

System Testing

- System tests check the entire system functionality and the communication of the system under development with external systems.
- Most of the software and hardware compatibility issues can be uncovered during system test execution.

Validation Phase

Acceptance Testing

- Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment.
- Acceptance tests uncover the compatibility issues with the other systems available in the user environment.
- It also discovers the nonfunctional issues such as load and performance defects in the actual user environment.

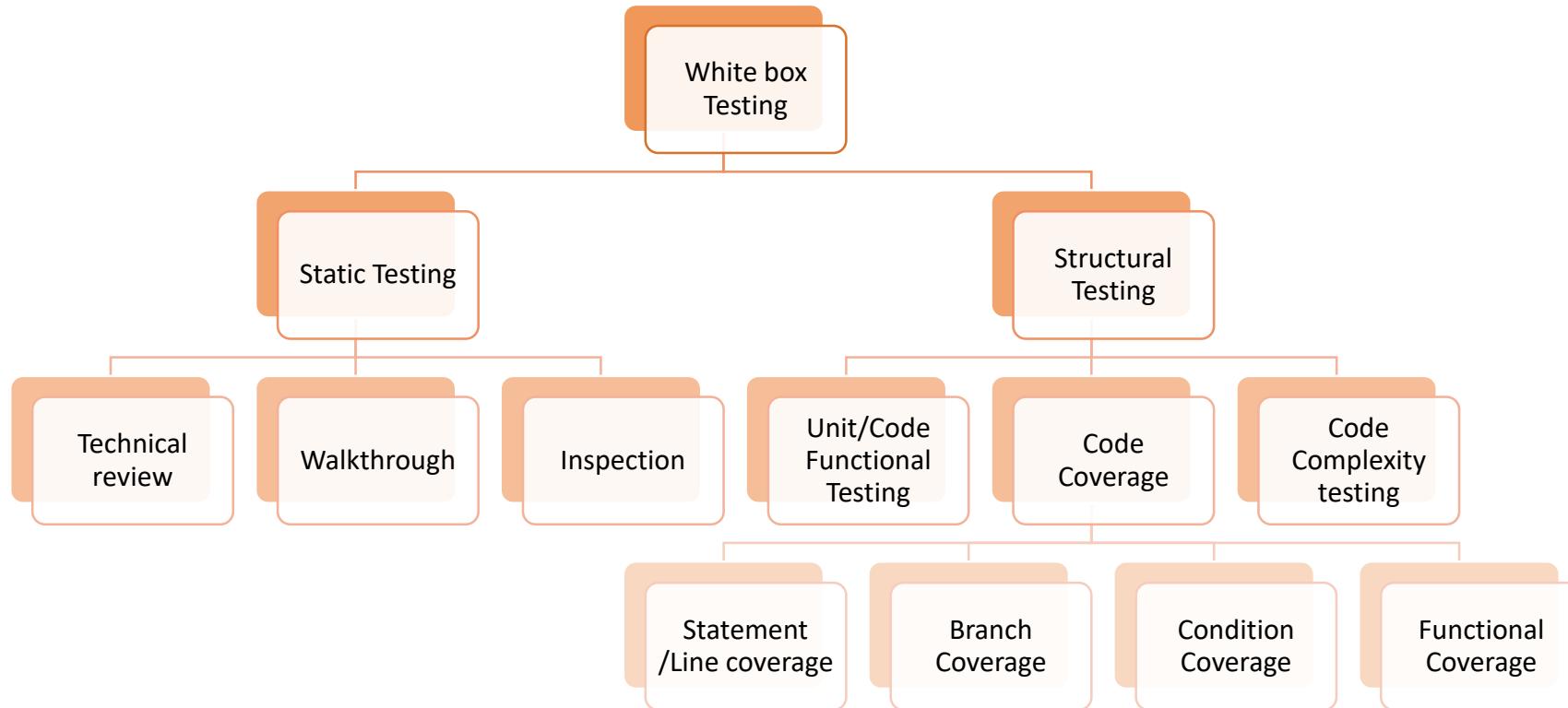
Advantages

- Simple and easy to understand and use.
- This is a highly disciplined model and Phases are completed one at a time.
- Testing activities like planning, test designing happens well before coding.
- This saves a lot of time.
- Works well for small projects where requirements are easily understood.

Disadvantages

- 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.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

Classification of Whitebox Testing



- Source code is tested without running it.
- Only need to examine and review the code/design document.

Static Testing

Technical Review (Peer Review)

Aim :To review code and looks for problems and oversights.

- Least formal method
- Presenter :Code is presented by author .
- Participants: Author, one or two programmers(peers) or testers .

Static Testing

Technical Review (Peer Review)

- It can happen at any time during development/testing life cycle.
- Sometimes, peer review is done jointly by the author and the peer.
- The peer may complete the review of the artifact on his own and give feedback to the author.
- There are no (or less) egos or pride attached between the peers, and defects are accepted easily.
- Peer-to-peer relationships are important in such reviews.

Static Testing

Technical Review (Peer Review)

Advantages:

- Fewer scheduling
- Mostly, defects are discussed and decision is reached very informally .This helps in finding and fixing the defects fast.

Static Testing

Technical Review (Peer Review)

Disadvantages:

- The person doing the review may or may not be an expert on the artifacts under review, and his/her suggestions may not be valid always.
- Peers may fix the defects without recording them, as they may not like to show defects of their partners/friends.
- Root causes of the defects may not be analyzed as defect fixes may not be known.
- Sometimes, the correct implementation may get replaced by the wrong one as reviewer may update the artifact without recording a defect.
- Approach-related defects may not be fixed, if both the author and the reviewer are at the same status and understand things in a similar way.

Static Testing

Walkthrough

Aim :To review code or documents and to find defects in order to improve quality of the product.

- More formal method than peer review.
- Presenter :Presented by author .
- Review artifacts: Requirement specification, architectural documents, diagrams or piece of code.
- Participants: Author, programmers or testers, senior programmer etc. .

Static Testing

Walkthrough steps which need to be carried out:

1. Reviewer receives copy of documents in advance .
2. Pre –meeting preparation is necessary.
3. Author guide the participants through the document according to his or her thought process to achieve a common understanding and to gather feedback.
4. Author presents code/documents.
5. Reviewers listens and questions if any thing looks suspicious.
6. After the review presenter write a report.

Static Testing

Walkthrough

Advantages:

- Saves time and money as defects are found and rectified very early in the life cycle.
- Provides value added comments from reviewers with different technical background and experience.
- It gives progress of the development to project management team.
- Problems can get recorded and suggestions can be received from the team for improving the work product further.

Static Testing

Walkthrough

Disadvantages:

- Availability of people can be an issue when teams are large. Multi-locations and distributed teams are major challenges in walkthrough.
- Team members may not be experts in giving comments and may need some training and basic knowledge about the project, artifact under walkthrough and so on.
- Time can be a constraint where schedules are very tight.

Static Testing

Inspection

Aim :To review code or documents to find defects and for doing process improvement.

- Formal method .
- Presenter :Other than author .
- Review artifacts: Requirement specification, architectural documents, diagrams and piece of code.
- Participants: Author, programmers or testers, senior programmer ,recorder (scribe)etc .

Static Testing

Inspection steps which need to be carried out:

1. Inspection is led by trained moderator .
2. Moderators role is to do peer examination of document.
3. Inspection is driven by checklist and rules.
4. Some inspectors are also assigned task such as moderator and recorder to assure that rules are followed and review runs effectively.
5. Inspectors receives copy of documents in advance .

Static Testing

Inspection steps which need to be carried out:

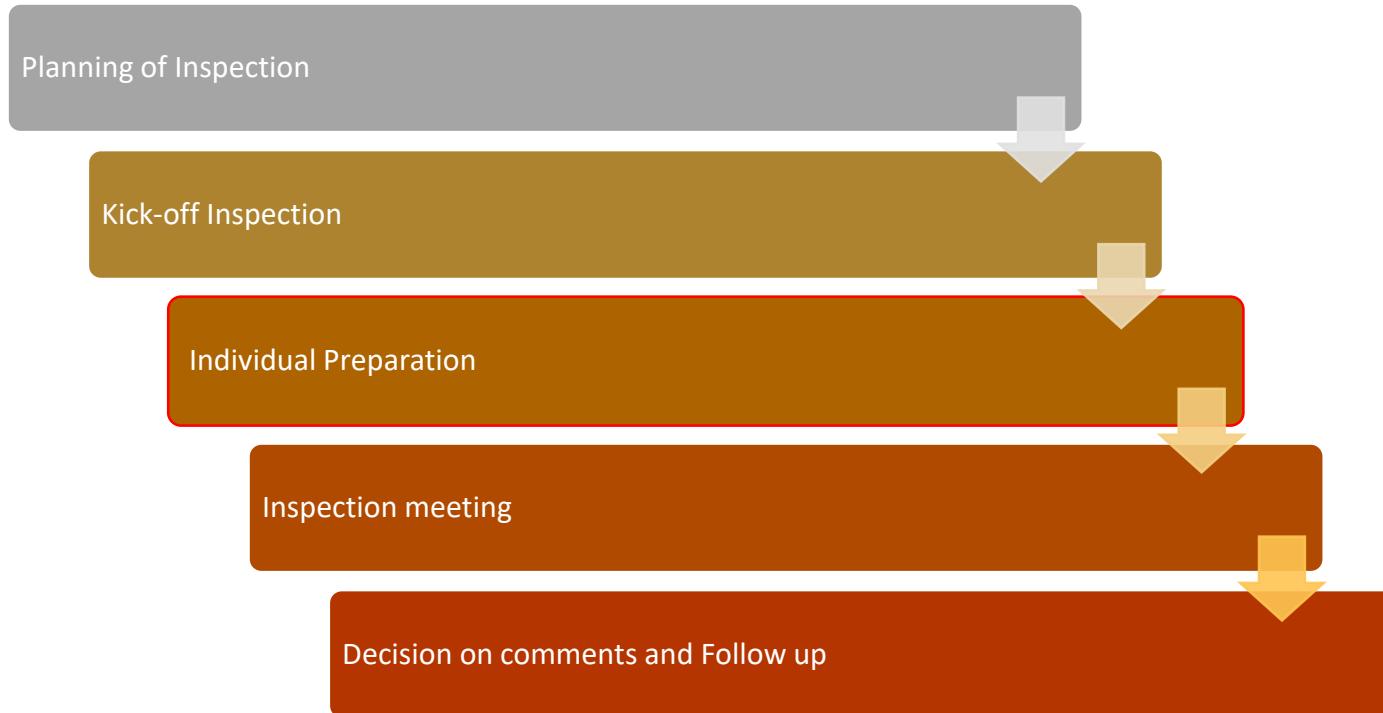
6. Pre –meeting preparation is necessary.
7. Presenter presents code/documents.
8. Reviewers listens and questions if any thing looks suspicious.
9. Recorder makes notes of comments given by the inspectors.
10. After Inspection meeting held, Inspector might meet again to discuss the defect they found .

Static Testing

Inspection steps which need to be carried out:

11. Inspectors work with moderator to prepare written report that identifies work necessary to address problems.
12. Report is shared with stakeholders.
13. Programmer make changes in the code.
14. Post inspection formal follow-up process is used to ensure timely and prompt corrective action.
15. According to need re-inspection can be scheduled.

Inspection Phases



Inspection Phases

Planning for Inspection:

- Planning for inspection involves selecting people for inspection.
- Allocating roles to other people
- Defining the entry and exit criteria for inspection
- Selecting which parts of artifacts are to be looked at.

Inspection Phases

Kick-off Inspection:

- Kick-off inspection may start by distributing artifacts
- Explaining the objectives of inspection, process to be followed for inspection.
- Checking entry criteria for the artifact as well as inspection process

Inspection Phases

Individual Preparation:

- It is expected that participants must come prepared for the inspection.
- Work must be reviewed and comments (including potential defects and questions) by each of the participants must be ready before the inspection meeting.

Inspection Phases

Inspection Meeting:

- Proceeds of the meeting such as discussions done or logging of comments.
- Preparation of minutes of meeting must be completed in defined time frame.
- The participants of the meeting may simply note defects, make recommendations for handling the defects or make decisions about the defects.
- Recorder shall note these comments.

Inspection Phases

Decision on Comments:

- An organization has to decide the fate of comments once the inspection is over.
- It is not necessary that all comments will be accepted.
- Comments may be rejected, deferred or may undergo another iteration of inspection.
- For accepted comments, the fixing of defects is done by the author.

Inspection Phases

Follow Up:

- Checking that the defects have been addressed
- Whether exit criteria has been met or not is required.
- The findings can be used to gather statistics about work product, project or process.

Static Testing

Inspection

Advantages:

- It helps the author to improve the quality of the document under inspection .
- It removes defects efficiently and as early as possible.
- It improves product quality .
- Expert's opinion is available as they are present during review. Subject-matter experts can clarify many issues.

Static Testing

Inspection

Disadvantages:

- Experts may not be ready with review comments before inspection and time may not be utilized properly
- Facilitator's job becomes critical in case there is a difference of opinion between two inspectors, as the final aim of the organization/project is to get inputs from inspectors.

Structural Testing

Functional Coverage

- This is to identify **how many program functions** are **covered** by test cases.
- The **requirements** of a product are **mapped into functions** during the design phase and each of the functions form a logical unit.
- Providing function coverage, **test cases can be written** so as to exercise each of the **different functions in the code**.

Structural Testing

Functional Coverage

- We can also measure how many times a given function is called.
- This will indicate which functions are used most often and hence these functions - become the target of any performance testing and optimization.
- Function coverage can help in improving the performance as well as quality of the product.

Function Coverage= $\frac{\text{Total number of functions exercised}}{\text{Total number of functions in a program}} * 100$

Structural Testing

Functional Coverage

Advantages:

- Functions are easier to identify in a program and hence it is easier to write test cases to achieve 100 percent function coverage.
- Functions have a more logical mapping to requirements and hence can provide a more direct correlation to the test coverage of the product.
- The importance of functions can be prioritized based on the importance of the requirements they realize.

It would be easier to prioritize the functions for testing.

Structural Testing

Code Complexity Testing

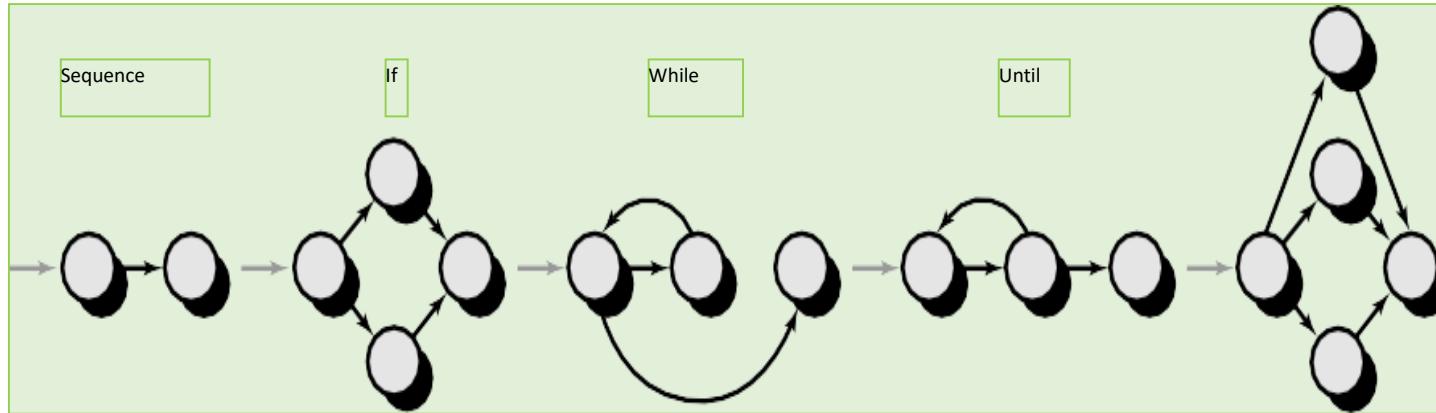
- Cyclomatic complexity is a software metric used to measure the complexity of a program.
- These metric, measures independent paths through program source code.
- Cyclomatic complexity can be calculated with respect to functions, modules, methods or classes within a program.
- This metric was developed by Thomas J. McCabe in 1976 and it is based on a control flow representation of the program.
- Control flow depicts a program as a graph which consists of Nodes and Edges.
- A program is represented in the form of a flow graph.

Code Complexity Testing → 1. Flow Graph Notation

- **flow graph node**, represents one or more procedural statements.
(A sequence of process boxes and a decision diamond can map into a single node)
- The arrows on the flow graph, called **edges** or links, represent **flow of control** and are analogous to flowchart arrows.
- Areas bounded by edges and nodes are called **regions**.
- Each node that contains a condition is called a **predicate node** and is characterized by two or more edges emanating from it.

Structural Testing

Code Complexity Testing → Flow Graph Notation



Structural Testing

Code Complexity Testing → 2.Compound Logic

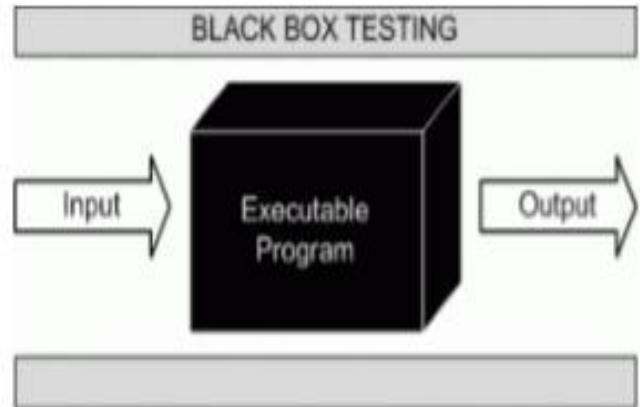
Compound Logic

- When compound conditions are encountered in a procedural design, the generation of a flow graph becomes slightly more complicated .
- A compound condition occurs when one or more Boolean operators (logical OR, AND, NAND, NOR) is present in a conditional statement.

Structural Testing

Code Complexity Testing → [Independent Paths and Cyclomatic Complexity](#)

Blackbox Testing



Blackbox Testing

- Black Box Testing, also known as **Behavioural Testing/closed box testing**.
- It is a software testing method in which the internal structure/design/implementation of the item being tested is not known to the tester.
- The main purpose of the Black Box is to check whether the software is working as per expected in requirement document & whether it is meeting the user expectations or not.

Blackbox Testing

Steps followed to carry out any type of Black Box Testing.

- Initially **requirements and specifications** of the system are **examined**.
- Tester chooses **valid inputs** (positive test scenario) to check whether **SUT** (software under test) **processes them correctly**.
- Also some **invalid inputs** (negative test scenario) are chosen **to verify** that the **SUT** is able to detect them.
- Tester **determines expected outputs** for all those inputs.

Blackbox Testing

Steps followed to carry out any type of Black Box Testing.

- Software tester constructs test cases with the selected inputs.
- The test cases are executed.
- Software tester compares the actual outputs with the expected outputs.
- Defects if any are fixed and re-tested.

Techniques for Blackbox Testing

- Requirement based testing
- Boundary value analysis
- Equivalence Partitioning

Blackbox Testing

Requirement Based Testing:

- Requirements-based testing is a testing approach in which test cases, conditions and data are derived from requirements.
- In this testing both implicit and explicit requirements are required.
- Explicit requirements are stated and documented as a part of the requirements specification.
- Implicit requirements are those are not documented but assumed to be incorporated in the system.
- In some organizations all explicit requirements and implicit requirements are collected and documented as "Test Requirements Specification (TRS)".

Requirement Based Testing:

Project 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.

Requirement Based Testing:

Project Stages in Requirements based Testing:

- **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.

Blackbox Testing

Boundary value Analysis:

- Most of the defects in software products hover around conditions and boundaries.
- 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**.

Blackbox Testing

Boundary value Analysis:

The basic idea in boundary value testing is to select input variable values at their:

1. Minimum
2. Just below the minimum
3. Just above the minimum
4. Just below the maximum
5. Maximum
6. Just above the maximum

Blackbox Testing

Boundary value Analysis:

Example :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

Blackbox Testing

Boundary value Analysis:

Advantages of Boundary Value Analysis:

- Very good at exposing potential user interface/user input problems.
- Very clear guide lines on determining test cases.
- Very small set of test cases generated.

Disadvantages of Boundary Value Analysis:

- Does not test all possible inputs.
- Does not test dependencies between combinations of inputs.

Blackbox Testing

Equivalence Partitioning:

- Equivalence partitioning (EP) is a specification-based or black-box technique.
- It can be applied at any level of testing and is often a good technique to use first.
- 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 an equivalence class or a partition.

Blackbox Testing

Equivalence Partitioning:

- In equivalence-partitioning technique we need to **test** only one condition from each partition.
- This is because we are assuming that all the conditions in one partition will be treated in the same way by the software.
- This is a software testing technique which divides the input date into many partitions. Values from each partition must be tested at least once.
- Partitions with valid values are used for Positive Testing. While, partitions with invalid values are used for negative testing.

Blackbox Testing

Steps to prepare Equivalence Partitioning table as follows:

1. Choose criteria for doing the equivalence partitioning (range, list of values and so on.)
2. Identify the valid equivalence classes based on the above criteria (number of ranges allowed values, and so on.)
3. Select a sample data from that partition.

Blackbox Testing

Steps to prepare Equivalence Partitioning table as follows:

4. Write the expected result based on the requirements given.
5. Identify special values, if any and include them in the table.
6. Check to have expected results for all the cases
7. If the expected result is not clear for any particular test case, mark appropriately and escalate for corrective actions.

Blackbox Testing

Equivalence Partitioning:

For example let us consider the bank given the interest rate for loan based on the age group as follows:

Age group Interest Rate Under 35 is 8%

Age group Interest Rate 35 to 59 is 9%

Age group Interest Rate 60 and Above 60 is 11%

Blackbox Testing

Equivalence Partitioning:

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 60 years of age (**Valid input**)
4. Negative Age (**Invalid input**)
5. Age is zero (**Invalid input**)
6. Age as any three digit number (**Valid input**)

Blackbox Testing

Equivalence Partitioning:

Sr. No	Equivalence partitions	Type of input	Test data	Expected output
1	Age below 35	Valid	20 ,30	Interest Rate= 8%
2	Age 35 to 59	Valid	40, 50	Interest Rate= 9%
3	Age above 60	Valid	65, 72	Interest Rate= 11%
4	Negative Age	Invalid	-10	Warning message Invalid input
5	Age is zero	Invalid	0	Warning message- Invalid input

Blackbox Testing

Equivalence Partitioning:

Advantages of Equivalence Partitions:

- Equivalence partitions are designed so that every possible input belongs to one and only one equivalence partition.

Disadvantages of Equivalence Partitions:

- Doesn't test every input
- No guide lines for choosing inputs.

Thank You!!!

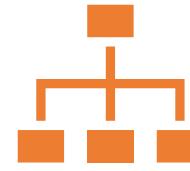
- Any Query?



Types and level of testing (18 Marks)



Unit Outcomes



Apply specified testing level
for the given web
application

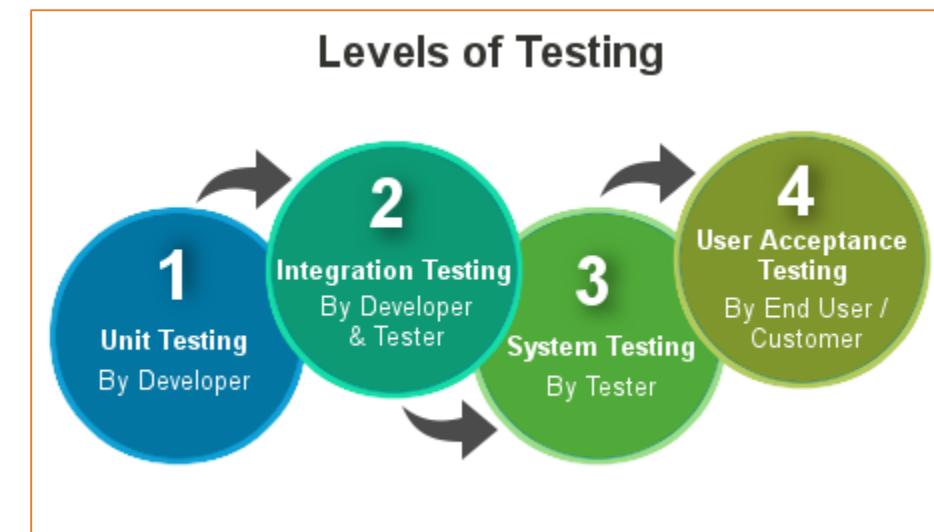
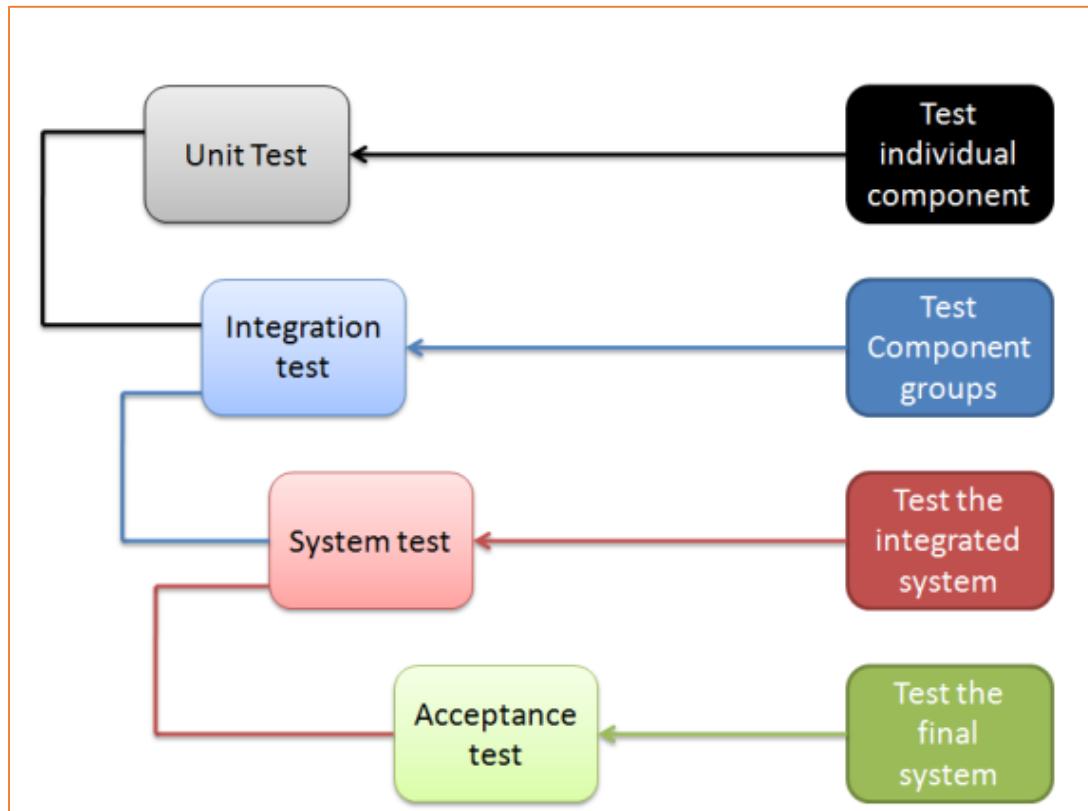


Apply acceptance testing for
given web based
application.



Apply performance testing
for the Specified application

Levels of Testing



Unit Testing

Unit Testing is a level of software testing where **individual units/ components** of a software are tested.

Purpose is to validate that each unit of the software performs as designed.

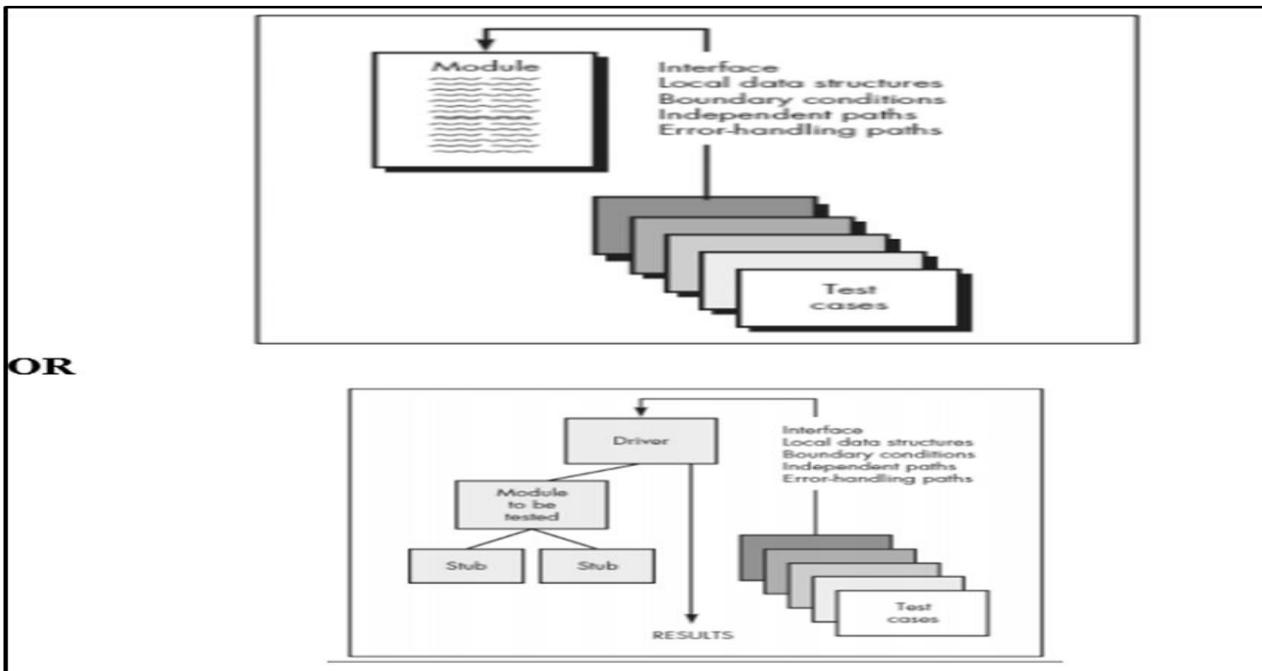
When it is performed?

Unit Testing is the first level of testing and is performed prior to Integration Testing.

Unit Testing

- A unit is the smallest testable part of software.
- It is executed by the Developer.
- Unit Testing is performed by using the White Box Testing method
- Example: - A function, method, Loop or statement in program is working fine.

Unit Testing



Unit Testing Environment

Unit Testing

Following test cases are performed for unit testing:

- **The module interface** is tested to ensure that information properly flows into and out of the program unit under test.
- **Local data structures** are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
- **All independent paths** through the control structure are exercised to ensure that all statements in a module have been executed at least once.

Unit Testing

Following test cases are performed for unit testing:

- **Boundary conditions** are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
- **Error-handling paths** Ensure algorithm responds correctly to specific error conditions .

Advantages of Unit testing

- Unit testing increases confidence in changing/ maintaining code.
- If good unit tests are written then we will be able to promptly catch any defects introduced due to the change.
- The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.
- Debugging is easy.

Drivers

- It can simulate the behavior of upper-level module that is not integrated yet.
- Drivers modules act as the temporary replacement of module and act as the actual products.
- A simple main program that accepts test case data, passes such data to the component being tested, and prints the returned results

- Serve to replace modules that are subordinate to (called by) the component to be tested
- It can simulate the behavior of lower-level module that are not integrated.
- They are act as a temporary replacement of module and provide same output as actual product.

Integration Testing

Integration Testing is a level of software testing where individual units are combined and tested as a group.

When it is performed?

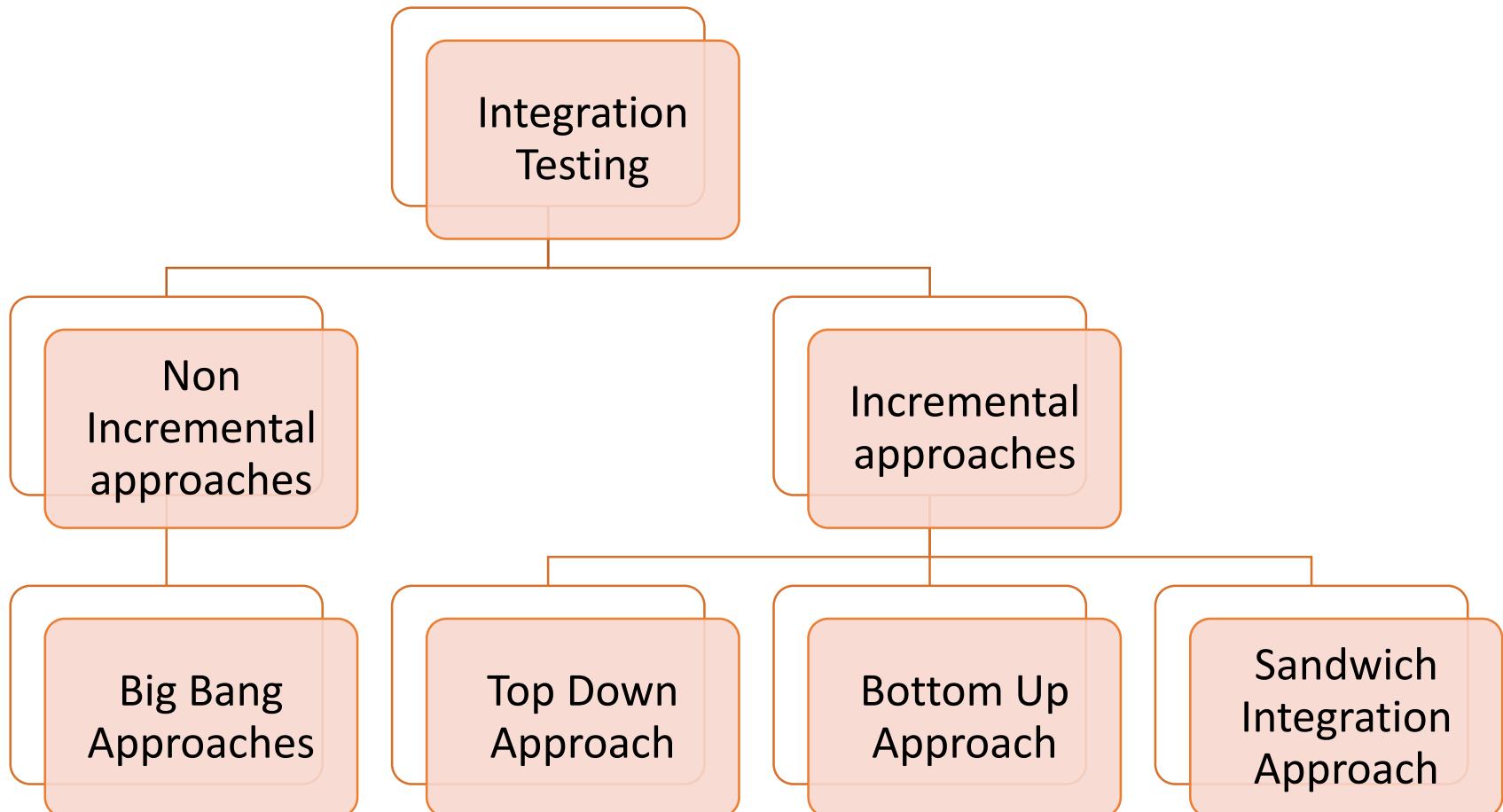
After Unit Testing integration testing is done.

Integration Testing

What it tests?

- Tests integration or interfaces between components.
- interactions to different parts of the system such as an operating system, file system and hardware or interfaces between systems.

Integration Testing Approaches



Integration Testing

Non incremental approach → Big Bang Approach

- Commonly called the “Big Bang” approach.
- All components are combined in advance.
- The entire program is tested as a whole.

When it is performed?

After all unit are ready big bang integration is performed.

Integration Testing

Advantages of Big Bang Approach

- Convenient for small systems.

Integration Testing

Disadvantages of Big Bang Approach

- Since all modules are tested at once, high risk critical modules are not isolated and tested on priority.
- Correction is difficult because isolation of causes is complicated.
- Once a set of errors are corrected, more errors occur, and testing appears to enter an endless loop
- Integration testing can commence only after "all" the modules are designed, testing team will have less time for execution in the testing phase.

Integration Testing

Incremental approach

- Three kinds
 - Top-down integration
 - Bottom-up integration
 - Sandwich integration
- The program is constructed and tested in small increments.
- Errors are easier to isolate and correct.
- Interfaces are more likely to be tested completely.
- A systematic test approach is applied.

Integration Testing

Incremental approach → Top-down integration

- Modules are integrated by moving downward through the control hierarchy, beginning with the main module.
- It takes help of dummy program called stub for testing.

- Subordinate modules are incorporated in either a depth-first or breadth-first fashion.

Integration Testing

Incremental approach → Top-down integration

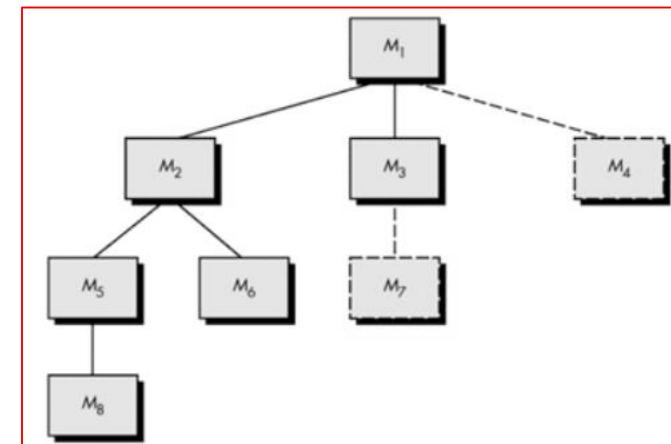
Integration can be done in two ways:

- Depth First Method: All modules on a major control path are integrated.
- Breadth First method: All modules directly subordinate at each level are integrated.

Integration Testing

Incremental approach → Top-down integration

Top
Down



Integration Testing

Incremental approach → Top-down integration procedure

1. Main control module used as a test driver and stubs are substitutes for components directly subordinate to it.
2. Subordinate stubs are replaced one at a time with real components. (following the depth-first or breadth-first approach).
3. Tests are conducted as each component is integrated.

Integration Testing

Incremental approach → Top-down integration procedure

4. On completion of each set of tests and other stub is replaced with a real component.
5. Regression testing may be used to ensure that new errors not introduced.

Integration Testing

Advantages of Top-down integration

- Fault Localization is easier.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could be found and fixed first.

Integration Testing

Disadvantages of Top-down integration

- Stubs need to be created to substitute for modules that have not been built or tested yet; this code is later discarded.

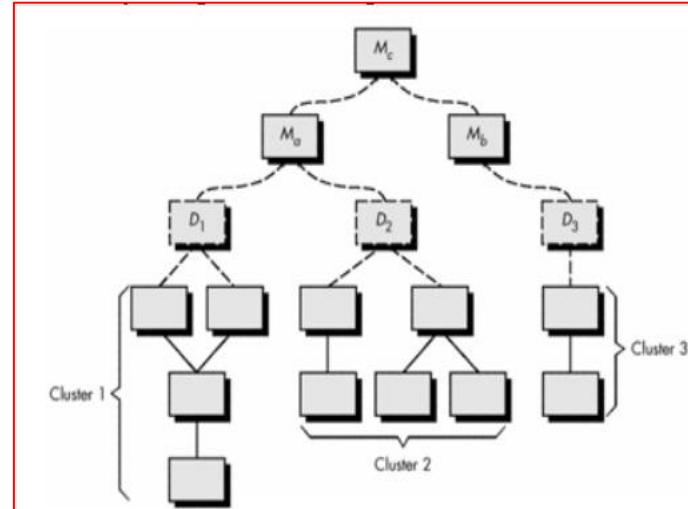
Integration Testing

Incremental approach → Bottom Up integration

- Modules are integrated by moving upward through the control hierarchy, beginning with the lower level module.
- It takes help of dummy program called driver for testing.

Integration Testing

Incremental approach → Bottom up integration



Integration Testing

Incremental approach → Bottom up integration

- Low level components are combined in clusters that perform a specific software function.
- A driver (control program) is written to coordinate test case input and output.

Integration Testing

Incremental approach → Bottom up integration procedure

1. Drivers are written .
2. The cluster is tested.
3. Drivers are removed.
4. Clusters are combined moving upward in the program structure.

Integration Testing

Advantages of Bottom Up integration

- This approach verifies low-level data processing early in the testing process.
- Need for stubs is eliminated

Integration Testing

Disadvantages of Bottom Up integration

- Driver modules need to be built to test the lower-level modules.
- This code written for drivers is later discarded or expanded into a full-featured version.

Integration Testing

Incremental approach → Sandwich integration

- Consists of a combination of both top-down and bottom-up integration.
- Occurs both at the highest level modules and also at the lowest level modules.
- Proceeds using functional groups of modules, with each group completed before the next.

Integration Testing

Incremental approach → Sandwich integration

- High and low-level modules are grouped based on the control and data processing they provide for a specific program feature.
- Integration within the group progresses in alternating steps between the high and low level modules of the group.
- When integration for a certain functional group is complete, integration and testing moves onto the next group.

Integration Testing

Incremental approach → Sandwich integration

Advantages :

- Repeats the advantages of both types of integration.

Disadvantages:

- Requires a disciplined approach so that integration doesn't tend towards the “big bang” scenario.

Regression Testing

- Each new addition or change to baselined software may cause problems with functions that previously worked flawlessly.
- Regression testing re-executes a small subset of tests that have already been conducted
 - Ensures that changes have not propagated unintended side effects
 - Helps to ensure that changes do not introduce unintended behavior or additional errors
 - May be done manually or through the use of automated capture/playback tools

Regression Testing

Regression test suite contains three different classes of test cases

- A representative sample of tests that will exercise all software functions.
- Additional tests that focus on software functions that are likely to be affected by the change.
- Tests that focus on the actual software components that have been changed.

Client Server Testing

- Client server testing gives an opportunity for number of users to work with software at time.
- In simple terms request are coming from number of clients for doing some actions and server is serving these request.

Client Server Testing

Testing approaches of client server system → Component Testing

- For testing Client and server individually approach and test plan need to be defined.
- One may have to devise simulators to replace corresponding components while testing the component targeted by the test.
- When server is tested, we may need a client simulator, while testing of client may need server simulator.

Client Server Testing

Testing approaches of client server system → Integration Testing

- After successful testing of servers, clients and network, they are brought together to form the system and system test cases are executed.
- Communication between client and server is tested in integration testing.

Client Server Testing

Testing approaches of client server system → Performance Testing

- System performance is tested when number of clients are communicating with server at a time.
- we can test the system under maximum load as well as normal load expected.

Client Server Testing

Testing approaches of client server system → Concurrency Testing

- It may be possible that multiple users may be accessing same record at a time.
- Concurrency testing is required to understand the behavior of a system under such circumstances.

Client Server Testing

Testing approaches of client server system -→Disaster Recovery Testing

- When the client and server are communicating with each other, there exists a possibility of breaking of the communication due to various reasons or failure of either client or server or link connecting them.
- It may involve testing the scenario of such failure at different points in the system and action taken by the system in each case.

Client Server Testing

Testing approaches of client server system -→Testing for extended periods

- In client server application it may be expected that server is running 24*7 for extended period.
- one need to conduct testing over an extended period to understand if service level of network and server deteriorates over time due to some reasons.

Client Server Testing

Testing approaches of client server system → Compatibility Testing

- Servers may be in different hardware, software or operating environment than the recommended one.
- Client may differ significantly from the expected environmental variables.
- Testing must ensure that performance must be maintained on the range of hardware and software configurations.

- Web application is further improvement in client server applications.
- The clients can communicate with server through virtual connectivity.

Web Application Testing

Testing approaches of web application testing-→Component Testing

- For testing web application approach and test plan need to be defined.
- One may have to devise simulators to replace corresponding components while testing the component targeted by the test.
- When server is tested, we may need a client simulator, while testing of client may need server simulator.

Web Application Testing

Testing approaches of web application testing-→Integration Testing

- After successful testing of servers, clients and network, they are brought together to form the system and system test cases are executed.
- Communication between client and server is tested in integration testing.

Web Application Testing

Testing approaches of web application testing-→Performance Testing

- System performance is tested when number of clients are communicating with server at a time.
- we can test the system under maximum load as well as normal load expected.

Web Application Testing

Testing approaches of web application testing-→Concurrency Testing

- It may be possible that multiple users may be accessing same record at a time.
- Concurrency testing is required to understand the behavior of a system under such circumstances.

Web Application Testing

Testing approaches of web application testing-→ Disaster Recovery Testing

- When machine and web server are communicating with each other, there exists a possibility of breaking of the communication due to various reasons or failure of either client or server or link connecting them.
- It may involve testing the scenario of such failure at different points in the system and action taken by the system in each case.

Web Application Testing

Testing approaches of web application testing-→ Testing for extended period

- In web application it may be expected that server is running 24*7 for extended period.
- one need to conduct testing over an extended period to understand if service level of network and server deteriorates over time due to some reasons.

Web Application Testing

Testing approaches of web application testing→ Compatibility Testing

- Servers may be in different hardware, software or operating environment than the recommended one.
- Client browser may differ significantly from the expected environmental variables.
- Testing must ensure that performance must be maintained on the range of hardware and software configurations.

Web Application Testing

Testing approaches of web application testing→ Security Testing

- As the communication is through virtual network, security becomes an important issue.
- Application may use communication protocols, coding and decoding mechanism and schemes to maintain security of system.
- System must be tested for possible weak areas called vulnerabilities and possible intruders trying to attack the system.

Performance Testing

- Performance testing is intended to find whether the system meets its performance requirements under normal load or abnormal level of activities.
- Normal load must be defined by the requirement statement defined by the customer and system design implements them.
- Performance criteria must be expressed in numerical terms.
- This is one area where verification does not work to that much extents and one needs to test it by actually performing the operation on the system.

Performance Testing

Load Testing

- Load testing is a type of non-functional testing.
- A load test is type of software testing which is conducted to understand the behavior of the application under a specific expected load.
- Load testing is performed to determine a system's behavior under both normal and at peak conditions.

Performance Testing

Load Testing

- It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation.
- Example : If the software operates on peripherals such as printer, or communication ports, connect as many as you can.
- Example: If you are testing an internet server that can handle thousands of simultaneous connections, do it.

Performance Testing

Load Testing

- Scenario like, most software it is important for it to run over long periods.
- Some software's should be able to run forever without being restarted.

Performance Testing

Stress Testing

- It is a type of non-functional testing.
- **Stress testing** is testing the software under less than ideal conditions.
- So subject your software to low memory, low disk space, slow CPU, and slow modems and so on.

Performance Testing

Stress Testing

- Look at your software and determine what external resources and dependencies it has.
- Stress testing is simply limiting them to bare minimum. With stress testing you starve the software.
- The goals of such tests may be to ensure the software does not crash in conditions of insufficient computational resources (such as memory or disk space).

Performance Testing

Security Testing

- Verifies that protection mechanisms built into a system will, in fact, protect it from improper access.
- As the communication is through virtual network, security becomes an important issue.
- Application may use communication protocols, coding and decoding mechanism and schemes to maintain security of system.

Performance Testing

Security Testing

- System must be tested for possible weak areas called vulnerabilities and possible intruders trying to attack the system.

Acceptance Testing

- Final stage of testing.
- Generally done by end user /customer/third party/system testing people.
- Required document: Test plan/Project plan which defines acceptance criteria.

Acceptance Testing

Acceptance testing validates following:

- Whether user needs defined in SRS are satisfied by System or not .
- Whether system meets performance requirement as specified in SRS.
- It also determines whether the application satisfies its acceptance criteria.

Acceptance Testing

Acceptance testing has two types:

1. Alpha Testing

2. Beta testing



Acceptance Testing

Alpha Testing:

- Testing is done by end user at developer site.
- It is done in natural setting of software.
- Developer is present.
- Over the shoulder approach.
- Usage related error recorded.
- Controlled environment.

Acceptance Testing

Beta Testing:

Aim: Developer can release error free software to customer immediately.

- Testing is done by end user at end user site.
- It is live kind of application testing.
- Developer is not present.

Acceptance Testing

Beta Testing:

- Not done in controlled environment.
- End user get an opportunity to record error.
- Error reporting on day today basis.
- Error those were reported on daily basis get resolved immediately/some times fixed in next release.

Acceptance Testing

Difference between Alpha Testing and Beta Testing

Alpha Testing	Beta Testing
Testing is done by end user at developer site.	Testing is done by end user at end user site.
It is done in controlled environment.	Developer is not present so not done in controlled environment.
Alpha testing is not effective as it is done in controlled environment.	Beta test is more effective as end user do not have any restriction of testing software.
End user may have to test software under influence of developer.	Beta testing is live application testing, not done in influence of developer.

Defect Management (12 Marks)



Program: Computer Engineering

Course Name: Software Testing

Course Code:22518

Course Outcome: Identify bug to create defect report in given application.

Unit Outcome:

- Classify defects on the basis of estimated impact.
- Prepare defect template on the given application.
- Apply defect management process on the given application.
- Write procedure to find defect using given techniques.

Topic/Sub Topic

1. Defect classification
2. Defect management process
3. Defect Life cycle
4. Defect Template
5. Estimated expected impact of the defect
6. Techniques for finding defect
7. Reporting defect

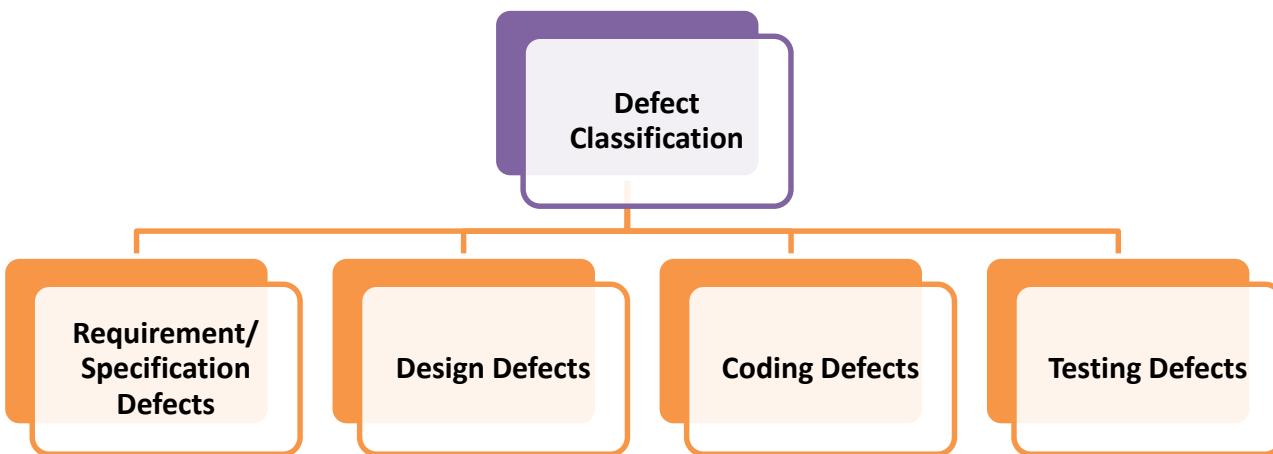
Defect

A Software Defect / Bug is a condition in a software product which does not meet a software requirement or end-user expectations .

OR

Defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results.

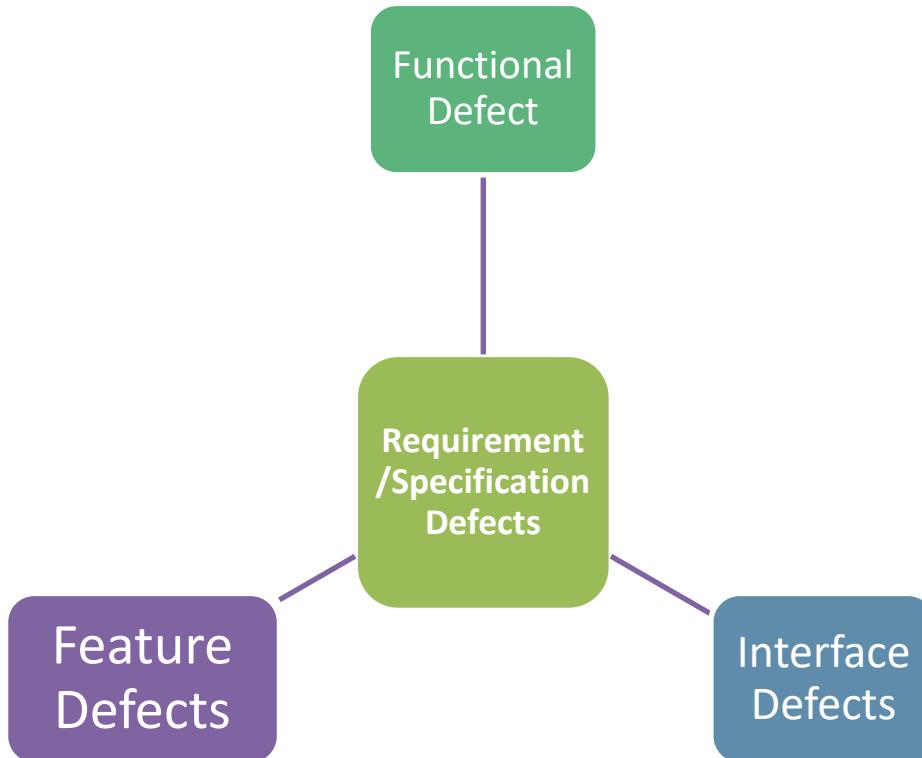
Defect Classification



Defect Classification: Requirement/Specification Defects

- Requirement related defects arise in a product when one fails to understand what is required by the customer.
- These defects may be due to customer gap, where the customer is unable to define his requirements
- Producer gap, where developing team is not able to make a product as per requirements.
- Requirements/Specification documents are written using natural language representation, there are very often occurrences of ambiguous, contradictory, unclear, redundant and imprecise requirements.

Defect Classification: Requirement/Specification Defects



Requirement/Specification Defects: Functional Defects

The overall description of what the product does and how it should behave, is incorrect, ambiguous and/or incomplete

These defects are mainly about the functionalities present / absent in the applications which are expected/not expected by the customer

Non-working functions and functions not provided as required may represent functional defects.

Requirement/Specification Defects: Feature Defects

Features refer to functional aspects of the software that map to functional requirements as described by the users and clients

Features also maps to quality requirements such as performance and reliability.

Feature defects are due to feature descriptions that are missing, incorrect, incomplete or unessential.

Requirement/Specification Defects: Interface Defects

These are defects that occur in the description of how the target software is to interface with external software, hardware and users.

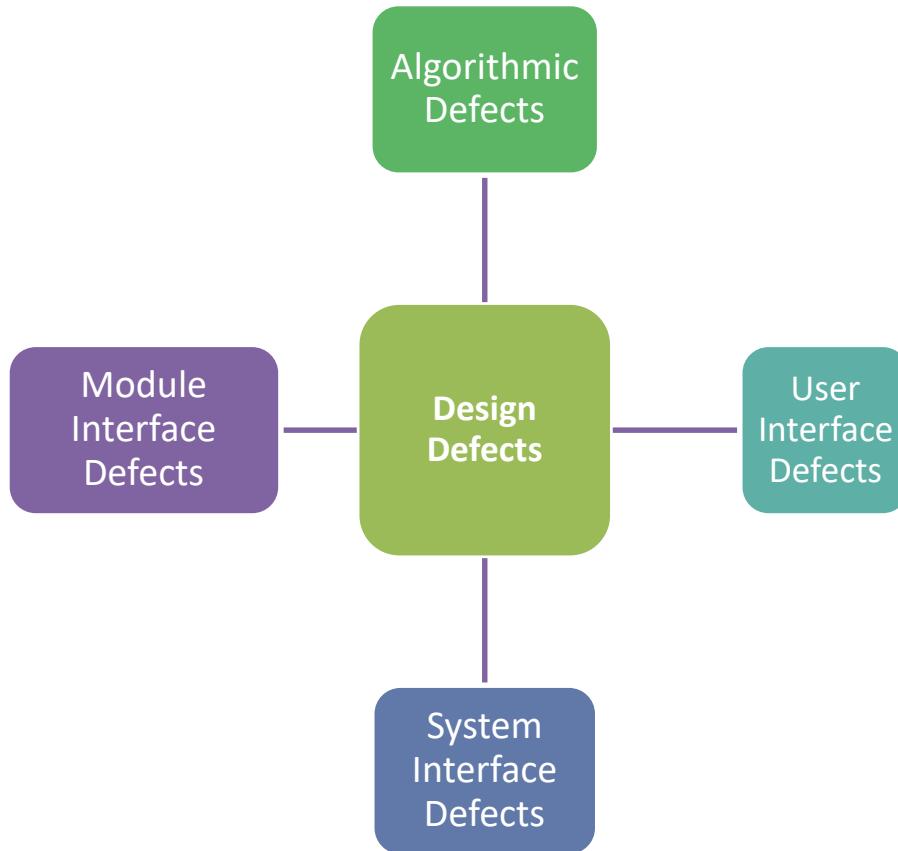
Generally, these defects may be due to user interface problems, problems with connectivity to other systems including hardware etc.

Many features interaction and interface description defects are detected using black box-based test designs at the integration and system levels.

Defect Classification: Design Defects

- Design defects occur when system components, interactions between system components, interactions between the outside software/hardware, or users are incorrectly designed.
- This covers in the design of algorithms, control, logic/ data elements, module interface descriptions and external software/hardware/user interface descriptions.
- Design defects generally refer to the way of design creation or its usage while creating a product.

Defect Classification: Design Defects



Design Defects: Algorithmic Defects

These defects occur when the processing steps in the algorithm are incorrect.

For example pseudo code may contain a calculation that is incorrect specified, or the processing steps in the algorithm are written may not be in the correct order.

Another examples are a step may be missing or a steps may not be duplicated, omission of error condition .

Design Defects: Module Interface Defects

Module interface defects are about communication problem between various modules.

If one module gives some parameters which are not recognized by another, it creates module interface defects.

For example, incorrect and/or inconsistent parameter types, an incorrect number of parameters or an incorrect ordering of parameters.

Design Defects: System Interface Defects

System interface defects may be generated when application communication with environmental factors is hampered.

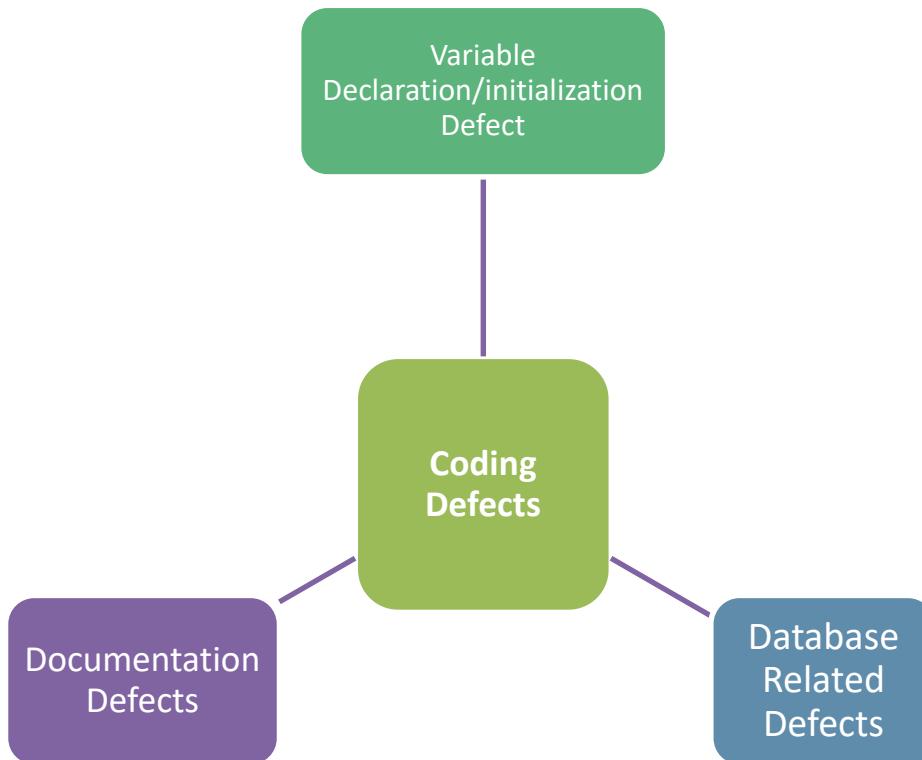
System may not be able to recognize inputs coming from the environment or may not be able to give outputs which can be used by the environment.

Design Defects: User Interface Defects

User interface defects may be a part of navigation, look and feel type of defects which affect usability of an application.

Other examples of user interface description defects are where there are missing or improper commands, improper sequence of commands, lack of proper messages and/or lack of feedback message for the user.

Defect Classification: Coding Defects



Coding Defects: Variable declaration /Initialization Defects

This defect arises when variables are not initialized properly, or variable are not declared correctly.

These types of defects refer to wrong coding practices which may arise due to Coding standards of development are not followed.

Coding Defects: Database related Defects

Database related defects may occur when a database is not created /implemented appropriately.

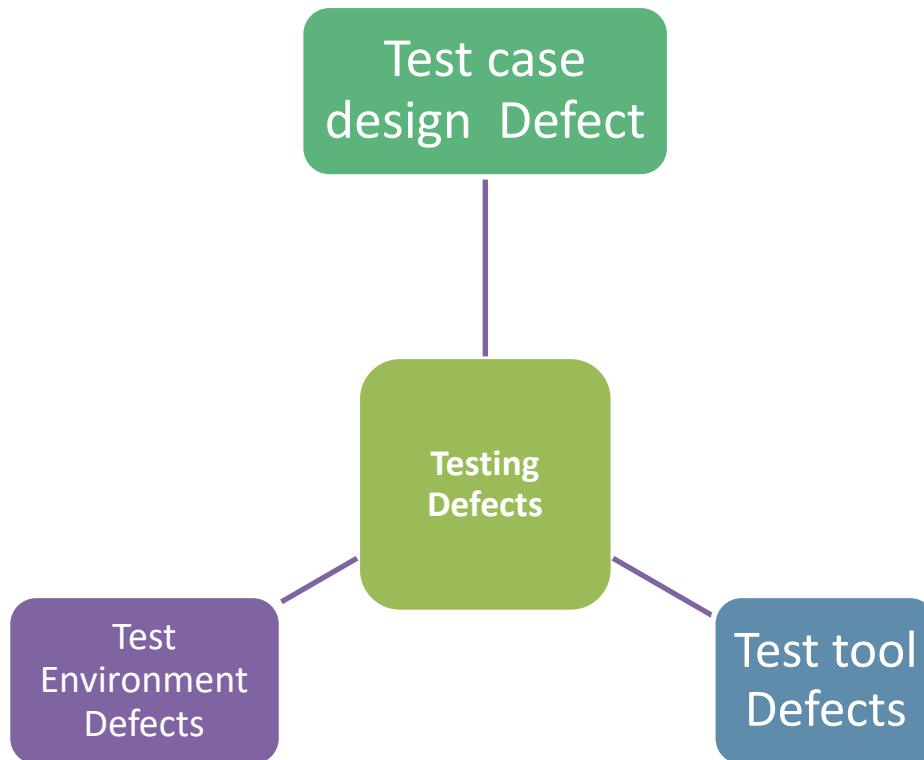
It may be a part of design defects if database design is wrong.

Coding Documentation /Commenting Defects

Coding also needs adequate commenting to make it readable and maintainable in future.

Also there must be adequate documentation associated with various stages of development so that it is useful in future for maintenance

Defect Classification: Testing Defects



Testing : Test Case Design Defects.

These would encompass incorrect, incomplete, missing inappropriate test cases and test procedures.

Testing : Test Environment Defects.

Test environment defects may arise when test environment is not set properly.

Test environment definition and reality are mismatched, it may lead to defects

Testing : Test tool Defects.

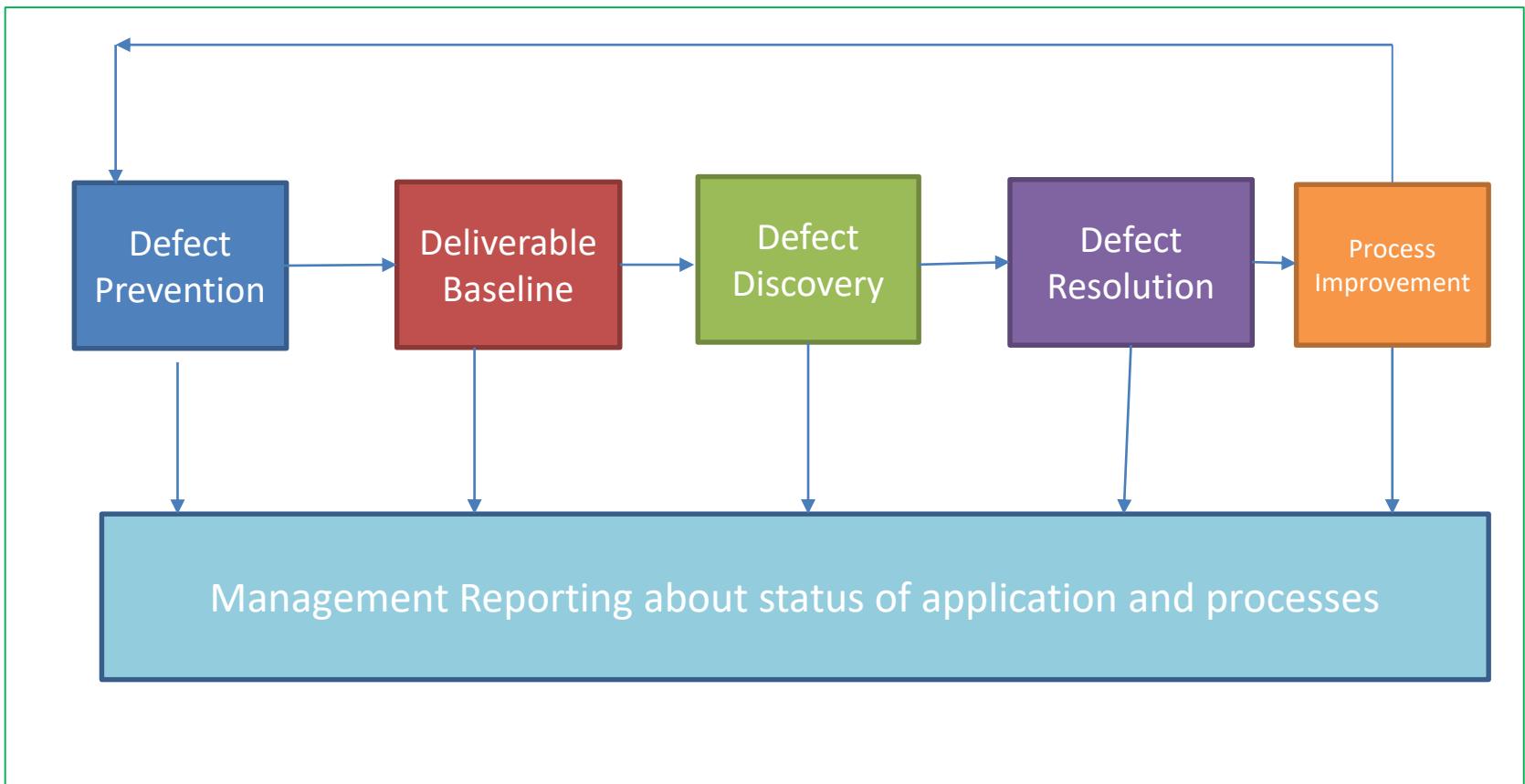
Any defect introduced by a test tool may be very difficult to find and resolve, as one may have to find the defect using manual tests as against automated tools.

Defect may be introduced while writing automation script.

Defect Management Process

- Defects found during Verification and validation process in SDLC must be recorded so that it helps in further analysis and root causes of the defect.
- The defects found during these phases are used to find weak areas of project /process so that action can be initiated to strengthening it.

Defect Management Process



Defect Management Process :Prevention

- It is a process of improving quality and productivity by preventing the defects into a software product.
- It is virtually impossible to eliminate the defects altogether.
- Implementation of techniques, methodology and standard processes are used to reduce the risk of defects.
- Defect prevention is intended to remove the possibility of any defects before it occurs.

Defect Management Process :Deliverable Baselining

- Once the defect is fixed, retested and found to be closed, the product is created again .
- If the newly created product satisfies the acceptance criteria, it is base lined.
- Only base lined work products can go to the next stage.

Defect Management Process : Defect Discovery

- A defect is said to be discovered when it is brought to the attention of the developers and acknowledged (i.e., “Accepted”) to be valid one.
- Team should find defects before they become major problems. As soon as team finds the defects, they should report them so that those can be resolved.
- Team should also make sure that defects should be acknowledged by developers and should be valid one.

Defect Management Process : Defect Resolution

- Work by the development team to prioritize, schedule and fix a defect, and document the resolution.
- This also includes notification back to the tester to ensure that the resolution is verified.

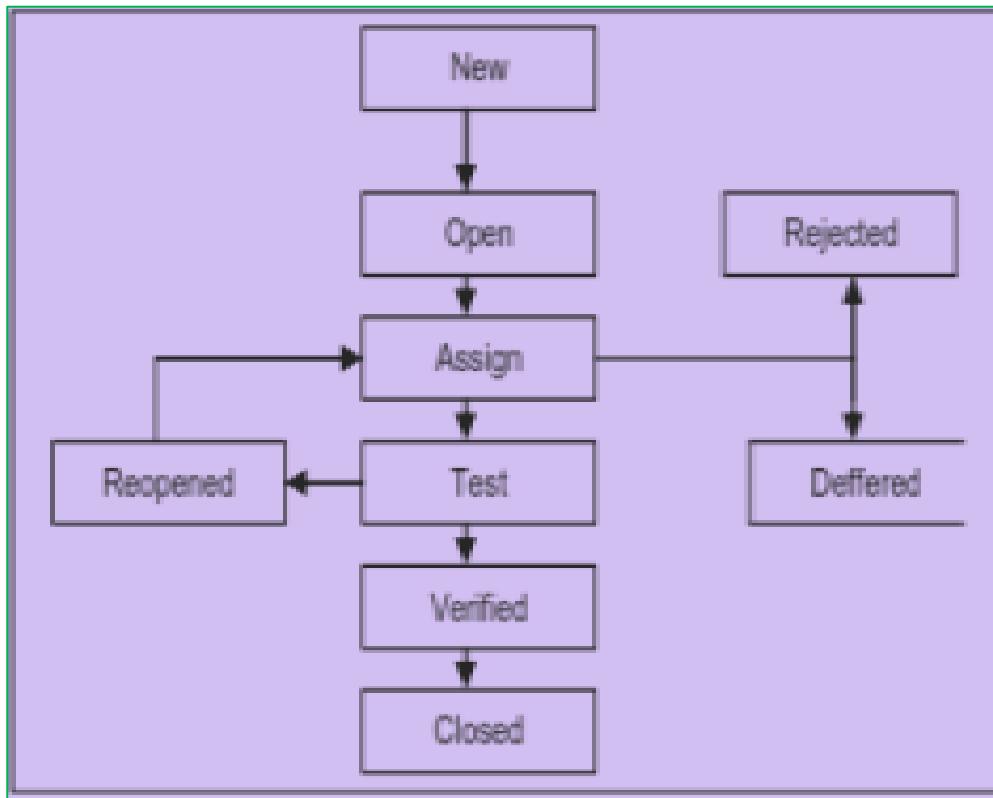
Defect Management Process :Process Improvement

- All problems are due to failure in the process involved in creating software.
- Defects gives an opportunity to identify the problem with process used and update them.
- Better processes mean better product with less defect.

Defect Management Process :Management Reporting

- Analysis and reporting of defect information to assist management with risk management, process improvement and project management.

Defect Life Cycle



Defect Life Cycle

New

- When the bug is posted for the first time, its state will be “NEW”. This means that the bug is not yet approved.

Open

- After a tester has posted a bug, the lead of the tester approves that the bug is genuine, and he changes the state as “OPEN”.

Defect Life Cycle

Assign

- Once the lead changes the state as “OPEN”, he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to “ASSIGN”.

Test/Retest

- Once the developer fixes the bug, he must assign the bug to the testing team for next round of testing. Before he releases the software with bug fixed, he changes the state of bug to “TEST”.

Defect Life Cycle

Deferred

The bug, changed to deferred state means the bug is expected to be fixed in next releases.

Rejected

If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to “REJECTED”.

Defect Life Cycle

Verified

Once the bug is fixed and the status is changed to “TEST”, the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to “VERIFIED”.

Reopened

If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “REOPENED”. The bug traverses the life cycle once again.

Defect Life Cycle

Closed

Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to “CLOSED”.

Fixed

When developer makes necessary code changes and verifies the changes then he/she can make bug status as “Fixed” and the bug is passed to testing team.

Pending retest

After fixing the defect the developer has given that code for retesting to the tester. Here the testing is pending on the testers end. Hence its status is pending retest.

Defect Life Cycle(Optional State)

Duplicate

If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to “duplicate”.

Not a bug

The state given as “Not a bug” if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change

Defect Template

Table: Defect Report Template

ID	Unique identifier given to the defect. (Usually Automated)
Project	Project name.
Product	Product name.
Release Version	Release version of the product. (e.g. 1.2.3)
Module	Specific module of the product where the defect was detected.
Detected Build Version	Build version of the product where the defect was detected (e.g. 1.2.3.5)
Summary	Summary of the defect. Keep this clear and concise.
Description	Detailed description of the defect. Describe as much as possible but without repeating anything or using complex words. Keep it simple but comprehensive.
Steps to Replicate	Step by step description of the way to reproduce the defect. Number the steps.
Actual Result	The actual result you received when you followed the steps.
Expected Results	The expected results.
Attachments	Attach any additional information like screenshots and logs.
Remarks	Any additional comments on the defect.
Defect Severity	Severity of the Defect.
Defect Priority	Priority of the Defect.
Reported By	The name of the person who reported the defect.
Assigned To	The name of the person that is assigned to analyse/fix the defect.
Status	The status of the defect.

Severity

- **Severity:** Your bug report will also include Severity, which describes the impact of the defect on the application.
- Severities may be ‘critical, high, medium or low depending on the impact of a defect. Severity definitions may be as follows:
- 1. S1= Defect where an application fails or particular functionality is completely missing. There is no work around available.
- 2. S2= Defect where functionality is not available but work around is possible. This may make customer unhappy but alternate way to achieve same result does exist. 3.
- S3= Defects of very minor nature (such as cosmetic defects) are in this category. Spelling mistakes, color and font mismatch may be put under least severity defects.

Priority

- Priority is defined based on how the project decides a schedule to take the defects for fixing.
- It is defined based on how many times the defect can be seen by normal users under normal circumstances, or how many customers are being affected due to this.
- Priority which is related to defect fixing urgency. Priority could be High/Medium/Low based on the impact urgency at which the defect should be fixed, respectively.
- **Priority definitions are:**
- 1. P1= Defects having highest probability of affecting the customer. Either it affects most of the users or it appears at a place where maximum users will be visiting.
- 2. P2= Defects having lesser probability of affecting customer than P1 defects. They may be appearing in circumstances and hitting customer at fewer instances.
- 3. P3= Defects having minimum probability of affecting customer.

Estimate Expected Impact of a Defect

- Actual impact of a defect can be measured when the risk realizes or becomes a reality in production environment.
- Estimation may be done by different methods/approaches to find the probability of risk occurrence and impact when it becomes reality.
- Some organizations categories risk impact as high, medium and low.

Ways to handle risk

Accept the Risk as It Is:

- Some risks may not have any solutions. e.g., natural disasters.
- The actions to reduce the probability and/or impact of certain risks may be very costly; hence the organization may not be able to implement them.
- These risks may be accepted by the organization as it is. Such decisions are generally taken by the senior management.
- They may prepare a fallback arrangement but may not define the ways of minimizing/eliminating risks.

Ways to handle risk

Accept the Risk as It Is:

- This makes a team psychologically prepared to accept the risk so that the impact on the organization/project/user can be reduced to some extent.
- Listing of known issues or defects in the product indicates acceptance of risk by the management.
- Customer user are given information about probable failures and effect of such failures.
- This is a methodology of declaring ‘accident prone zone’ to users.

Ways to handle risk

Bypassing the Risk:

- If the approach is very risky to the users/customer, the management may decide to bypass the risk by avoiding the approach.
- Bypassing of risk is required when the risk faced by the user cannot be accepted, or no action can be taken to reduce probability/impact arising due to realization of risk.

Minimize Risk Impact or Probability

- Risk is a product of probability, impact, and detection ability.
- Risk minimization has three different methods of handling its probability, impact, or detection ability.
- The decisions may be driven by organization policy, values, cost-benefit analysis etc..

Ways of Minimization of problem due to risk

Eliminate Risk:

- Elimination of risk involves taking steps to remove risk from the root.
- Risk's probability is reduced to almost '0' by removing the causes of risk, so that the risk must not happen at all.
- Organization user may be protected from the possible losses arising due to risk.

Ways of Minimization of problem due to risk

Eliminate Risk:

- Preventive controls can eliminate the probability of risk to a large extent.
- Preventive controls are management-decided controls.
- Preventive controls are applied if the probability of occurrence is very high.
- Preventive controls are useless if the probability of happening is already negligible.

Ways of Minimization of problem due to risk

Mitigation of Risk

- Actions initiated by an organization to minimize the possible damage due to realization of risk are considered mitigation actions.
- Mitigation actions are planned by an organization/project so that if the risk is realized, the impact due to it can be reduced to minimum possible.
- The corrective controls used from the mitigation action.
- Corrective controls may be auto corrective or suggestive.

Ways of Minimization of problem due to risk

Detection Ability Improvements:

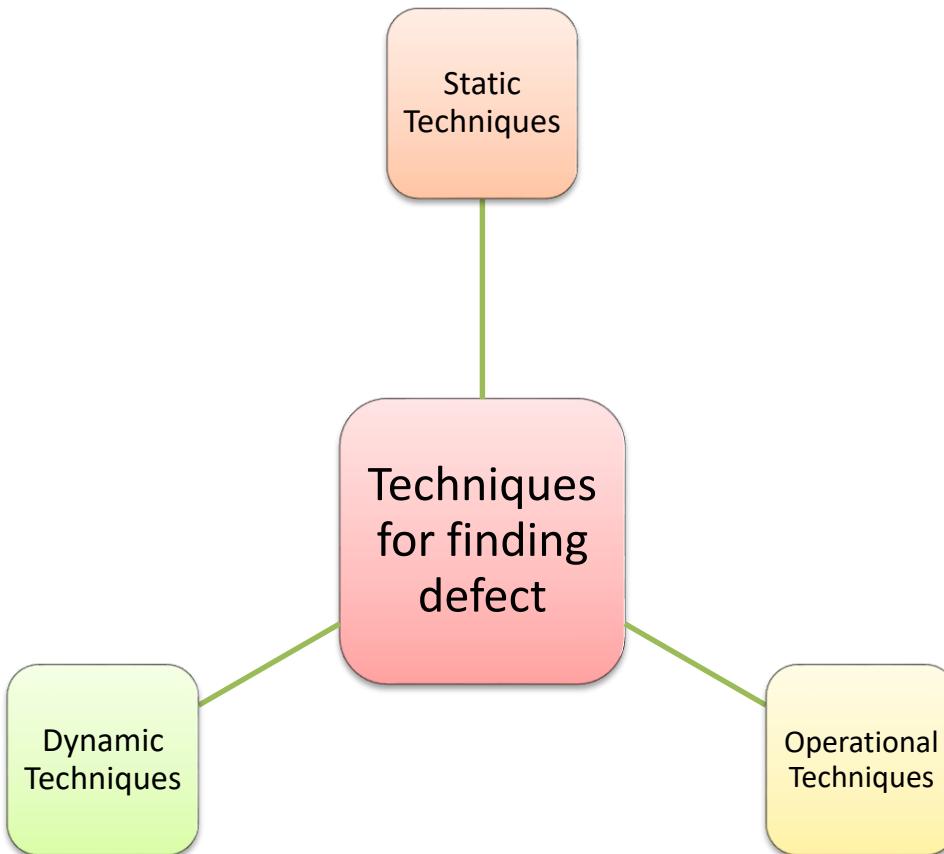
- Impact of a risk is more if it catches the user unprepared.
- If people are aware of the risks, they can be well prepared to handle them.
- Generally, detective controls are used to increase the visibility towards risks.
Sometimes, detective controls give threshold to corrective controls.

Ways of Minimization of problem due to risk

Contingency Planning:

- Contingency planning refers to the actions initiated by an organization, when preventive or corrective actions fail, and risk occurs.
- They are previously planned ways of tackling risks when all other planned activities for reducing probability and impact of the risk fail, and the risk becomes reality.

Techniques for finding defect



Techniques for finding defect -→Static Techniques

Testing that is done without physically executing a program or system.

No execution of code, product, documentation.

Helps in establishing conformance to requirements view

Techniques for finding defect -→Static Techniques

Checking the software product and related artifacts without executing them.

The work product is reviewed by the reviewer with the help of a checklist, standards, any other artifact, knowledge, and experience, to locate the defect with respect to the established criteria.

It may include reviews, walkthroughs, inspection, and audits.

Techniques for finding defect → Dynamic Techniques

Testing in which system components are physically executed to identify defects.

Execution of test cases is an example of a dynamic testing technique.

Dynamic testing is a validation technique which includes dummy or actual execution of work products to evaluate it with expected behavior.

Techniques for finding defect → Dynamic Techniques

It includes black box testing methodology such as system testing.

The testing methods evaluate the product with respect to requirements defined; designs created and mark it as 'pass' or 'fail'.

This technique establishes 'fitness for use' view.

Techniques for finding defect → Operational Techniques

Operational system produces a deliverable containing a defect found by users, customers, or control personnel .

i.e., the defect is found because of a failure..

Operational techniques typically include auditing work products and projects.

To understand whether the processes defined for development/testing are being followed correctly or not, and whether they are effective or not.

Techniques for finding defect → Operational Techniques

It also includes revisiting the defects before and after fixing and analysis.

Operational technique may include smoke testing and sanity testing of a work product.

Reporting a Defect

- Defect finding and reporting are the important steps in software development life cycle.
- When we find defect, analyses it, take corrective and preventive actions for further steps.

Points for defects reporting are:

- Give Complete Record of Discrepancies:
- Complete description of a defect indicates the symptom observed by the tester.
- when the defect is located, corrective actions and preventive actions.
- Project manager must conduct complete analysis of how the defect occurred and what was not done correctly.
- They should perform root-cause analysis to identify and correct the defect and responsible processes which have resulted into the defect.
- Defect finding must lead to process improvement.

Defect management may go through the following stages:

Correct the Defect:

- Correcting the defect logged by the reviewer/tester in verification/validation during software development life cycle
- It is the primary activity done by the author or developer responsible for defect fixing.
- Corrected detect may undergo review, unit testing and retesting.
- In case it is not possible to correct the defect or it is not advisable to do so, it may be noted as known issue in release note.

Defect management may go through the following stages:

Report Status of System:

- The status of system with respect to organizational standards and acceptance criteria fulfillment can be assessed through status reporting.
- Number of defects found and fixed is a major measurement to establish the status of software.
- It helps in informing users, customer, and management about the readiness of an application for delivery.
- Requirements of any efforts needed for fixing them, and any level of retesting and regression testing required.

Defect management may go through the following stages:

Gather Statistics to Predict Future:

- Defect statistics is used by an organization to plan corrective and preventive actions (CAPA) at project/organizational level.
- It can be used for benchmarking a product as well as an organization and assessing the quality improvement programs initiated by the organization.
- Matured organizations use defect calculators for predicting future and initiate actions to improve the process capabilities.

Defect management may go through the following stages:

Process Improvement:

- Testing is not the most efficient way of improving software quality.
- Verification and validation activities are costly.
- Finding and fixing defects cannot improve quality of software as well as productivity of team.
- An organization must stress on quality improvement programs through process improvements to ensure that defects are not introduced in the system

Thank You

Supriya Kadam

Department of Computer Engineering (NBA Accredited)

Vidyalankar Polytechnic

Vidyalankar College Marg, Wadala(E), Mumbai 400 037

E-mail: supriya.angne@vpt.edu.in



Test Management

(10 Marks)



Program: Computer Engineering

Course Name: Software Testing

Course Code:22518

Course Outcome: Prepare Test Plan for Application

Unit Outcome:

Prepare test plan for given application.

Identify the resource requirement of given application.

Prepare test case for given application.

Prepare test report of executed test cases for given application.

Topic/Sub Topic

1. Test Planning
2. Test Management
3. Test Process
4. Test Reporting

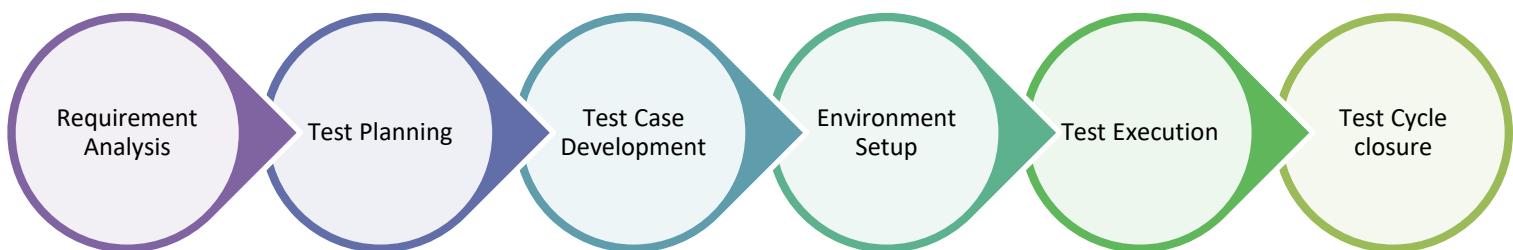
What we will learn today

1. Test Planning
2. Test Management
3. Test Process
4. Test Reporting

Key Takeaway

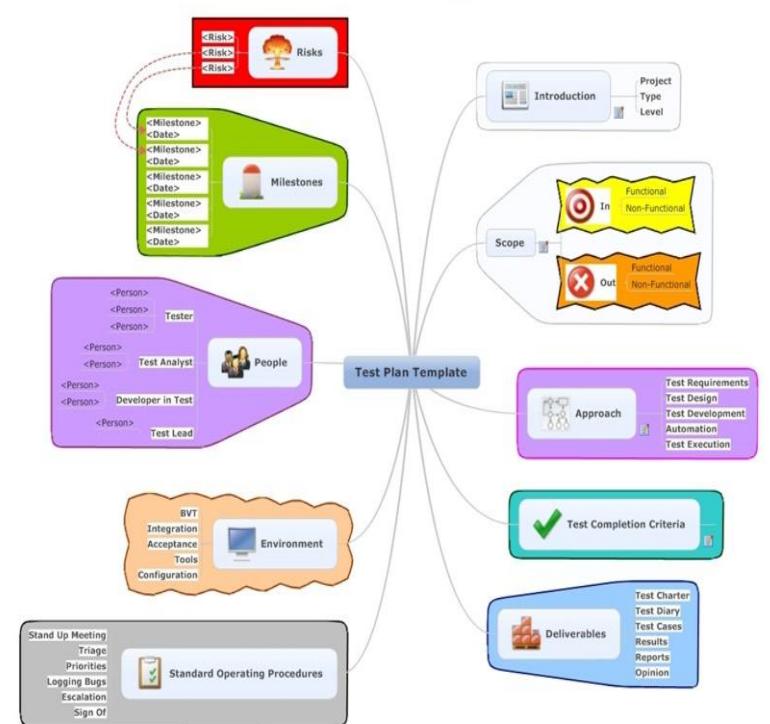
Test Management

Concept Map



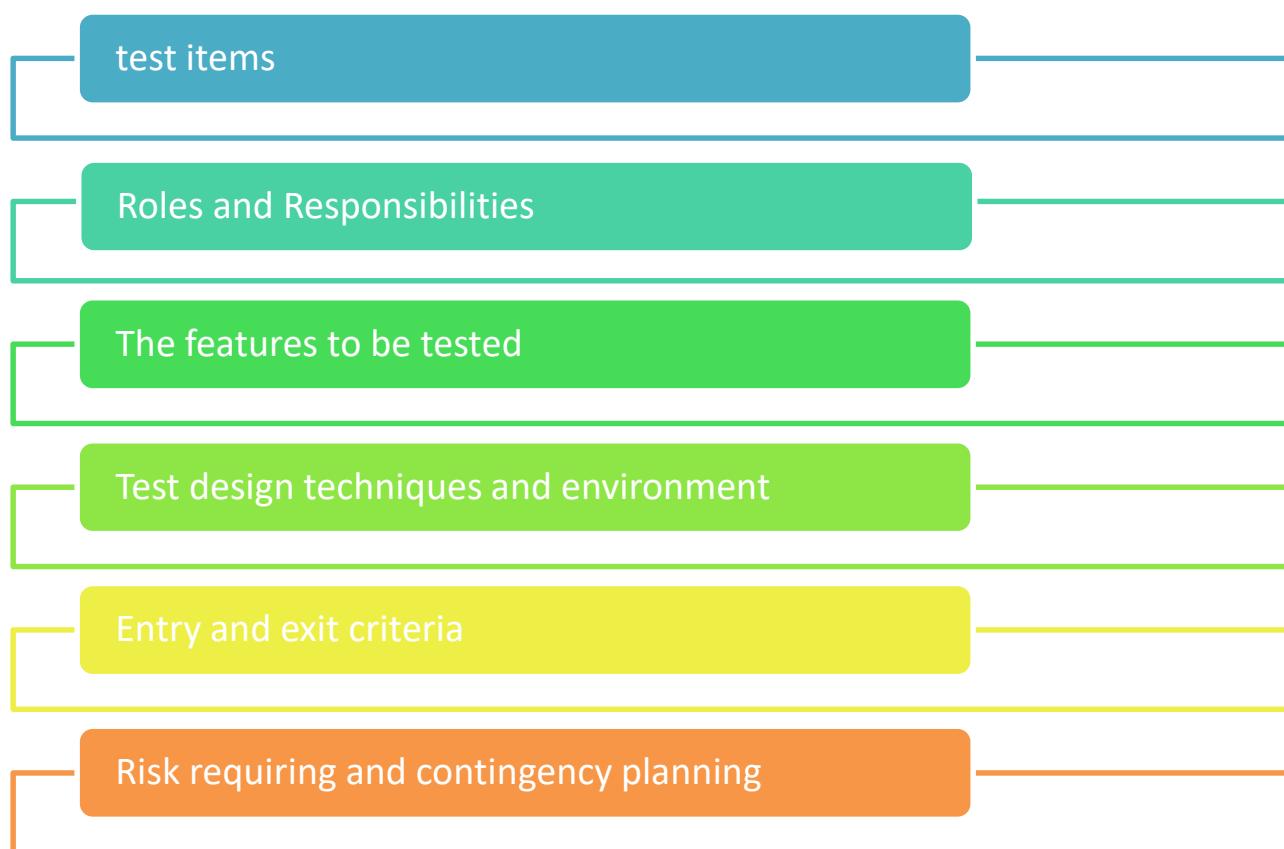
Test Plan:

Test Plan Template



- A document describing the scope, approach, resources and schedule of intended test activities.
- It is a record of the test planning process.

Test Planning → Preparing test plan



Test Planning → Scope management

- Understanding what supports a release of product
- Breaking down release into features
- Prioritizing the features for testing
- Deciding which features will be tested and which will not be
- Gathering details for estimating resources for testing.

Test Planning → Scope management

The following factors drive the choice and prioritization of features to be tested

1) Features that are new and critical for the release:

- New feature will have new program code and thus have higher susceptibility and exposure to defects.
- Since features are new both development and testing team will have to go through a learning phase.
- So these features are in high priority list for testing.
- Here product marketing team and selected customers participate in identification of features to be tested.

Test Planning → Scope management

The following factors drive the choice and prioritization of features to be tested

2) Features whose failures can be catastrophic:

Any feature which have highest or adverse impact on business has to be high on the list of features to be tested.

3) Features that are expected to be complex to test:

Early participation by testing team can help identify features that are difficult to test.

This can help in starting the work on these features early.

Test Planning → Scope management

The following factors drive the choice and prioritization of features to be tested

4) Features which are extensions of earlier features that have been defect prone:

- In regression testing, certain areas of code tend to be defect prone and such areas need detailed testing so that old defects will not come again or occur again.

Test Planning → Deciding test approach

- What type testing would you use for testing the functionality?
- What are the scenarios for testing the features?
- Which type integration testing will be performed ?
- What type localization would be needed ?
- What “non – functional” test would need to do?
- Specify the metrics to be collected.
- Identify significant constraint on testing, such as test-item availability and deadline.

Test Planning → Setting up criteria for testing

- For each phase of testing there must be entry and exit criteria.
- The entry criteria for test specify threshold criteria for each phase or type of test.
- The exit criteria specify when a test cycle or testing activity can be complete.
- Suspension criteria specify when a test cycle or test activity can be suspended.
- Resumption criteria specify when the suspended test can be resumed.

Test Planning → Identifying Responsibilities

Identify the groups responsible for managing, designing, preparing, executing, checking and resolving.

Identify the groups responsible for providing the test item identified in the test item section.

Identify the groups responsible for providing the environmental needs.

Test Planning → Staffing and Training needs

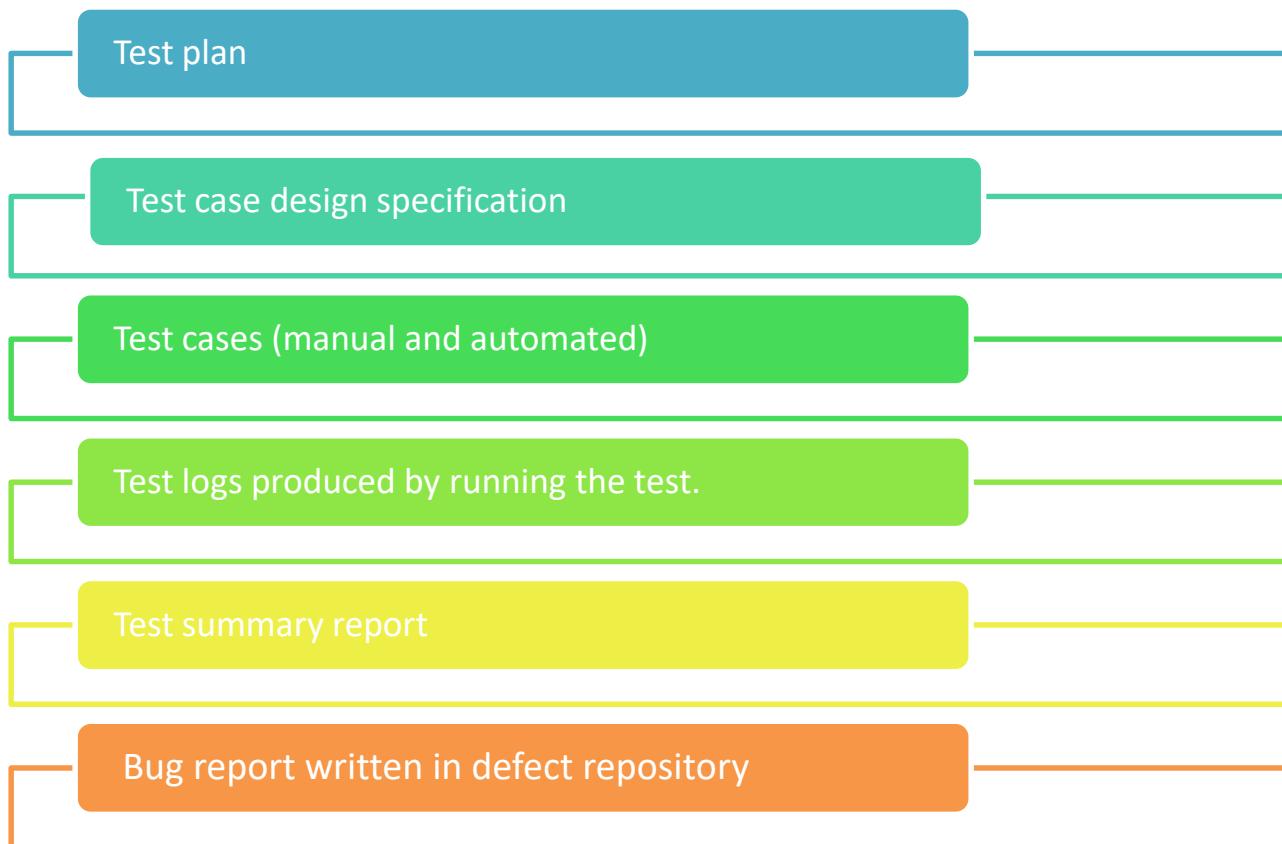
Specify staffing needs by skill level.

Identify training needs for providing necessary skill.

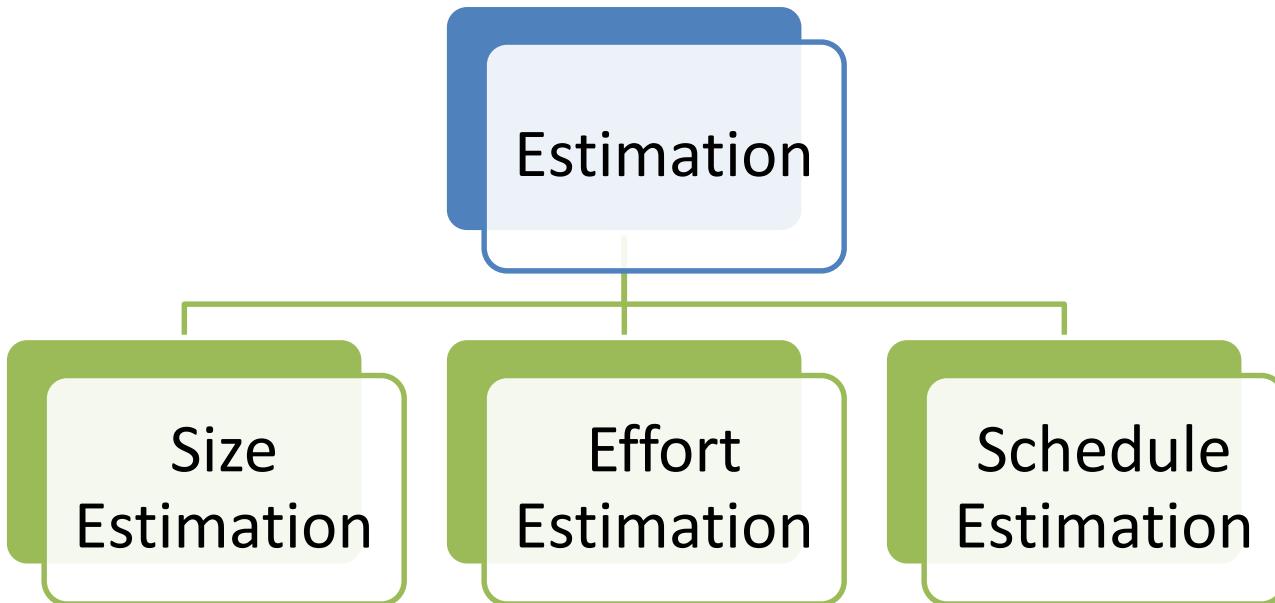
Test Planning → Resource Requirement

- Machine configuration needed to run product under test.
- Different configuration or versions of the software that must be present.
- Need of automation tool like load runner, win runner, QTP etc.
- Supporting tools such as compilers, test data generators, configuration management tools and so on.
- Appropriate number of licenses of all software.

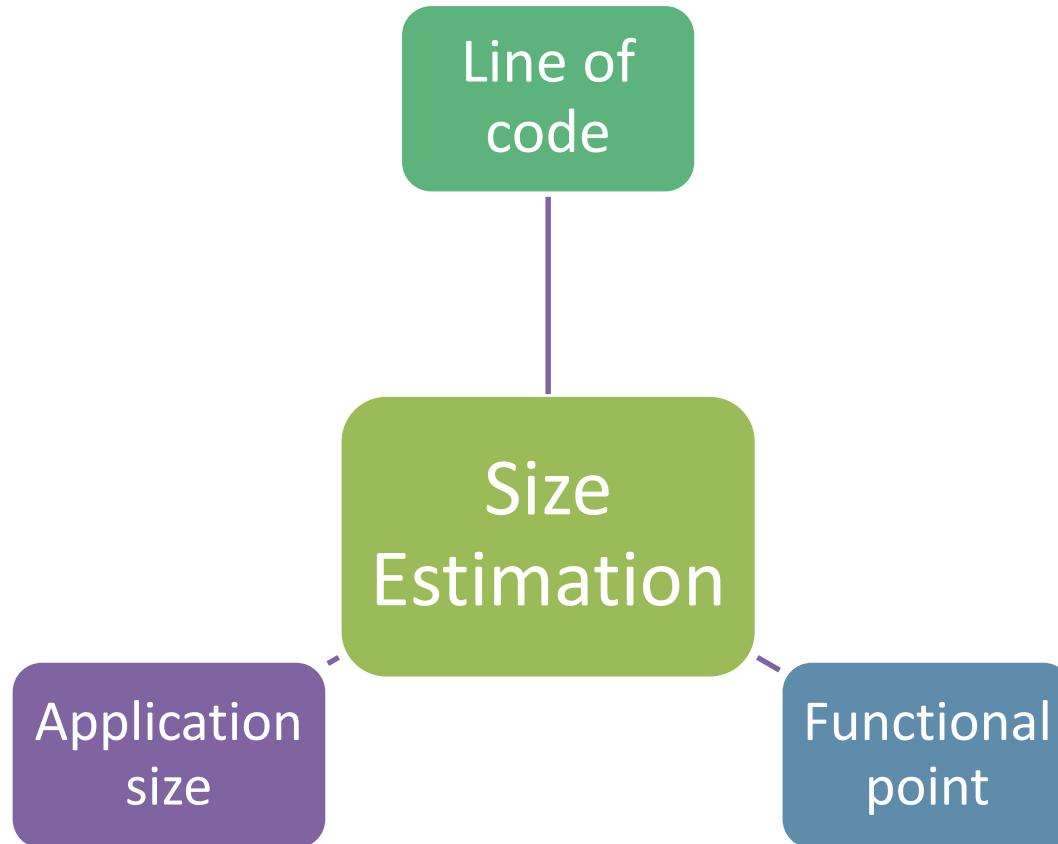
Test Planning → Test Deliverable



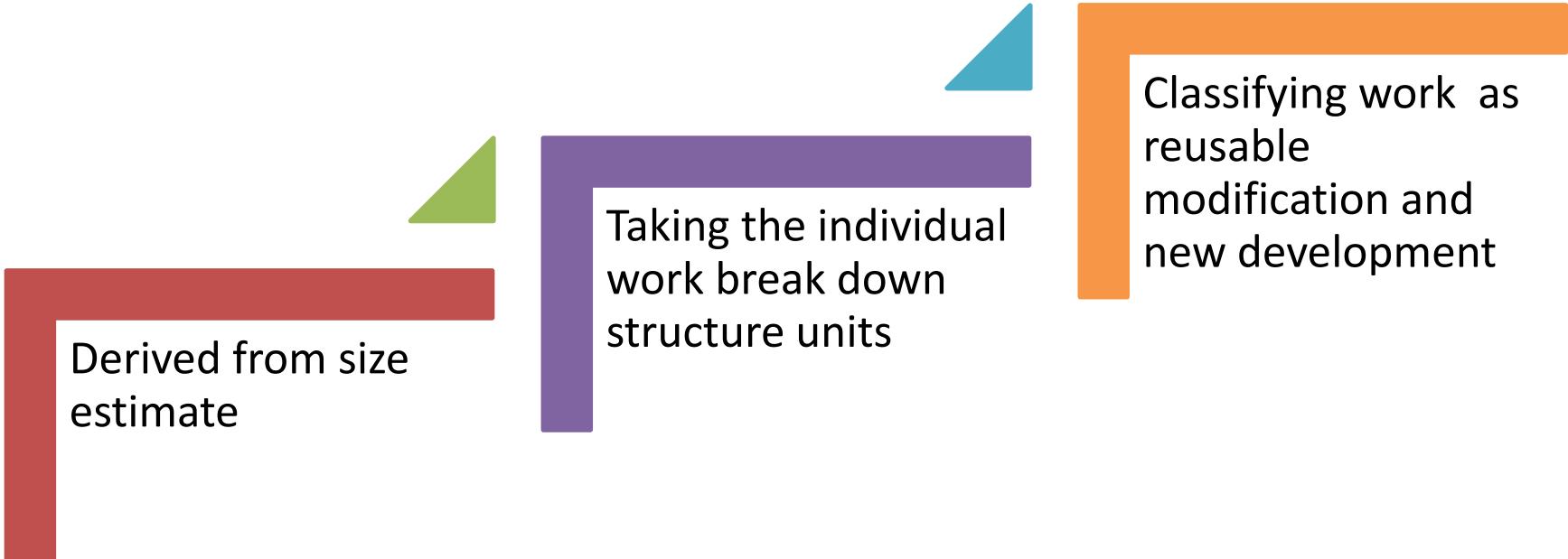
Test Planning → Testing task → Estimation



Test Planning → Testing task → Size Estimation Method



Test Planning → Testing task → Effort Estimation



Derived from size estimate

Taking the individual work break down structure units

Classifying work as reusable modification and new development

Test Planning → Testing task → Schedule Estimation



Test Management:



- Test management is a process of managing testing activities, such as planning, execution, monitoring, and controlling activities.

Test Management → Choice of standards

Standards are of two types:

- **External standards**:-are standards that a product should comply with, are externally visible and usually stipulated by external consortia.
- Compliance to external standards is usually mandated by external parties.
Example: ISO

- **Internal standards**:-are standards formulated by testing organization to bring consistency.
- It standardize the processes and methods of working within the organization

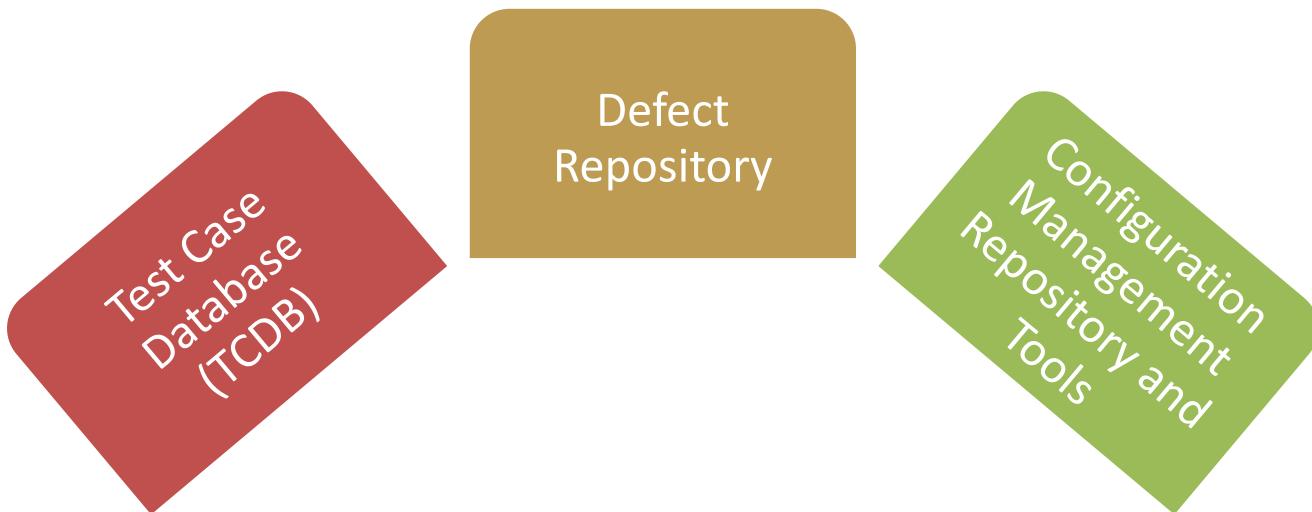
Test Management → Choice of standards

Internal standards include

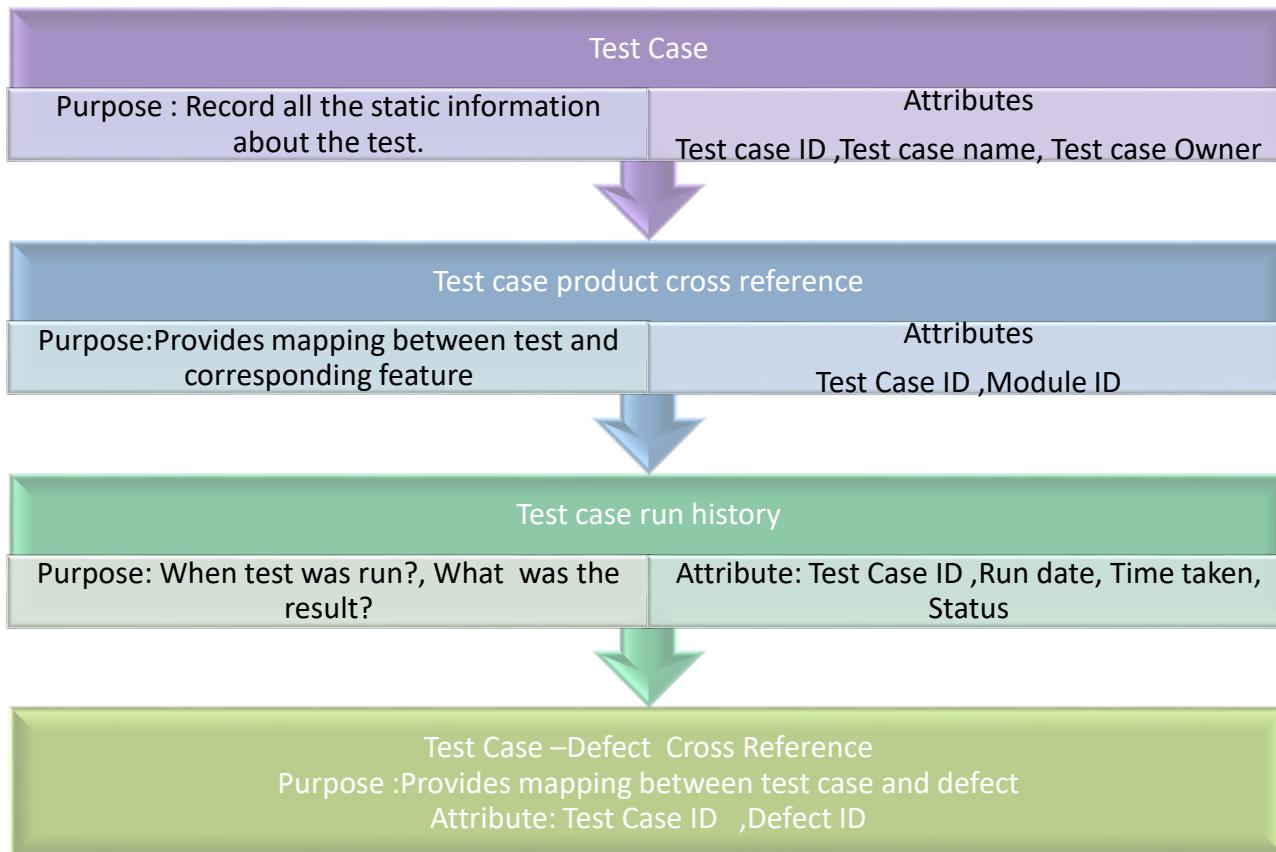
1. Naming and storage convention for test artifacts:- Every test artifact like test cases, test result, test specification and so on must be named appropriate and meaningfully.
2. Documentation standards:- Documentation standard specify how to record information about the test within the test script themselves.
3. Test coding standards:- Test coding standards go one level deeper into the test and it feels how the test should be written.

Test Management → Test Infrastructure Management

Testing infrastructure is made up of three essential elements:



Test Infrastructure Management → TCDB



Test Infrastructure Management → Defect Repository

- It captures relevant details of defect.
- It is tool of communication.
- Defect Metrics are derived from defect repository.

Test Infrastructure Management → Configuration management repository and tools.

Keeps track of change control of all the files/entities that makeup a software product.

Keeps track of version control of all the files/entities that makeup a software product.

Test people management

People management is an integral part of any project management.

A person relies on his or her own skills to accomplish an assigned activity.

People management also requires the ability to hire, motivate and retain the right people .

Success of testing organization depends on careful people management skills .

Integration with product release

- Success of a product depends on the effectiveness of integration of the development and testing activities.
- Both team should work in co-ordination with each other.
- Schedule of testing have to be considered or linked to product release.

Integration with product release

Following points to be decided for this planning:

1. **Synchronization** between testing and development as to when different types of testing can start. For example When unit testing or system testing could start and so on.
2. **Services level agreement** between development and testing as to how long it would take for testing team to complete the testing.
3. **Definition** of the various **priorities and severities** of the defects. Development and testing team should have same vision.
4. **Establish communication mechanisms** to ensure that the document is kept in sync with product development and testing.

Test Process

- Testing is not a single activity instead it's a set of number of processes.
- **It includes following activities:**
 1. Base Lining a Test Plan
 2. Test Case Specification
 3. Update of Traceability Matrix

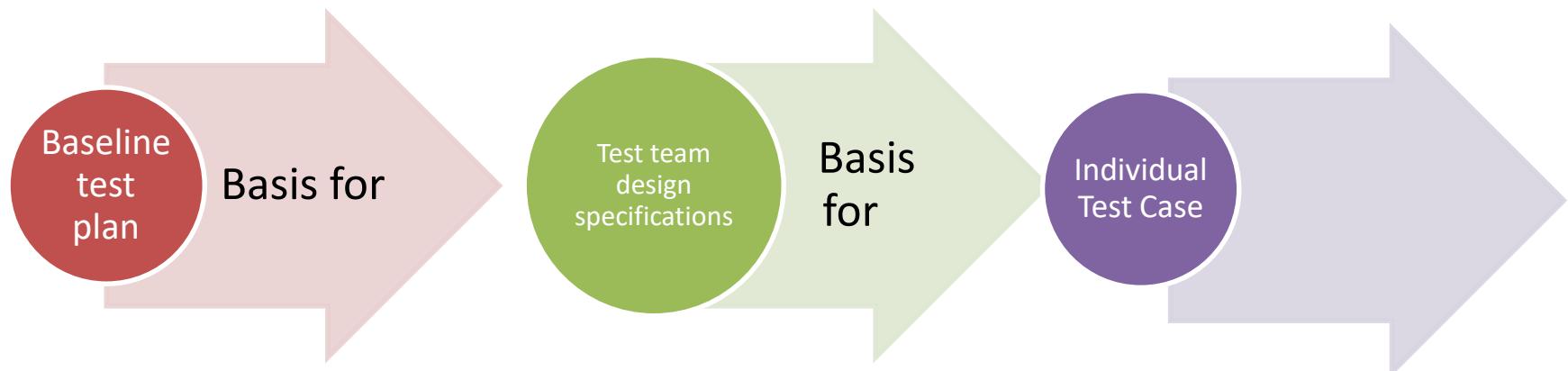
Base Lining test plan

- Every organization have template of test plan.
- The test plan is reviewed by a designated set of competent authority in organization.
- After this plan is approved by authority who is responsible for testing.

Base Lining test plan

- After this, the test plan is baseline into the configuration management repository.
- From then baseline test plan becomes the basis for running the testing project.
- Any significant changes in the testing project should thereafter be reflected in the test plan .
- Changed test plan baseline again the configuration management repository.

Test Case Specification



Test Case Specification

Test case specification should clearly identify following :

1. The purpose of the test:-This will list which feature the test is intended for
2. Items being tested along with their version release number
3. Environment that needs to be set up for running the test case.eg hardware environment set up, software environment set up
4. Input data to be used for the test case. This choice is depending on technique followed in the case. For example manual or automated or type of testing (equivalence portioning or boundary value analysis)

Test Case Specification

Test case specification should clearly identify following :

5. Steps to be followed to execute test
6. The expected result that are correct results
7. A step to compare the actual result produced with the expected result
8. Any relationship between this test and other tests:-by this we can find dependencies among the test

Requirement Traceability Matrix

Traceability matrix is tool to validate that every requirement is tested. It provides mapping between requirement and test cases.

- The traceability matrix is created during the requirement gathering phase.
- Unique identifier is assigned to each requirement.
- As project proceeds through design phase identifier for design feature is entered in matrix.

Requirement Traceability Matrix

- When project enters coding phase identifier for program file is entered in the traceability matrix.
- When test case specification is complete the row corresponding to the requirement which is being tested by the test case is updated with the test case specification identifier.

Requirement Traceability Matrix

- **Testability matrix helps in:**
 - Ensuring 100% test coverage
 - Showing requirement/document inconsistencies
 - Displaying the overall defect/execution status with focus on business requirements.
 - Template for RTM(Requirement Traceability matrix) :

Unique No	Require- mnt	Source of Requirement	Design Spec.	Program Module	Test Spec.	Test Case	Successful Test Verification	Modification of Requirement	Remark

Test Reporting



Executing Test Cases

Proper execution of test cases is essential which will minimize the work and reduce the time to release s/w product.

Test execution task:

1. Follow the test procedures to execute test cases.
2. Do the confirmation testing for the failed test cases.
3. Log the result for test execution.

Executing Test Cases

4. Compare actual and expected results. In case of difference defect occurrence is reported.
5. Update defect database which is used to communicate between developer and tester team.
6. So the execution of test cases will decide the suspension or resumption of further test cases.

Test Summary Report

- Test summary report is a document which contains summary of test activities and final test results.
- After the testing cycle it is very important that you communicate the test results and findings to the project stakeholders .
- so that decisions can be made for the software release. i.e. If further testing is required and we need to delay the release.

Test Summary Report

- Test summary report will be different for different kind of testing.
- In addition to test coverage and unresolved defects test summary reports should also contain test strategy, test objectives and overall result of test effort.

References

<https://www.gcreddy.com/2014/07/software-test-plan-templates.html>

<https://www.kualitee.com/test-management/best-test-management-tools-must-used-2019/>

Thank You

Supriya Kadam

Department of Computer Engineering (NBA Accredited)

Vidyalankar Polytechnic

Vidyalankar College Marg, Wadala(E), Mumbai 400 037

E-mail: supriya.angne@vpt.edu.in



Test Management

(10 Marks)



Program: Computer Engineering

Course Name: Software Testing

Course Code:22518

Course Outcome: Prepare Test Plan for Application

Unit Outcome:

Prepare test plan for given application.

Identify the resource requirement of given application.

Topic/Sub Topic

1. Test Management

What we will learn today

1. Test infrastructure management
2. Test People management

Key Takeaway

Test Management

Test Management:



- Test management is a process of managing testing activities, such as planning, execution, monitoring, and controlling activities.

Test Management → Choice of standards

Standards are of two types:

- **External standards**:-are standards that a product should comply with, are externally visible are usually stipulated by external consortia.
- Compliance to external standards is usually mandated by external parties.
Example: ISO

- **Internal standards**:-are standards formulated by testing organization to bring consistency.
- It standardize the processes and methods of working within the organization

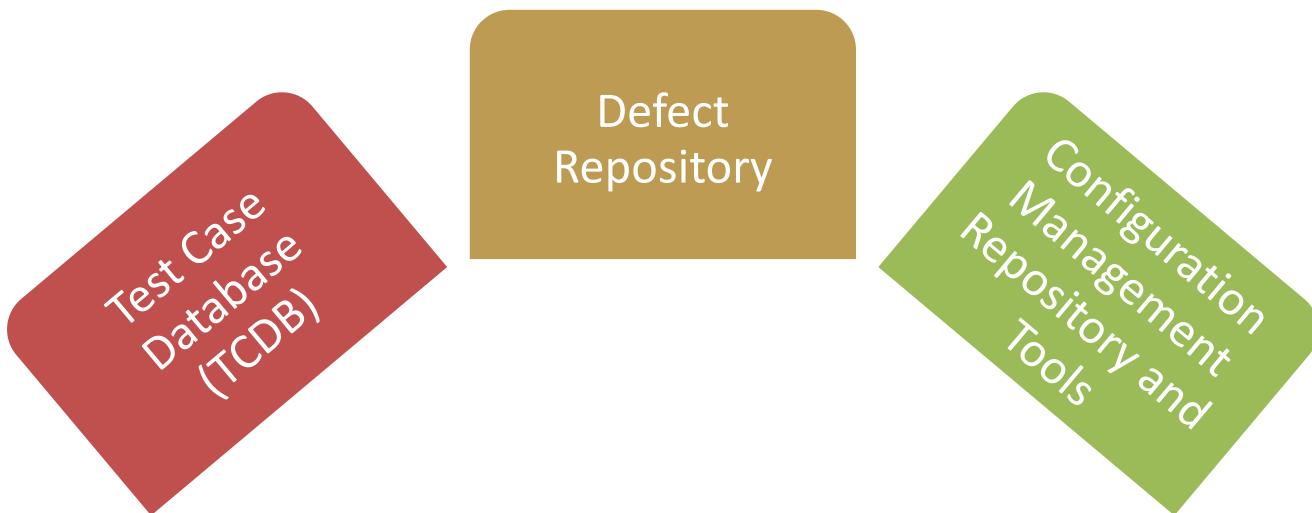
Test Management → Choice of standards

Internal standards include

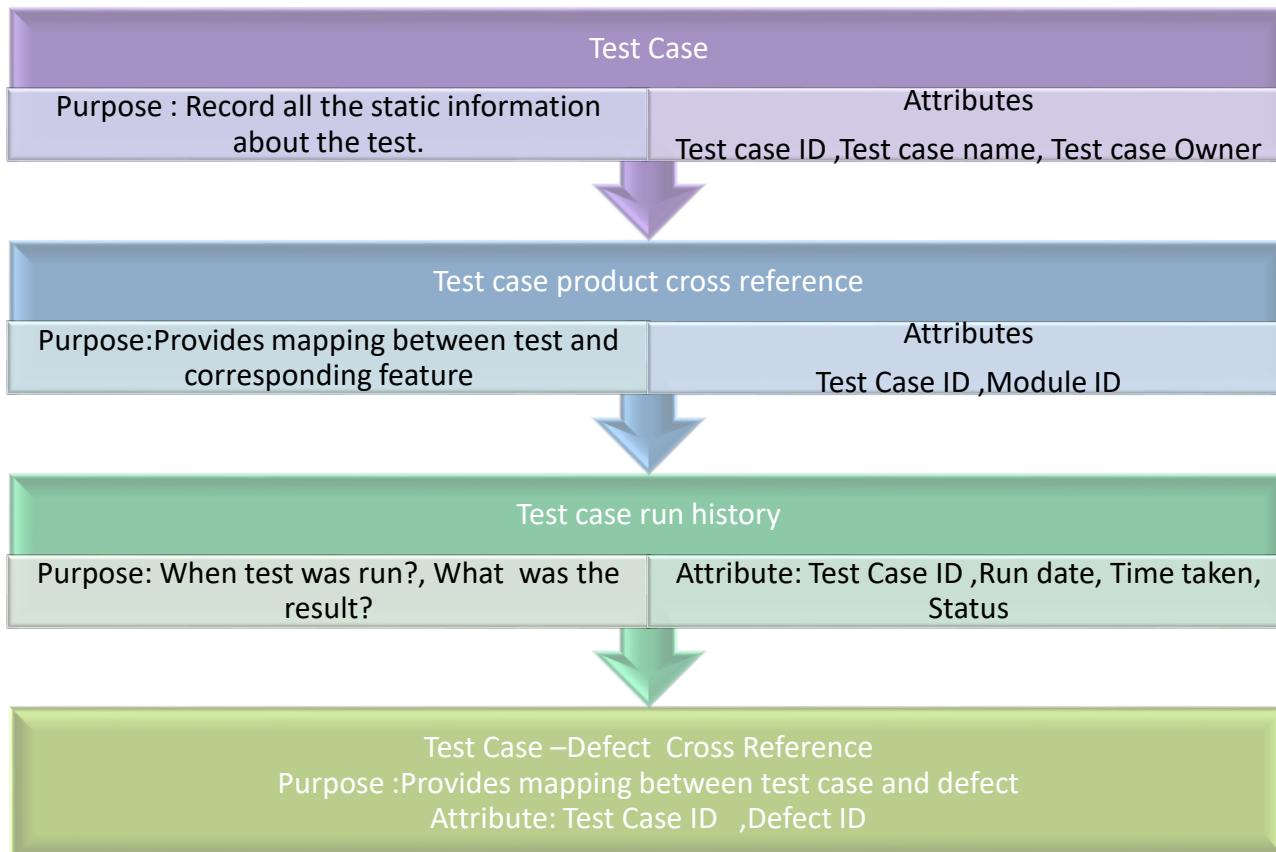
1. Naming and storage convention for test artifacts:- Every test artifact like test cases, test result, test specification and so on must be named appropriate and meaningfully.
2. Documentation standards:- Documentation standard specify how to record information about the test within the test script themselves.
3. Test coding standards:- Test coding standards go one level deeper into the test and it feels how the test should be written.

Test Management → Test Infrastructure Management

Testing infrastructure is made up of three essential elements:



Test Infrastructure Management → TCDB



Test Infrastructure Management → Defect Repository

- It captures relevant details of defect.
- It is tool of communication.
- Defect Metrics are derived from defect repository.

Test Infrastructure Management → Configuration management repository and tools.

Keeps track of change control of all the files/entities that makeup a software product.

Keeps track of version control of all the files/entities that makeup a software product.

Test people management

People management is an integral part of any project management.

A person relies on his or her own skills to accomplish an assigned activity.

People management also requires the ability to hire, motivate and retain the right people .

Success of testing organization depends on careful people management skills .

Integration with product release

- Success of a product depends on the effectiveness of integration of the development and testing activities.
- Both team should work in co-ordination with each other.
- Schedule of testing have to be considered or linked to product release.

Integration with product release

Following points to be decided for this planning:

1. **Synchronization** between testing and development as to when different types of testing can start. For example When unit testing or system testing could start and so on.
2. **Services level agreement** between development and testing as to how long it would take for testing team to complete the testing.
3. **Definition** of the various **priorities and severities** of the defects. Development and testing team should have same vision.
4. **Establish communication mechanisms** to ensure that the document is kept in sync with product development and testing.

Test Process

- Testing is not a single activity instead it's a set of number of processes.
- It includes following activities:
 1. Base Lining a Test Plan
 2. Test Case Specification
 3. Update of Traceability Matrix

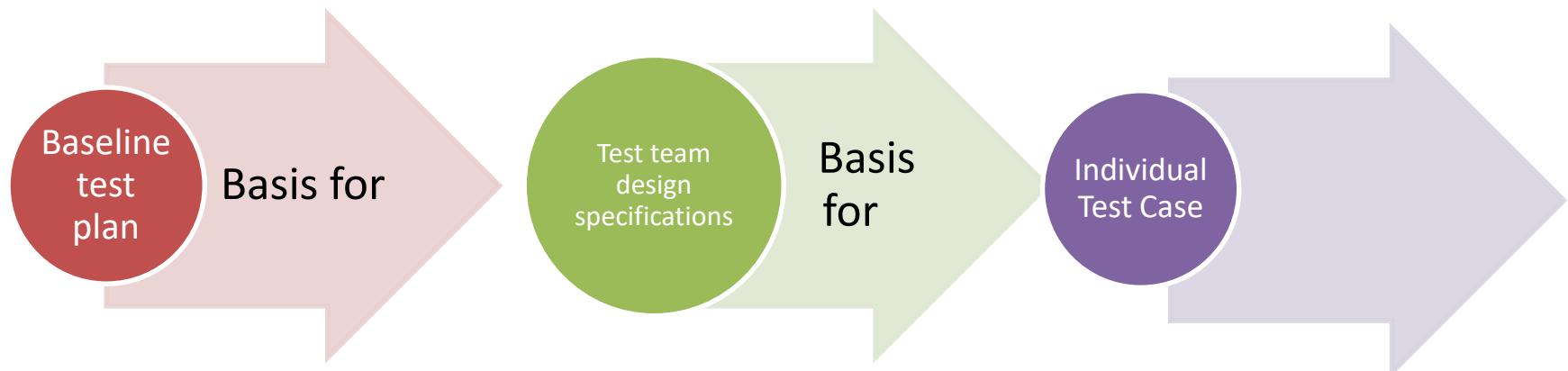
Base Lining test plan

- Every organization have template of test plan.
- The test plan is reviewed by a designated set of competent authority in organization.
- After this plan is approved by authority who is responsible for testing.

Base Lining test plan

- After this, the test plan is baseline into the configuration management repository.
- From then baseline test plan becomes the basis for running the testing project.
- Any significant changes in the testing project should thereafter be reflected in the test plan .
- Changed test plan baseline again the configuration management repository.

Test Case Specification



Test Case Specification

Test case specification should clearly identify following :

1. The purpose of the test:-This will list which feature the test is intended for
2. Items being tested along with their version release number
3. Environment that needs to be set up for running the test case.eg hardware environment set up, software environment set up
4. Input data to be used for the test case. This choice is depending on technique followed in the case. For example manual or automated or type of testing (equivalence portioning or boundary value analysis)

Test Case Specification

Test case specification should clearly identify following :

5. Steps to be followed to execute test
6. The expected result that are correct results
7. A step to compare the actual result produced with the expected result
8. Any relationship between this test and other tests:-by this we can find dependencies among the test

Requirement Traceability Matrix

Traceability matrix is tool to validate that every requirement is tested. It provides mapping between requirement and test cases.

- The traceability matrix is created during the requirement gathering phase.
- Unique identifier is assigned to each requirement.
- As project proceeds through design phase identifier for design feature is entered in matrix.

Requirement Traceability Matrix

- When project enters coding phase identifier for program file is entered in the traceability matrix.
- When test case specification is complete the row corresponding to the requirement which is being tested by the test case is updated with the test case specification identifier.

Requirement Traceability Matrix

- Testability matrix helps in:
 - Ensuring 100% test coverage
 - Showing requirement/document inconsistencies
 - Displaying the overall defect/execution status with focus on business requirements.
 - Template for RTM(Requirement Traceability matrix) :

Unique No	Require- mnt	Source of Requirement	Design Spec.	Program Module	Test Spec.	Test Case	Successful Test Verification	Modification of Requirement	Remark

Test Reporting



Executing Test Cases

Proper execution of test cases is essential which will minimize the work and reduce the time to release s/w product.

Test execution task:

1. Follow the test procedures to execute test cases.
2. Do the confirmation testing for the failed test cases.
3. Log the result for test execution.

Executing Test Cases

4. Compare actual and expected results. In case of difference defect occurrence is reported.
5. Update defect database which is used to communicate between developer and tester team.
6. So the execution of test cases will decide the suspension or resumption of further test cases.

Test Summary Report

- Test summary report is a document which contains summary of test activities and final test results.
- After the testing cycle it is very important that you communicate the test results and findings to the project stakeholders .
- so that decisions can be made for the software release. i.e. If further testing is required and we need to delay the release.

Test Summary Report

- Test summary report will be different for different kind of testing.
- In addition to test coverage and unresolved defects test summary reports should also contain test strategy, test objectives and overall result of test effort.

References

<https://www.gcreddy.com/2014/07/software-test-plan-templates.html>

<https://www.kualitee.com/test-management/best-test-management-tools-must-used-2019/>

Thank You

Supriya Kadam

Department of Computer Engineering (NBA Accredited)

Vidyalankar Polytechnic

Vidyalankar College Marg, Wadala(E), Mumbai 400 037

E-mail: supriya.angne@vpt.edu.in



CHAPTER 01

BASIC OF SOFTWARE TESTING AND TESTING METHODS

Content:

- 1.1 Software Testing, Objective of Testing.
- 1.2 Failure , Error , Fault , Defect,Bug Terminology.
- 1.3 Test Case , When to Start and Stop Testing of Software(Entry and Exit Criteria).
- 1.4 Verification and Validation (V Model) , Quality Assurance, Quality Control.
- 1.5 Methods of Testing: Static and Dynamic Testing.
- 1.6 The box approach:White Box Testing: Inspections, Walkthroughs, Technical Reviews, Functional Testing, Code Coverage Testing, Code Complexity Testing.
- 1.7 Black Box Testing: Requirement Based Testing, Boundary Value Analysis, Equivalence Partitioning.

1.1 SOFTWARE TESTING:

1.1.1 OBJECTIVE OF TESTING:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as yet undiscovered error.
- A successful test is one that uncovers an as yet undiscovered error.
- Major objectives for performing Software Testing are: to ensure that the solution meets the business and user requirements; to catch errors that can be bugs or defects; to determine user acceptability; to ensure that a system is ready for use; to gain confidence that it works; evaluating the capabilities of a system to show that a system performs as intended; and verifying documentation
- Software Testing evaluate and map the final software against business and user requirements. Well performed software testing will have good test coverage. Higher the test coverage better is evaluation of the software
- Software testing shall design test cases with higher probability of finding errors. In order to ensure these test cases shall be effective. The number of errors reported per defined number of test cases is quite indicative of this objective. Higher the number better is test objective met.
- Software testing shall ensure that user accepts the final software released for him to operate with no complaints. In order to fulfill this objective, tester shall have end-user mindset which would help in writing test cases or scenarios to meet user expectations.
- Software Testing shall ensure that the proper testing is being done to ensure that the system is ready to use. Good testing covers functionality, install ability, and operational ease to learn and use the system. Also, it verifies that the system is easily deployable and replaceable. This would make the system that is easy to install, learn, and use.
- Software Testing shall allow to gain confidence that it works. This would happen when system testing proves that the system is reliable and does not crash or there will be no show stoppers.
- Software testing shall evaluate the capabilities of a system to show that a system performs as intended. This allows us to understand the limits of performance, and also learning what a system is able to do and not able to do. For the user with good

knowledge on the system, the

- results from the system shall be predictable.
- Software testing shall also verify documentation. Many documents are created and also, evolve throughout software development life cycle. Along with this there can be on-line help, installation and troubleshooting related, user training related documents. Software testing shall evaluate all these documents for correctness.

1.2 FAILURE :

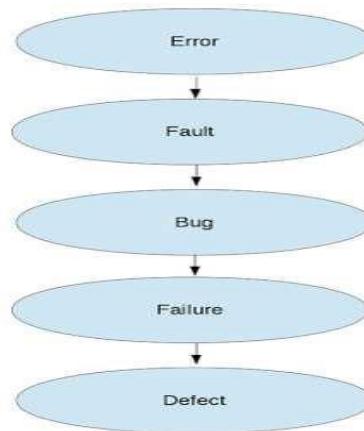
It is the inability of a system or component to perform required function according to its specification. Failure occurs when fault executes. If under certain environment and situation defects in the application or product get executed then the system will produce the wrong results causing a failure. Not all defects result in failures, some may stay inactive in the code and we may never notice them. Failures can also be caused because of the other reasons also like:

- Because of the environmental conditions as well like a radiation burst, a strong magnetic field, electronic field or pollution could cause faults in hardware or firmware. Those faults might prevent or change the execution of software.
- Failures may also arise because of human error in interacting with the software, perhaps a wrong input value being entered or an output being misinterpreted.
- Finally failures may also be caused by someone deliberately trying to cause a failure in the system.

ERROR:

The mistakes made by programmer is known as an 'Error'. Error is deviation from actual and expected value. This could happen because of the following reasons:

- Because of some confusion in understanding the functionality of the software
- Because of some miscalculation of the values
- Because of misinterpretation of any value, etc.



FAULT:

It is a condition that causes the software to fail to perform its required function. An incorrect step, process, or data definition in a program. Fault is incorrect step, process or data definition in a

computer program which causes the program to behave in an unintended or unanticipated manner. It is the result of the error

DEFECT:

- The bugs introduced by programmer inside the code are known as a defect. This can happen because of some programmatically mistakes.
- A defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results.
- A defect is said to be detected when a failure is observed.

BUG TERMINOLOGY:

Bug: A fault in a program which causes the program to perform in an unintended or unanticipated manner. It is an evidence of fault in the program.

If an error has been produced in the requirement specification phase and not detected in the same phase, then it results in a bug in the next phase, i.e. design phase. In design phase, there are also errors produced and bugs come from the previous stage, if not detected they pass on to the next stage i.e. coding phase. In this way, errors and bugs appear and travel through various stages of SDLC and it is very hard to detect and debug. The life cycle of a bug is classified into two phases:

Bugs-In Phase: In this phase the errors and bugs are introduced in the software. When we commit a mistake, it creates errors in the software and consequently, when this error goes unnoticed, it causes some conditions to fail, leading to a bug in the software. If these bugs are not detected they are spread out to the subsequent phases of SDLC. Each stage has its own errors as well as bugs received from previous phase. If we are not performing verification on earlier phase there are no chance to detect these bugs.

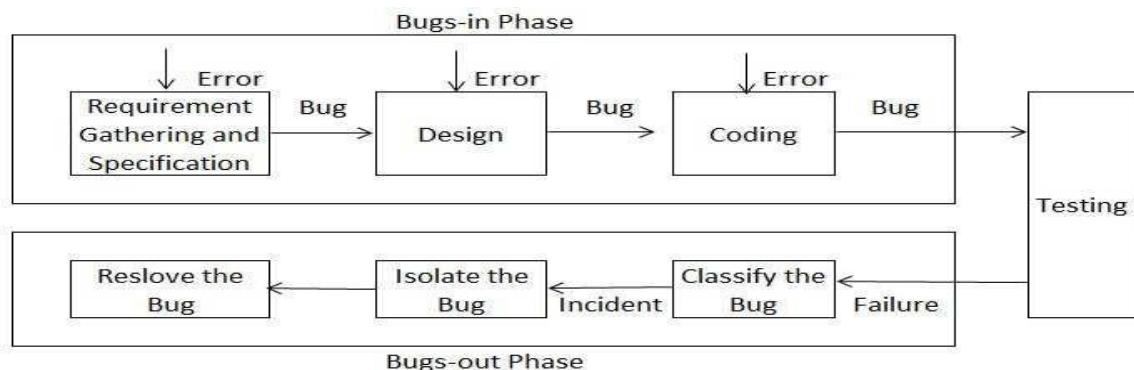


Fig. Life Cycle of a bug

Bugs-Out Phase: If we found failure while testing a software product, we concluded that it

is affected by bugs. Following activities are performed to get products bug free.

1) **Bug classification:** The bugs can be classify in several ways

- **Critical Bugs:** The defect affects critical functionality or critical data. This type of bugs has the worst effect such that it stops or hangs the normal functioning of the software. The person using the software becomes helpless when these types of bugs appear. For example, in the sorting program when we give input numbers, the system hangs and needs to be reset.
- **Major Bugs:** The defect affects major functionality or major data. This type of bug does not stop the functioning of the software but it causes a functionality to fail to meet its requirement as expected. For example in a sorting program, the output is being displayed but not the correct one.
- **Medium Bugs:** They are less critical in nature. If the outputs are not according to the standards or conventions, e.g. redundant or truncated output, then the bug is a medium bug.
- **Minor Bugs:** These types of bugs do not affect the functionality of the software. These are just mild bugs which occur without any effect on the expected behavior or continuity of the software. For example, typographical error or misaligned printout.

There are many bugs to be resolved, but tester may not have sufficient time. This categorization of bugs may help by handling high criticality bugs first and considering other trivial bugs on the list later, if time permits.

1) **Bugs Isolation:** Bug isolation is an activity where we locate the module in which the bug appears. We observe the symptoms and back-trace the design of the software and reach the module/files and the condition inside it which has caused the bug. This is known as bug isolation.

2) **Bug Resolution:** Once we isolate the bug, we back-trace the design to pinpoint the exact location of the error and a bug is resolved.

States of a Bug:

Defect life cycle is a cycle which a defect goes through during its lifetime. It starts when defect is found and ends when a defect is closed, after ensuring it's not reproduced. Defect life cycle is related to the bug found during testing. The bug has different states in the Life Cycle. The Life cycle of the bug can be shown diagrammatically as follows:

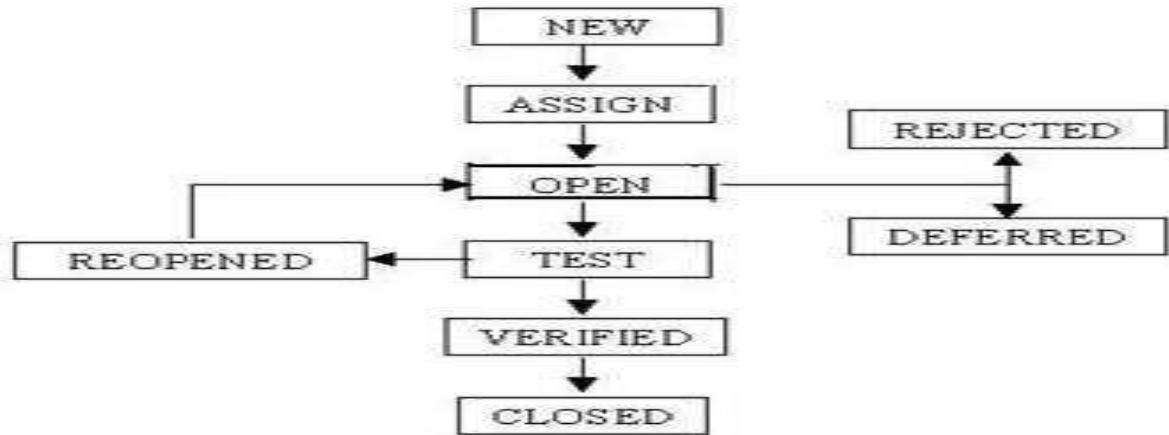


Fig: States of a Bug (Bugs life cycle)

- **New:** When a bug is found for the first time, the software tester communicates it to his/her team leader in order to confirm if that is a valid bug. After confirmation from the Test Leader, the software tester logs the bug and the status of "New" is assigned to the bug.
- **Assign:** After the bug is reported as "New", it's come to the Development Team. The Development Team verifies it validity. If the bug is valid, a developer is assigned the job to fix it and the state of the bug now is "Assign".
- **Open:** Once the developer starts working on the bug. He/she changes the state of the bug to "Open" to indicate that he/she is working on it to find a solution.
- **Deferred:** The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.
- **Rejected:** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to "Rejected".
- **Test:** After fixing the valid bug, the developer sends it back to the testing team for next round of checking. Before releasing to the testing team, the developer changes the bug's state to "Test". It specifies that the bug has been fixed by the development team but not tested and is released to the testing team.
- **Reopened:** Once the bug is fixed and the status is changed to "Test", the tester tests the bug. If the bug is reproducible in the software, he changes the status to "Reopened".
- **Verified:** Once the bug is fixed and the status is changed to "Test", the tester tests the bug. If the bug is not reproducible in the software, he changes the status to "Verified"

- **Closed:** Once the tester and other team members are confirmed that the bug is completely eliminated they change its status to "Closed".

Why Bugs do occurs?

A software bug may be defined as a coding error that causes an unexpected defect, fault, flaw, or imperfection in a computer program. In other words, if a program does not perform as intended, it is most likely a bug. There are many reasons for software bugs. Most common reason is human mistakes in software design and coding. Once you know the causes for software defects it will be easier for you to take corrective actions to minimize these defects. The reasons are:

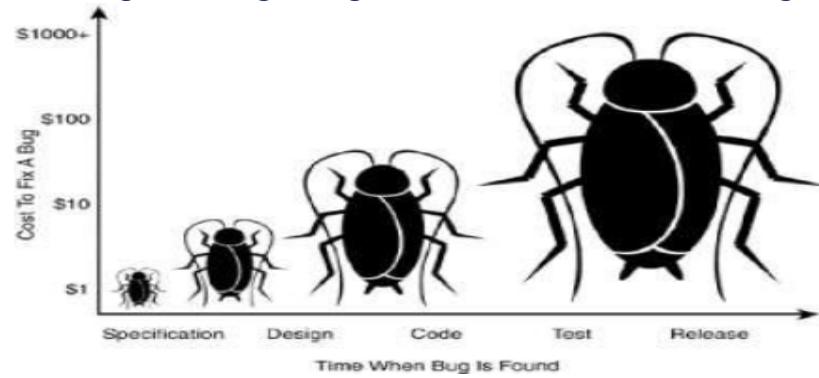
- **Miscommunication or no communication:** Success of any software application depends on communication between stakeholders, development and testing teams. Unclear requirements and misinterpretation of requirements are two major factors causing defects in software. Also defects are introduced in development stage if exact requirements are not communicated properly to development teams.
- **Software complexity:** The complexity of current software applications can be difficult to comprehend for anyone without experience in modern-day software development. Windows-type interfaces, client-server and distributed applications, data communications, enormous relational databases, and sheer size of applications have all contributed to the exponential growth in software/system complexity. And the use of object-oriented techniques can complicate instead of simplify a project unless it is well-engineered.
- **Programming errors:** Programmers, like anyone else, can make mistakes. Not all developers are domain experts. Inexperienced programmers or programmers without proper domain knowledge can introduce simple mistakes while coding. Lack of simple coding practices, unit testing, debugging are some of the common reasons most issues get introduced at development stage.
- **Changing requirements:** The customer may not understand the effects of changes, or may understand and request them anyway – redesign, rescheduling of engineers, effects on other projects, work already completed that may have to be redone or thrown out, hardware requirements that may be affected, etc. If there are many minor changes or any major changes, known and unknown dependencies among parts of the project are likely to interact and cause problems, and the complexity of keeping track of changes may result in errors. Enthusiasm of engineering staff may be affected. In some fast-changing business environments, continuously modified requirements may be a fact of life. In this case, management must understand the resulting risks, and QA and test engineers must adapt and plan for continuous extensive testing to keep the inevitable bugs from running out of control.
- **Time pressures:** Scheduling of software projects is difficult at best, often requiring a lot of guesswork. When deadlines loom and the crunch comes, mistakes will be made. Unrealistic schedules though not common but major concern in small scale projects/companies results in software bugs. If there is not enough time for proper design, coding and testing, it's quite obvious that defects will be introduced.
- **Poorly documented code:** It's tough to maintain and modify code that is badly written or poorly documented; the result is software bugs. In many organizations management provides no incentive for programmers to document their code or write clear, understandable code. In fact, it's usually the opposite: they get points mostly for quickly turning out code, and there's job security if nobody else can understand it

('if it was hard to write, it should be hard to read'). Any new programmer starting to work on this code may get confused due to complexity of the project and poorly documented code. Many times it takes longer to make small changes in poorly documented code as there is huge learning curve before making any code change.

- **Software development tools:** Visual tools, class libraries, compilers, scripting tools, etc. often introduce their own bugs or are poorly documented, resulting in added bugs. Continuously changing software tools used by software programmers. Keeping pace with the different versions and their compatibility is a major ongoing issue.
- **Obsolete automation scripts:** Writing automation scripts takes lot of time especially for complex scenarios. If automation teams record/write any test script but forget to update it over the period of time that test could become obsolete. If the automation test is not validating the results properly it won't be able to catch the defects.
- **Lack of skilled testers:** Having skilled testers with domain knowledge is extremely important for success of any project. But appointing all experienced testers is not possible for all companies. Domain knowledge and the tester's ability to find defects can produce high quality software. Compromise on any of this can result in buggy software.
- Not having proper test setup (test environment) for testing all requirements
- Starting to write code or test cases without understanding the requirements clearly.
- Incorrect design which leads to issues being carried out in all phases of software development cycle.
- Releasing software patches frequently without completing the software testing life cycle.
- Not providing training to resources for the skills needed for developing or testing the application properly.
- Giving very less or no time for regression testing.
- Not automating repetitive test cases and depending on the testers for manual verification every time.
- Not prioritizing test execution.
- Not tracking the development and test execution progress continuously. Last minute changes are likely to introduce errors.
- Wrong assumption made while coding and testing stages.

Cost of Bugs:

The costs are logarithmic; they increased tenfold as times increases. A bug found & fixed during the early stages that are requirement or product specification stage can be found & fixed by a brief interaction with the concerned & might cost next to nothing. During coding, a swiftly spotted mistake may take only very less efforts to fix. When the specification is being written might cost next to nothing or 10 cents in our examples. The same bug, if not found until software is coded & tested, might cost \$1 to \$10. If customer finds it, the cost could easily top \$100. The cost of fixing these bugs can grow over time is as shown in diagram.



1.3 TEST CASES:

Test case is a well-documented procedure designed to test the functionality of a feature in the system. A test case has an identity and is associated with a program behavior. The primary purpose of designing a test case is to find errors in the system. For designing the test case, it needs to provide a set of inputs and its corresponding expected outputs. 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.

Need of test case planning.

Test planning is important for following reasons:

- **Organization:** Proper planning will organize test cases so that all testers & other project team members can review & use them effectively.
- **Repeatability:** Same test cases run several times to look new bugs & to make sure that old ones have been fixed.
- **Tracking:** Tracking of -how many test cases did you plan to run?
 - How many did you run on last software release?
 - How many passed & how many failed?
 - Were any test cases skipped?
- **Proof of testing:** In a few high-risk industries, the software test team must prove that it did indeed run the tests that it planned to run. It could actually be illegal to release software in which a few test cases were skipped. Proper test case planning & tracking provides a means for proving what was tested.

Test Case Template:

- **Test case ID:** Is the identification number given to each test case.
- **Purpose:** Defines why the case is being designed.
- **Preconditions:** Preconditions for running the inputs in a system can be defined, if required in a test case.
- **Inputs:** Inputs should not be hypothetical. Actual inputs must be provided, instead of general inputs.
- **Expected outputs:** Are the outputs which should be produced when there is no failure. Test cases are designed based on the chosen testing techniques. They provide inputs when the system is executed. After execution, observed results are compared with expected outputs mentioned in the test case.
- **Testware:** The documents created during testing activities are known as testware. Taking the analogy from software and hardware as a product, testware are the documents that a test engineer produces. It may include test plans, test specifications, test case design, test reports etc. Testware documents should also be managed and updated like a software product.
- **Incident:** When a failure occurs it may or may not be readily apparent to the user. An incident is the symptom(s) associated with a failure that alerts the user about the occurrence of a failure.
- **Test Oracle:** An oracle is the means to judge the success or failure of a test, i.e. to judge the correctness of the system for some test. The simplest oracle is comparing actual results with expected results by hand. This can be very time consuming, so automated oracles are sought.

A test case can have the following elements. Note, however, that normally a test management tool is 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 Smith
Date of Creation	01/14/2012
Executed By	Jane Smith
Date of Execution	02/16/2012
Test Environment	OS: Windows Y Browser: Chrome N

Writing Good Test Cases:

- As far as possible, write test cases in such a way that you test only one thing at a time. Do not overlap or complicate test cases. Attempt to make your test cases 'atomic'.
- Ensure that all positive scenarios and negative scenarios are covered.
- Language:
 - Write in simple and easy to understand language.
 - Use active voice: Do this, do that.
 - Use exact and consistent names (of forms, fields, etc).

Characteristics of a good test case:

- *Accurate*: Exacts the purpose.
- *Economical*: No unnecessary steps or words.
- *Traceable*: Capable of being traced to requirements.
- *Repeatable*: Can be used to perform the test over and over.
- *Reusable*: Can be reused if necessary.

Example-1: Write sample test-cases to test “facebook” web site. Note: Testing Cases may vary from student to student

Sr. no	Check item	Test case Objective	Steps to Execute	Test data i/p	Expected Results
1	Log in Page	Leave all fields as blank as click on login button	Click log in	NA	It should display invalid login or password message on the screen.
2	Username	Enter invalid username	Type in the password and press enter,	Username: smith@yahoo.co.in	Invalid username or password should be displayed on the screen.
3	Like button	Check the like button	Click on the like button under post	NA	Unlike should appear under the post and your username should be added in the list of users who have liked the post.
4	POST WALL	Click the post button when nothing is typed in the post area	Click on Post button	NA	Nothing to message display should be on the screen.
5	Upload picture	Upload a picture format which is not supported	Add the picture or image to album or on wall	Add the format of the image which is not supported by facebook	Image not supported should be displayed by the screen.
6	Delete a post	Check the delete post activity	Click on the delete post activity	NA	Message asking whether you are sure to delete the post should appear on the screen.

7	Tag picture	the	Check the tag activity	Click on the tag button	NA	As you press tag the name of the person should b e automatically detected or it should
---	-------------	-----	------------------------	-------------------------	----	--

					give you the clue of name from your list of friends.
--	--	--	--	--	--

Example-2: Write four test cases to test the “rediff home page”.

Test case ID	Description	Input Data	Expected Result	Actual Result	Status
1	Click on rediffmail link	NA	Username and password page should get displayed	Username and password page is displayed	pass
2	Click on News link	NA	News page should get displayed	News page is displayed	pass
3	Click on shopping link	NA	Shopping page should get displayed	Shopping page is displayed	pass
4	Click on sign in link	NA	Sign in dialog box should get displayed	Sign in dialog box is displayed	pass

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 the context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, following professionals are involved in testing of a system within their respective capacities:

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

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

WHEN TO START AND STOP TESTING OF SOFTWARE(ENTRY AND EXIT CRITERIA):

When to Start Testing:

An early start to testing reduces the cost, time to rework and 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 lasts till the deployment of the software. However it also depends on the development model that is being used. For example in Water fall model formal testing is conducted in the Testing phase, but in incremental model, testing is performed at

the end of every increment/iteration and at the end the whole application is tested. Testing is done in different forms at every phase of SDLC like during Requirement gathering phase, the analysis and verification of requirements are also considered testing. Reviewing the design in the design phase with intent to improve the design is also considered as testing. Testing performed by a developer on completion of the code is also categorized as Unit type of testing.

When to Stop Testing:

Unlike when to start testing it is difficult to determine when to stop testing, as testing is a never ending process and no one can say that any software is 100% tested.

Following are the aspects which should be considered to stop the testing:

- 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.
- Test budget depleted.

1.4 VERIFICATION :

- Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way we want it to.
- Verification process describes whether the outputs are according to inputs or not, and
- Verification is a process of confirming whether the software meets its specifications.
- Verification is the process of examining a product to discover its defects.
- Verification is usually performed by static testing, or inspecting without execution on a computer.
- Verification is a "human" examination or review of the work product. Verification determining if phase completed correctly.

VALIDATION:

- Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.
- Validation process describes whether the software is accepted by the user or not.
- Validation is a process of confirming whether the software meets user requirements.
- Validation is the process of executing a product to expose its defects.
- Validation is performed by dynamic testing or testing with execution on a computer.
- Validation determining if product as whole satisfies requirements.

Difference between Verification & Validation

Verificatio n	Validatio n
Are we building the product right?	Are we building the right product?
Verification is the process of evaluating products of a development phase to find out whether they meet the specified requirements.	Validation is the process of evaluating software at the end of the development process to determine whether software meets the customer expectations and requirements.
The objective of Verification is to make sure that the product being developed is as per the requirements and design specifications.	The objective of Validation is to make sure that the product actually meets up the user's requirements, and check whether the specifications were correct in the first place.
Following activities are involved in Verification: Reviews, Meetings and Inspections and audits.	Following activities are involved in Validation: Testing like black box testing, white box testing, gray box testing etc.
Verification is carried out by QA team to check whether implementation software is as per specification document or not.	Validation is carried out by testing team.
Ensure that the software system meets all the functionality.	Ensure that functionalities meet the intended behavior.

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.
Done by developers.	Done by Testers.
Execution of code is not comes under Verification.	Execution of code is comes under Validation.
Verification process explains whether the outputs are according to inputs or not.	Validation process describes whether the software is accepted by the user or not.
Following items are evaluated during Verification: Plans, Requirement Specifications, Design Specifications, Code, Test Cases etc,	Following item is evaluated during Validation: Actual product or Software under test.
Verification talks about a process, standard and guidelines.	Validation talks about the product.
Cost of errors caught in Verification is less than errors found in Validation.	Cost of errors caught in Validation is more than errors found in Verification.
It can catch errors that validation cannot catch. It is low level exercise.	It can catch errors that verification cannot catch. It is High Level Exercise.
Verification can find about 60% of the defects.	Validation can find about 30% of the defects.
It is human based checking of documents and files.	It is computer based execution of program.
Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
It generally comes first-done before validation.	It generally follows after verification.
It is an objective process and no subjective decision should be needed to verify the Software.	It is a subjective process and involves subjective decisions on how well the Software works.

V MODEL:

The V - model is SDLC model where execution of processes happens in a sequential manner in V- shape. It is also known as Verification and Validation model.

V - Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.

Phases of V-Model:

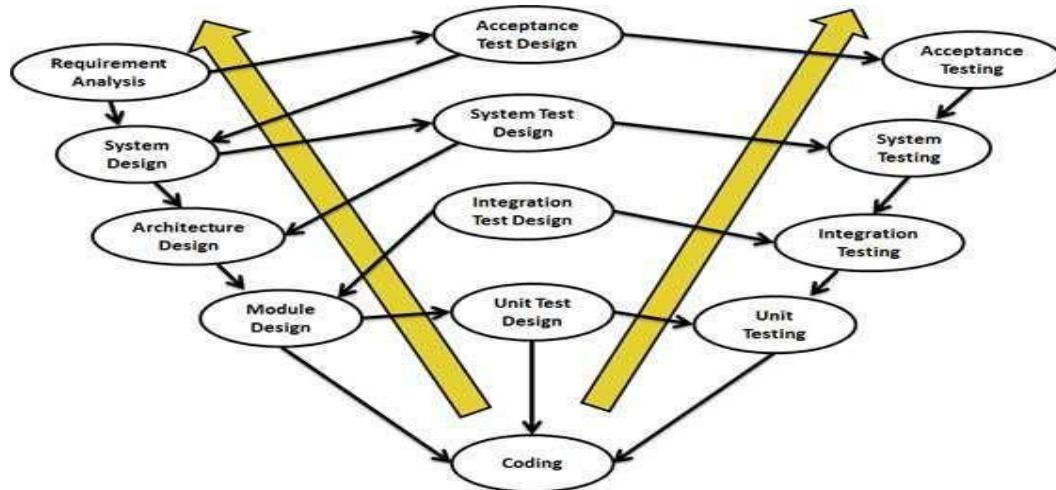


Fig: V-Model

Verification

Phases:

Following are the Verification phases in V-Model:

- **Business Requirement Analysis:** This is the first phase in the development cycle where the product requirements are understood from the customer perspective. This phase involves detailed communication with the customer to understand his expectations and exact requirement. This is a very important activity and need to be managed well, as most of the customers are not sure about what exactly they need. The acceptance test design planning is done at this stage as business requirements can be used as an input for acceptance testing.
- **System Design:** Once you have the clear and detailed product requirements, it's time to design the complete system. System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development. System test plan is developed based on the system design. Doing this at an earlier stage leaves more time for actual test execution later.
- **Architectural Design:** Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. System design is broken down further into modules taking up different functionality. This is also referred to as High Level Design (HLD). The data transfer and communication between the internal modules and with the outside world (other systems) is clearly understood and defined in this stage. With this information, integration tests can be designed and documented during this stage.
- **Module Design:** In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD). It is important that the design is compatible with the other modules in the system architecture and the other external systems. Unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. Unit tests can be designed at this stage based on the internal module designs.

Coding Phase:

The actual coding of the system modules designed in the design phase is taken up in the Coding phase. The best suitable programming language is decided based on the system and architectural requirements. The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases:

Following are the Validation phases in V-Model:

- **Unit Testing:** Unit tests designed in the module design phase are executed on the code during this validation phase. Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.
- **Integration Testing:** Integration testing is associated with the architectural design phase. Integration tests are performed to test the coexistence and communication of the internal modules within the system.
- **System Testing:** System testing is directly associated with the System design phase. System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during system test execution.
- **Acceptance Testing:** Acceptance testing is associated with the business requirement analysis phase and involves testing the product in user environment. Acceptance tests uncover the compatibility issues with the other systems available in the user environment. It also discovers the nonfunctional issues such as load and performance defects in the actual user environment.

V- Model Application:

V-Model application is almost same as waterfall model, as both the models are of sequential type. Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly disciplined domain.

Following are the suitable scenarios to use V-Model:

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements.
- The project is short.

Advantages of V-model:

- Simple and easy to understand and use.
- This is a highly disciplined model and Phases are completed one at a time.
- 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.
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- 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.

- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

- Once an application is in the testing stage, it is difficult to go back and change a functionality
- No working software is produced until late during the life cycle.

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.

QUALITY ASSURANCE:

Quality Assurance (QA)

- Quality assurance is about the work process. They measure the process, identify deficiencies and suggest improvements.
- QA is planned and systematic way to evaluate quality of process used to produce a quality product.
- The goal of a QA is to provide assurance that a product is meeting customer's quality expectations.
- QA deals with how to prevent bugs from occurring in a product being developed.
- Software Quality Assurance Engineer's main responsibility is to create and implement methods and standards to improve development process.
- QA is associated with activities like measuring the quality of process used to develop a product, process improvement and defect prevention.
- It consists of auditing and reporting procedures related to development and testing.

QUALITY CONTROL:

Quality Control (QC)

- Quality Control is about the product; they measure the product, identify deficiencies and suggest improvements.
- Quality control name comes from manufacturing industry where QC inspector evaluate sample products taken from the manufacturing line, test them & if the products fail the test they have the authority to shut down the whole production line.
- QC implements the process established by QA.
- Software tester is responsible for QC.
- If required, personnel involve in QC have to carry out QA tasks as well.

Difference between QA and QC

Quality Assurance	Quality Control
Process oriented activities.	Product oriented activities.
QA is the process of managing for quality.	QC is used to verify the quality of the output.
They measure the process, identify the deficiencies/weakness and suggest improvements.	They measure the product, identify the deficiencies/weakness and suggest improvements.
SQA is a set of activities for ensuring quality in software engineering processes (that ultimately result in quality in software products). The activities establish and evaluate the processes that produce products.	SQC is a set of activities for ensuring quality in software products. The activities focus on identifying defects in the actual products produced.
Relates to all products that will ever be created by a process	Relates to specific product
Activities of QA are Process Definition and Implementation, Audits and Training	Activities of QC are Reviews and Testing
Preventive activities.	It is a corrective process.
Quality assurance is a proactive process	Quality control is a reactive process.
It is a subset of Software Test Life Cycle (STLC).	QC can be considered as the subset of Quality Assurance.
QA is a line function (Activity contributing directly to the output of an organization)	QC is staff function. (A staff function is a secondary business activity that supports the line functions of a business.)
Verification is an example of QA	Validation/Software Testing is an example of QC
QA is a managerial tool	QC is a corrective tool

1.5 METHODS OF TESTING:STATIC AND DYNAMIC TESTING

Static Testing:

- Static Testing is a type of a Software Testing method which is performed to check the defects in software without actually executing the code of the software application.
- Static testing is performed in early stage of development to avoid errors as it is easier to find sources of failures and it can be fixed easily. The errors that can't not be found using Dynamic Testing, can be easily found by 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".
- 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

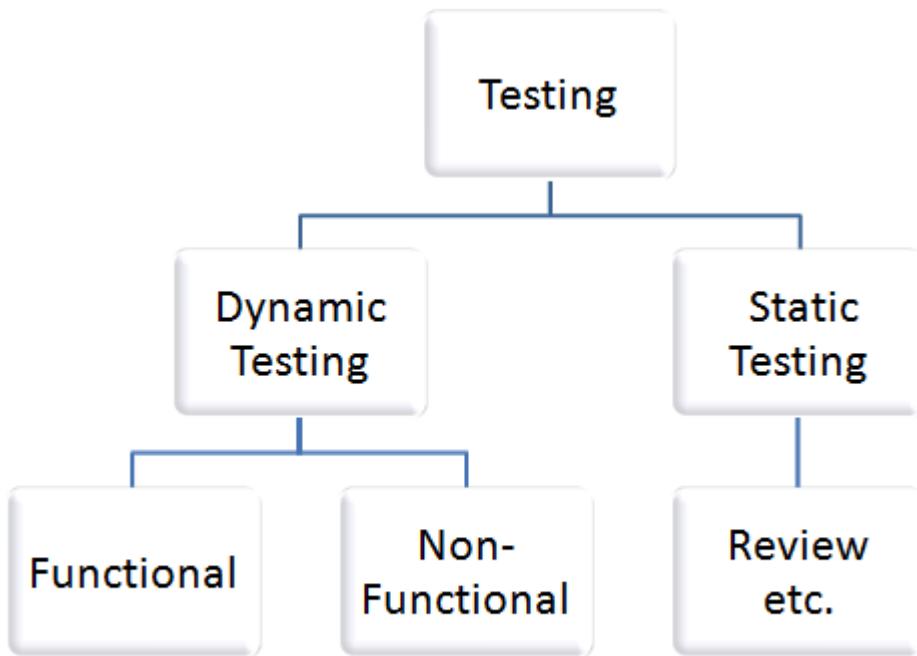
Static Testing Techniques:

- **Informal Reviews:** This is one of the type 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.
- **Technical Reviews:** 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.
- **Walkthrough:** 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
- **Inspection:** 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.
- **Static code Review:** This is a systematic review of the software source code without executing the code. It checks the syntax of the code, coding standards, code optimization, etc. This is also termed as white box testing. This review can be done at any point during development.

Dynamic Testing:

- Dynamic Testing is a type of Software Testing which is performed to analyze the dynamic behavior of the code.
- It includes the testing of the software for the input values and output values that are analyzed.
- Under **Dynamic Testing**, a code is executed. It checks for functional behavior 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.



©guru99.com

What does dynamic testing do?

The main aim of the Dynamic tests is to ensure that software works properly during and after the installation of the software ensuring a stable application without any major flaws(this statement is made because no software is error free, testing only can show presence of defects and not absence)

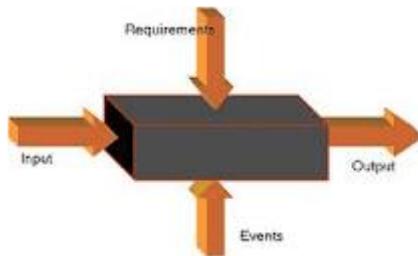
The main purpose of the dynamic test is to ensure consistency to the software; lets discuss this with an example.

In a Banking Application, we find different screens like My Accounts Section, Funds Transfer, Bill Pay, etc.. All these screens contain amount field which accepts some characters.

Let's say My Accounts field displays amount as **25,000** and Funds Transfer as **\$25,000** and Bill pay screen as **\$25000** though the amount is the same, the way amount is displayed is not the same hence making the software nonconsistent.

Consistency is not only limited to the functionality it also refers to different standards like performance, usability, compatibility etc, hence it becomes very important to perform Dynamic 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.

Also, Non-functional testing like performance, Security Testing fall under the category of dynamic testing.

Difference between Static Testing and Dynamic Testing:

STATIC TESTING	DYNAMIC TESTING
It is performed in the early stage of the software development.	It is performed at the later stage of the software development.
In static testing whole code is not executed.	In dynamic testing whole code is executed.
Static testing prevents the defects.	Dynamic testing finds and fixes the defects.

STATIC TESTING	DYNAMIC TESTING
Static testing is performed before code deployment.	Dynamic testing is performed after code deployment.
Static testing is less costly.	Dynamic testing is highly costly.
Static Testing involves checklist for testing process.	Dynamic Testing involves test cases for testing process.
It includes walkthroughs, code review, inspection etc.	It involves functional and nonfunctional testing.
It generally takes shorter time.	It usually takes longer time as it involves running several test cases.
It can discover variety of bugs.	It exposes the bugs that are explorable through execution hence discover only limited type of bugs.
Static Testing may complete 100% statement coverage in comparably less time.	While dynamic testing only achieves less than 50% statement coverage.
Example: Verification	Example: Validation

1.6 THE BOX APPROACH:

White Box Testing:

- White box testing is the detailed investigation of Program code, internal logic and structure of the code.
- White box testing is also called Glass Box Testing, Open Box Testing, Clear Box Testing, Transparent Box Testing, Code Based Testing or Structural Testing.
- In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

White box testing involves the testing of the software code for the following:

- Internal security holes
- Broken or poorly structured paths in the coding processes
- The flow of specific inputs through the code
- Expected output
- The functionality of conditional loops
- Testing of each statement, object and function on an individual basis

Function coverage :

This is a new addition to structural testing to identify how many program functions (similar to functions in "C" language) are covered by test cases.

The requirements of a product are mapped into functions during the design phase and each of the functions form a logical unit. For example in a database software, "inserting a row into the database" could be a function. Or, in a payroll application, "calculate tax" could be a function. Each function could in turn be implemented using other function.

While providing function coverage, test cases can be written so as to exercise each of the different functions in the code.

Advantages of functional coverage:

- Functions are easier to identify in a program and hence it is easier to write test cases to provide function coverage. Since functions are at a much higher level of abstraction than code, it is easier to achieve 100 percent function coverage than 100 percent coverage in any of the earlier methods.

- Functions have a more logical mapping to requirements and hence can provide a more direct correlation to the test coverage of the product. Functions provide one means to achieve traceability.
- Since functions are a means of realizing requirements, the importance of functions can be prioritized based on the importance of the requirements they realize. Thus, it would be easier to prioritize the functions for testing. This is not necessarily the case with the earlier methods of coverage.

Function coverage provides a natural transition to black box testing.

We can also measure how many times a given function is called. This will indicate which functions are used most often and hence these functions - become the target of any performance testing and optimization.

As an example, if in networking software, we find that the function that assembles and disassembles the data packets is being used most often, it is appropriate to spend extra effort in improving the quality and performance of that function. Thus, function coverage can help in improving the performance as well as quality of the product.

Code Coverage Testing:

- Code coverage is dynamic white box testing because it requires you to have full access to the code to view what parts of the software you pass through when you run your test cases. The simplest form of code coverage testing is using your compiler's debugger to view the lines of code you visit as you single step through the program.
- Code coverage testing involves designing and executing test cases and finding out the percentage of code that is covered by testing. The percentage of code covered by a test is found by adopting a technique called instrumentation of code.

• Following are the types of coverage's:

1) **Statement Coverage:** The statement coverage is also known as line coverage or segment coverage. The statement coverage covers only the true conditions. Through statement coverage we can identify the statements executed and where the code is not executed because of blockage. In this process each and every line of code needs to be checked and executed.

The statement coverage can be calculated:

$$\text{Statement Coverage: } \frac{\text{Number of statement exercised}}{\text{Total number of statements}} * 100$$

Statement coverage criteria call for having adequate number of test cases for the program to ensure execution of every statement at least once. In spite of achieving 100% statement coverage, there is every likelihood of having many undetected bugs.

Example-1:

```
READ X
READ Y
IF X>Y
PRINT "X is greater than Y"
ENDIF
```

We can have 100% coverage by just one test set in which variable X is always greater than variable Y. TEST SET 1: X=10, Y=5

Example-2

```
1 READ X  
2 READ Y  
3 Z =X + 2*Y  
4 IF Z> 50 THEN 5  
PRINT large Z  
6 ENDIF
```

TEST SET 1

Test 1_1: X= 2, Y = 3

Test 1_2: X =0, Y = 25

Test 1_3: X =47, Y = 1

- In Test 1_1, the value of Z will be 12, so we will cover the statements on lines 1 to 4 and line 6.
- In Test 1_2, the value of Z will be 50, so we will cover exactly the same statements as Test 1_1.
- In Test 1_3, the value of Z will be 49, so again we will cover the same statements.
- Test 1_4: X = 20, Y = 25 This time the value of Z is 70, so we will print 'Large Z' and we will have exercised all six of the statements, so now statement coverage = 100%.

Note that Test 1_4 on its own is more effective which helps in achieving 100% statement coverage, than the first three tests together.

Advantage of statement coverage:

- It verifies what the written code is expected to do and not to do
- It measures the quality of code written
- It checks the flow of different paths in the program and it also ensure that whether those path are tested or not.

Disadvantage of statement coverage:

- It cannot test the false conditions.
- It does not report that whether the loop reaches its termination condition.
- It does not understand the logical operators.

Code Complexity Testing

BASIS PATH TESTING

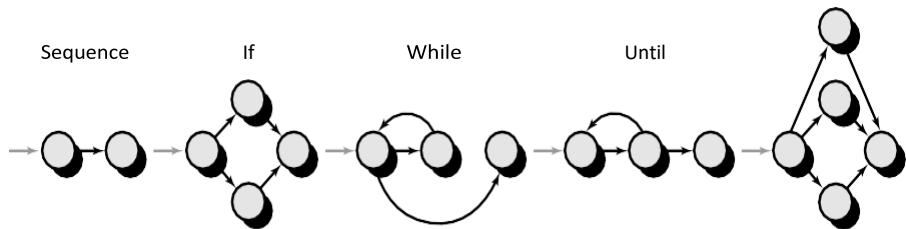
Basis path testing is a white-box testing technique first proposed by Tom McCabe.

Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

1.Flow Graph Notation

The flow graph depicts logical control flow using the notation. Each structured construct has a corresponding flow graph symbol. Here, a flowchart is used to depict program control structure.³⁷

The structured constructs in flow graph form:



Where each circle represents one or more non branching source code statements. each circle, called **a flow graph node**, represents one or more procedural statements. A sequence of process boxes and a decision diamond can map into a single node. The arrows on the flow graph, called **edges** or links, represent flow of control and are analogous to flowchart arrows. Areas bounded by edges and nodes are called **regions**. Each node that contains a condition is called a **predicate node** and is characterized by two or more edges emanating from it.

Fig A: Flowchart

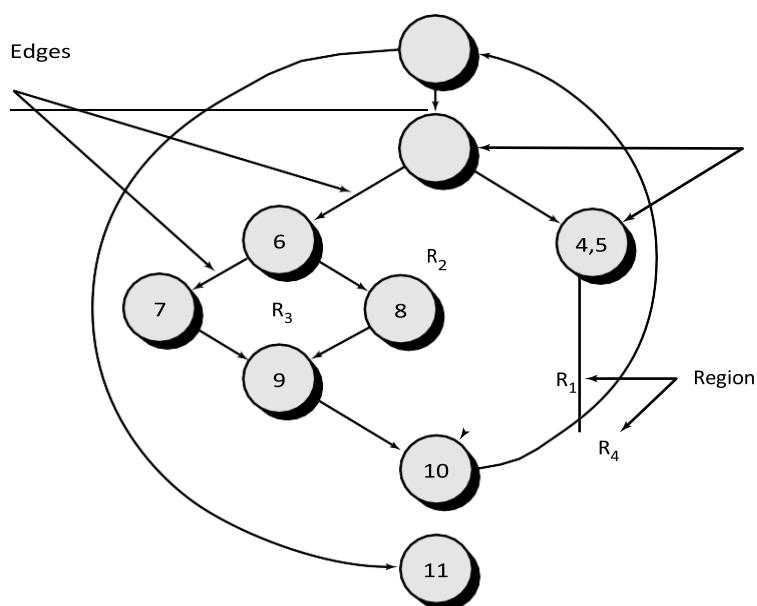
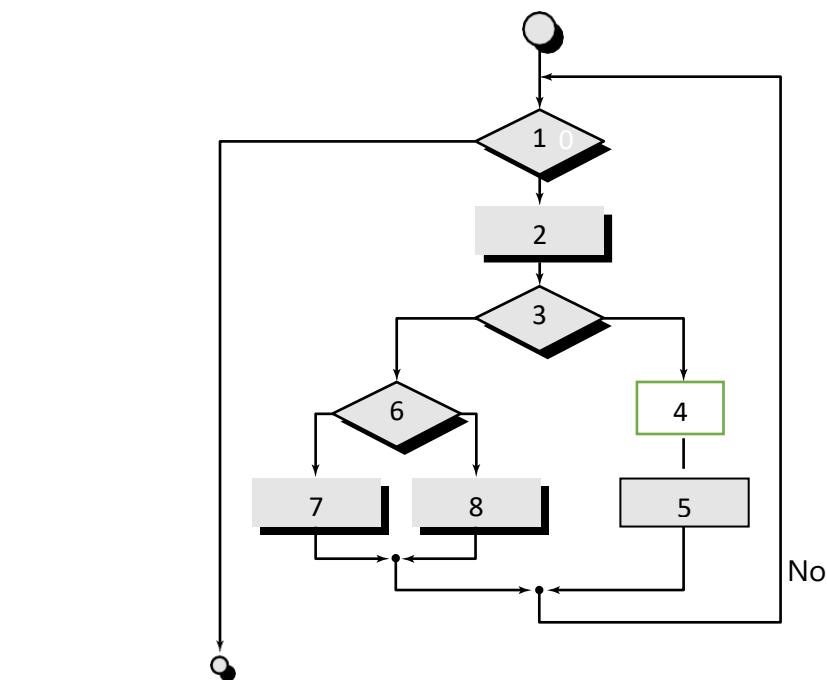


Fig B: Flow graph

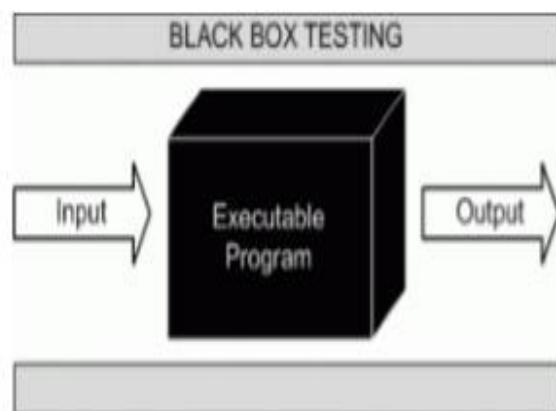
1.7 BLACK BOX TESTING:

Differentiate Between White box and Black box testing

White box testing	Black Box Testing
The tester needs to have the knowledge of internal code or program.	This technique is used to test the software without the knowledge of internal code or program
It aims at testing the structure of the item being tested.	It aims at testing the functionality of the software
It is also called structural testing, clear box testing, code-based testing, or glass box testing.	It also knowns as datadriven, closed box testing, data-, and functional testing.
Testing is best suited for a lower level of testing like Unit Testing, Integration testing.	This type of testing is ideal for higher levels of testing like System Testing, Acceptance testing.
Statement Coverage, Branch coverage, and Path coverage are White Box testing technique.	Equivalence partitioning, Boundary value analysis are Black Box testing technique
Can be based on detailed design documents.	Can be based on Requirement specification document.

Black Box Testing:

Testing software without any knowledge of the back-end of the system, structure or language of the module being tested. Black box test cases are written from a definitive source document, such as a specification or requirements document.



- 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 is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see.
- The main purpose of the Black Box is to check whether the software is working as per expected in requirement document & whether it is meeting the user expectations or not.

Steps followed to carry out any type of Black Box Testing:

- Initially requirements and specifications of the system are examined.⁴⁰
- Tester chooses valid inputs (positive test scenario) to check whether SUT (software under test) processes them correctly. Also some invalid

inputs (negative test scenario) are chosen to verify that the SUT is able to detect them.

- Tester determines expected outputs for all those inputs.
- Software tester constructs test cases with the selected inputs.
- The test cases are executed.
- Software tester compares the actual outputs with the expected outputs.
- Defects if any are fixed and re-tested.

Tools used for Black Box Testing: Tools used for Black box testing largely depends on the type of black box testing you are doing.

- For Functional/ Regression Tests you can use – QTP (Quick Test Professional)
- For Non-Functional Tests you can use - Load runner

Black Box Testing Advantages:

- More effective on larger units of code than glass 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.
- Tester and programmer are independent of each other.
- Test cases can be designed as soon as the specifications are ready.

Black Box Testing Disadvantages:

- Only a small number of possible inputs can be tested and many program paths will be left untested.
- Tests can be redundant if the software designer/ developer has already run a test case.
- It is difficult to identify all possible inputs in limited time.
- May leave many program paths untested.

Techniques for Black Box Testing: The various techniques for black box testing are as follows:

Requirement Based Testing:

Requirements-based testing is a testing approach in which test cases, conditions and data are derived from requirements.

In this testing both implicit and explicit requirements are required.

Explicit requirements are stated and documented as a part of the requirements specification.

Implicit requirements are those are not documented but assumed to be incorporated in the system.

In some organizations all explicit requirements and implicit requirements are collected and documented as "Test Requirements Specification (TRS)".

Requirements based testing can also be conducted based on such a TRS, as it captures the tester's perspective as well.

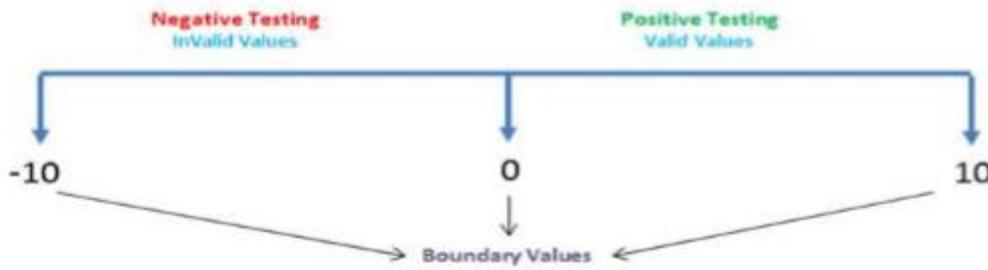
Project Stages in Requirements based Testing:

- **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.

Boundary Value Analysis (BVA)

- Most of the defects in software products hover around conditions and boundaries.
- 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.
- Same examples of Boundary value analysis concept are:
 - One test case for exact boundary values of input domains each means 1 and 100.
 - One test case for just below boundary value of input domains each means 0 and 99.
 - One test case for just above boundary values of input domains each means 2 and 101.
- For Example; A system can accept the numbers from 0 to 10 numeric values.



All other numbers are invalid values. Under this technique, boundary values 0 , 10 and -10 will be tested.

Another Example is in exam has a pass boundary at 40 percent, merit at 75 percent and distinction at 85 percent.

The Valid Boundary values for this scenario will be as follows: 49, 50 - for pass

74, 75 - for merit

84, 85 - for distinction Boundary values are validated against both the valid boundaries and invalid boundaries.

The Invalid Boundary Cases for the above example can be given as follows:

0 - for lower limit boundary value 101 - for upper limit boundary value

- Boundary value analysis is a black box testing 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.

Advantages of Boundary Value Analysis:

- Very good at exposing potential user interface/user input problems.
- Very clear guide lines on determining test cases.
- Very small set of test cases generated.

44

Disadvantages of Boundary Value Analysis:

- Does not test all possible inputs.
- Does not test dependencies between combinations of inputs.

Equivalence Partitioning

- Equivalence partitioning (EP) is a specification-based or black-box technique. It can be applied at any level of testing and is often a good technique to use first.
 - The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning'. Equivalence partitions are also known as equivalence classes
 - 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 an equivalence class or a partition.
 - In equivalence-partitioning technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.
 - This is a software testing technique which divides the input date into many partitions. Values from each partition must be tested at least once.⁴⁵ Partitions with valid values are used for Positive Testing. While, partitions with invalid values are used for negative testing.

- Equivalence partitioning is a software testing technique that involves identifying a small set of representative input values that produce as many different output conditions as possible. This reduces the number of permutations and combinations of input, output values used for testing, thereby increasing the coverage and reducing the effort involved in testing.
- Equivalence partitioning technique uncovers classes of errors. Testing uncovers sets of inputs that causes errors or failures, not just individual inputs.
- For example let us consider the bank given the interest rate for loan based on the age group

as follows: Age group Interest Rate Under 35 is 8%

35 to 59 9% 60 and

Above 60 11%

- Based on Equivalence partitioning 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 60 years of age (Valid input)

4. Negative Age (Invalid input)

5. Age is zero (Invalid input)

6. Age as any three digit number (Valid input)

- There are some valid and invalid input values. The invalid values are also required because they do not cause unforeseen errors. The test cases for the above example based on equivalence partitions are given below:

Sr. No	. Equivalence partitions	Type of input	Test data	Expected output
1	Age below 35	Valid	20 ,30	Interest Rate= 8%
2	Age 35 to 59	Valid	40, 50	Interest Rate= 9%
3	Age above 60	Valid	65, 72	Interest Rate= 11%

4	Negative Age	Invalid	-10	Warning message Invalid input
5	Age is zero	Invalid	0	Warning message- Invalid input

Steps to prepare an equivalence partitions table are as follows:

1. Choose criteria for doing the equivalence partitioning (range, list of values and so on.)
2. Identify the valid equivalence classes based on the above criteria (number of ranges allowed values, and so on.)
3. Select a sample data from that partition.
4. Write the expected result based on the requirements given.
5. Identify special values, if any and include them in the table.
6. Check to have expected results for all the cases prepared.
7. If the expected result is not clear for any particular test case, mark appropriately and escalate for corrective actions.

Advantages of Equivalence Partitions:

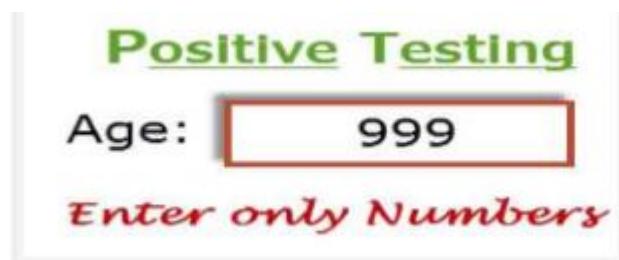
- Equivalence partitions are designed so that every possible input belongs to one and only one equivalence partition.

Disadvantages of Equivalence Partitions:

- Doesn't test every input
- No guide lines for choosing inputs.

Positive and Negative Testing

- Positive testing can be performed on the system by providing the valid data as input.
- When a test case verifies the requirements of the product with a set of expected output, it is called positive test case.
- In this testing tester always check for only valid set of values and check if a application behaves as expected with its expected inputs.
- The main intention of this testing is to check whether software application not showing error when not supposed to & showing error when supposed to.
- Such testing is to be carried out keeping positive point of view & only execute the positive scenario.
- Positive Testing always tries to prove that a given product and project always meets the requirements and specifications.
 - Under Positive testing is test the normal day to day life scenarios and checks the expected behavior of application.



- Consider a scenario where you want to test an application which contains a simple textbox to enter age and requirements say that it should take only integers values. So here provide only positive integer values to check whether it is working as expected or not is the Positive Testing.

Entering values up to 999 will be acceptable by the system and any other values apart from this should not be acceptable.

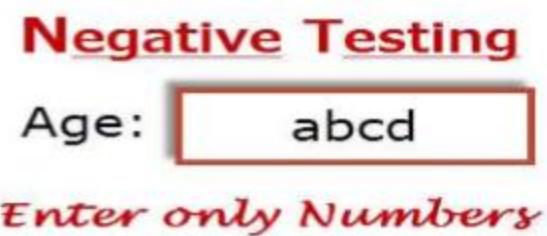
To do positive testing, set the valid input values from 0 to 999 and check whether the system is accepting the values. Most of the applications developers are implement Positive scenarios where testers get less defects count around positive testing.

Negative Testing can be performed on the system by providing invalid data as input.

- It checks whether an application behaves as expected with the negative input.
- This is to test the application that does not do anything that it is not supposed to do so.
- The main intention of this testing is to check whether software application not showing error when supposed to & showing error when not supposed to.
- Such testing is to be carried out keeping negative point of view & only execute the test cases for only invalid set of input data.
- The purpose of negative testing is to try and break the system.
- Negative testing covers scenarios for which the product is not designed and coded. i.e. the input values may not have been represented in the specification of the product.

These test conditions can be termed as unknown conditions for the product as far as the specifications are concerned.

- A negative test would be a product not delivering an error when it should or delivering an error when it should not.
- The Negative testing helps to improve the testing coverage of your software application under test.



- Consider a same above age textbox example which should accept only integer's values.

So here provide the characters like "abcd" in the age textbox & check the behavior of application, either it should show a validation error message ⁴⁹ for all invalid inputs (for all other than integer values) or system should not allow to enter a non-integer values.

- Both positive and negative testing approaches are equally important for making your application more reliable and stable.
 - The difference between positive testing and negative testing is in their coverage. For positive testing if all documented requirements and test conditions are covered, then coverage can be considered to be 100 percent.

If the specifications are very clear, then coverage can be achieved. In contrast there is no end to negative testing, and 100 percent coverage in negative testing is impractical. Negative testing requires a high degree of creativity among the testers to cover as many "unknown" as possible to avoid failure at a customer site

- . • Let's take another example of Positive and negative testing scenarios: If the requirement is saying that password text field should accept 6 – 20 characters and only alphanumeric characters.

Positive Test Scenarios:

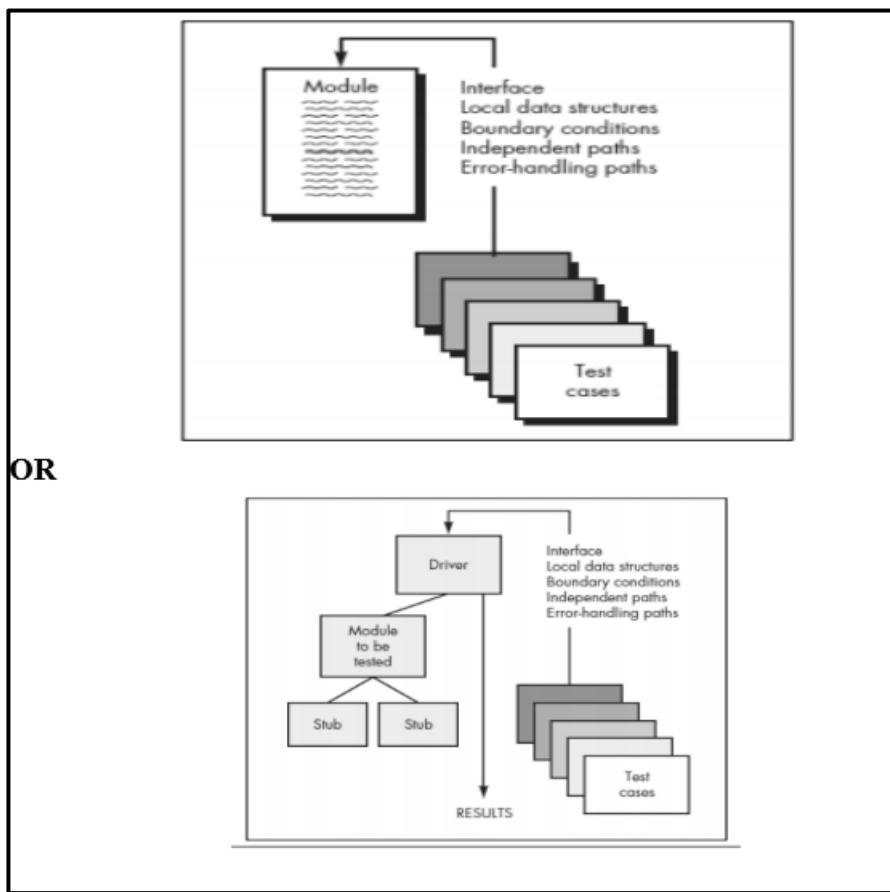
- Password textbox should accept 6 characters
- Password textbox should up to 20 characters
- Password textbox should accept any value in between 6-20 chars length.
- Password textbox should accept all numeric & alphabets values.

Negative Test scenarios:

- Password textbox should not accept less than 6 characters
- Password textbox should not exceed more than 20 characters
- Password textbox should not accept special characters

2.Types and level of testing

- Explain Unit testing in detail.



Unit testing focuses verification effort on the smallest unit of software design—the software component or module.

Using the component-level design description as a guide, important control paths are tested to uncover errors within the boundary of the module.

The relative complexity of tests and the errors those tests uncover is limited by the constrained scope established for unit testing.

51

The unit test focuses on the internal processing logic and data structures within the boundaries of a component. This type of testing can be conducted in parallel for multiple components.

Unit tests are illustrated schematically in above Figure

- The module interface** is tested to ensure that information properly flows into and out of the program unit under test.
- Local data structures** are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
- All independent paths** through the control structure are exercised to ensure that all statements in a module have been executed at least once.
- Boundary conditions** are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
- And finally, **all error-handling paths** are tested.

The unit test environment is illustrated in above Figure.

A **driver** is nothing more than a “main program” that accepts test case data, passes such data to the component (to be tested), and prints relevant results.

Stubs serve to replace modules that are subordinate (invoked by) the component to be tested. A stub or “dummy subprogram” uses the subordinate module’s interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing.

Client Server Testing

Client server testing gives an opportunity for number of users to work with software at time. In simple terms request are coming from number of clients for doing some actions and server is serving these request.

Testing approaches of client server system:

Component Testing

For testing Client and server individually approach and test plan need to be defined. One may have to devise simulators to replace corresponding components while testing the component targeted by the test. When server is tested, we may need a client simulator, while testing of client may need server simulator.

Integration Testing

After successful testing of servers, clients and network, they are brought together to form the system and system test cases are executed. Communication between client and server is tested in integration testing.

Performance Testing

System performance is tested when number of clients are communicating with server at a time. we can test the system under maximum load as well as normal load expected.

Concurrency Testing

It may be possible that multiple users may be accessing same record at a time and concurrency testing is required to understand the behaviour of a system under such circumstances.

Disaster Recovery Testing

When the client and server are communicating with each other, there exists a possibility of breaking of the communication due to various reasons or failure of either client or server or link connecting them. It may involve testing the scenario of such failure at different points in the system and action taken by the system in each case.

Testing for extended periods

In client server application it may be expected that server is running 24*7 for extended period.one need to conduct testing over an extended period to understand if service level of network and server deteriorates over time due to some reasons.

Compatibility Testing

Servers may be in different hardware, software or operating environment than the recommended one. Client may differ significantly from the expected environmental variables.

Testing must ensure that performance must be maintained on the range of hardware and software configurations.

Web Application Testing

Web application is further improvement in client server applications where the clients can communicate with server through virtual connectivity.

Testing approaches for web application:

Component Testing

For testing web application approach and test plan need to be defined individually at client side and server side. One may have to devise simulators to replace corresponding components. When server is tested, we may need a client simulator, while testing of client may need server simulator.

Integration Testing

After successful testing of servers, clients and network, they are brought together to form the web system and system test cases are executed. Communication between client and server is tested in integration testing.

53

There are regular testing like functionality testing and user interface testing to ensure that system meets the requirement specification and design specification correctly.

Performance Testing

System performance is tested when number of clients are communicating with server at a time. we can test the system under maximum load as well as normal load expected. Simulators are used extensively for such testing.

Concurrency Testing

It may be possible that multiple users may be accessing same record at a time and concurrency testing is required to understand the behaviour of a system under such circumstances.

Disaster Recovery Testing

When the machine and web server are communicating with each other, there exists a possibility of breaking of the communication due to various reasons or failure of either client or server or link connecting them. It may involve testing the scenario of such failure at different points in the system and action taken by the system in each case. MTTR (mean time to repair) and MTBF(mean time between failure)are very important test for web applications.

Testing for extended periods

In case of web application it may be expected that server is running 24*7 for extended period.one need to conduct testing over an extended period to understand if service level of network and server deteriorates over time due to some reasons.

Security testing

As the communication is through virtual network, security becomes an important issue. Application may use communication protocols, coding and decoding mechanism and schemes to maintain security of system. System must be tested for possible weak areas called vulnerabilities and possible intruders trying to attack the system.

Compatibility Testing

Servers may be in different hardware, software or operating environment than the recommended one. Client browser may differ significantly from the expected environmental variables.

Testing must ensure that performance must be maintained on the range of hardware and software configurations.

Describe following testing with example.

1)Security testing (2) Performance testing

(For each testing – 2 Marks)

Ans:

Security testing

Security Testing: Testers must use a risk-based approach, By identifying risks and potential loss associated with those risks in the system and creating tests driven by those risks, the testers can properly focus on areas of code in which an attack is likely to succeed. Therefore risk analysis at the design level can help to identify potential security problems and their impacts. Once identified ranked, software risks can help guide software security

- It is a type of non-functional testing.

- ♣ Security testing is basically a type of software testing that's done to check whether the application or the product is secured or not. It checks to see if the application is vulnerable to attacks, if anyone can hack the system or login to the application without any authorization.
- ♣ It is a process to determine that an information system protects data and maintains functionality as intended.
- ♣ The security testing is performed to check whether there is any information leakage in the sense by encrypting the application or using wide range of software's and hardware's and firewall etc.
- ♣ Software security is about making software behave in the presence of a malicious attack.
- ♣ The six basic security concepts that need to be covered by security testing are: confidentiality, integrity, authentication, availability, authorization and non-repudiation.

2) Performance Testing

Performance testing: Performance testing is intended to find whether the system meets its performance requirements under normal load or abnormal level of activities. Normal load must be defined by the requirement statement defined by the customer and system design implements them. Performance criteria must be expressed in numerical terms. Design verification can help in determining whether required measures have been taken to meet performance requirements or not. This is one area where verification does not work to that much extent and one needs to test it by actually performing the operation on the system.

- ♣ Performance testing is testing that is performed, to determine how fast some aspect of a system performs under a particular workload.
- ♣ It can serve different purposes like it can demonstrate that the system meets performance criteria.

♣ Stress Testing:

- ♣ It is a type of non-functional testing.
- ♣ It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results.
- ♣ It is a form of software testing that is used to determine the stability of a given system.
- ♣ It puts greater emphasis on robustness, availability, and error handling under a heavy load, rather than on what would be considered correct behaviour under normal circumstances.
- ♣ The goals of such tests may be to ensure the software does not crash in conditions of insufficient computational resources (such as memory or disk space).

♣ Load Testing:

- ♣ Load testing is a type of non-functional testing
- ♣ A load test is a type of software testing which is conducted to understand the behavior of the application under a specific expected load.
- ♣ Load testing is performed to determine a system's behavior under both normal and at peak conditions.
- ♣ It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation. E.g. If the number of users are increased then how much CPU, memory will be consumed, what is the network and bandwidth response time.

- Load testing can be done under controlled lab conditions to compare the capabilities of different systems or to accurately measure the capabilities of a single system.

- Examples of load testing include:

- Downloading a series of large files from the internet.

- Running multiple applications on a computer or server simultaneously.

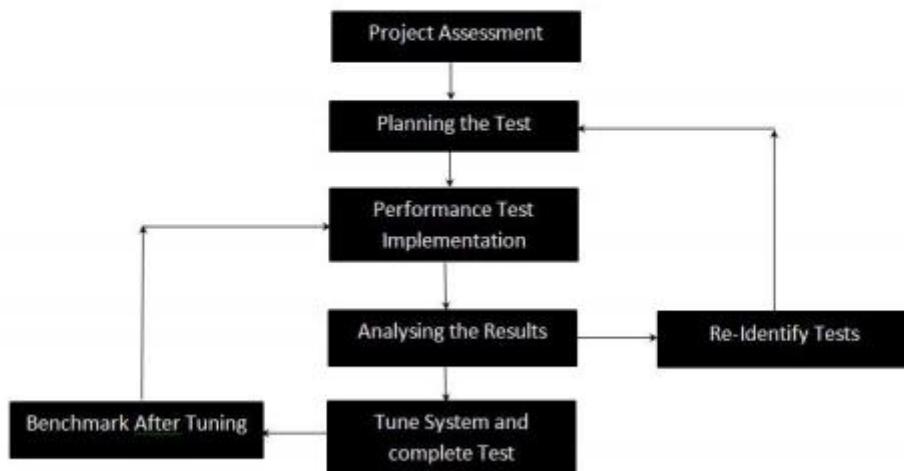
How performance testing is performed? List steps involved in it?

(Performance Testing - 2Marks, steps involved - 2 Marks)

Ans: i. Performance testing, a non-functional testing technique performed to determine the system parameters in terms of responsiveness and stability under various workload.

ii. Performance testing measures the quality attributes of the system, such as scalability, reliability and resource usage.

Performance Testing Process:



Techniques are:

Load testing - It is the simplest form of testing conducted to understand the behaviour of the system under a specific load. Load testing will result in measuring important business critical transactions and load on the database, application server, etc., are also monitored.

Stress testing - It is performed to find the upper limit capacity of the system and also to determine how the system performs if the current load goes well above the expected maximum.

Soak testing - Soak Testing also known as endurance testing, is performed to determine the system's performance under continuous expected load. During soak tests the parameters such as memory utilization is monitored to detect memory leaks or other performance issues. The main aim is to discover the system's performance under sustained use.

Spike testing - Spike testing is performed by increasing the number of users suddenly by a very large amount and measuring the performance of the system. The main aim is to determine whether the system will be able to sustain the workload.

What is load testing and stress testing? Describe with respect to system testing.

Stress testing is testing the software under less than ideal conditions. So subject your software to low memory, low disk space, slow CPU, and slow modems and so on.

Look at your software and determine what external resources and dependencies it has. Stress testing is simply limiting them to bare minimum. With stress testing you starve the software.

For e.g. Word processor software running on your computer with all available memory and disk space, it works fine. But if the system runs low on resources you had a greater potential to expect a bug. Setting the values to zero or near zero will make the software execute different path as it attempt to handle the tight constraint.

Ideally the software would run without crashing or losing data.

Load testing is testing the software under customer expected load. In order to perform load testing on the software you feed it all that it can handle.

Operate the software with largest possible data files. If the software operates on peripherals such as printer, or communication ports, connect as many as you can.

If you are testing an internet server that can handle thousands of simultaneous connections, do it. With most software it is important for it to run over long periods. Some software's should be able to run forever without being restarted.

So Time acts as a important variable.

Stress testing and load testing can be best applied with the help of automation tools. Stress testing and load testing are the types of performance testing.

The Microsoft stress utility program allows you to individually set the amounts of memory, disk space, files and other resources available to the software running on the machine.

Example: Open many number of browsers in the windows simultaneously.

Connect more than the specifies clients to the server.

Connect more than one printer to the system.

Describe use of load testing and stress testing to online result display facility of MSBTE website.

(Use of load testing with example :3 marks, use of stress testing of MSBTE online result display: 3 marks)

Stress Testing: In stress testing of MSBTE online result display, the resources used will be less than the requirement. For e.g. Provide less RAM for the server, or decrease the bandwidth of the internet connection, or provide less hits for page.

If the system has limited resources available, the response of the online result system may deteriorate due to non-availability of the resources. 57

It tries to break the page, site or connection under test by overwhelming its resources in order to find the circumstances under which it will crash. It is also a type of load testing.

It is designed to determine the behavior of the software under abnormal situations.

In stress testing test cases are designed to execute the system in such a way that abnormal conditions.

Load Testing: When a MSBTE online result display facility is tested with a load that causes it to allocate its resources in maximum amounts.

The idea is to create an environment more demanding than the application would experience under normal workloads.

Eg. Apply more number of hits on the result section, try displaying multiple results in multiple browsers, etc. Load is varied from minimum to the maximum level the system can sustain without running out of resources.

Load is being increased transactions may suffer excessive delays

. Load testing involves simulating real-life user load for the target application.

It helps to determine how application behaves when multiple students hits it simultaneously to check the results.

Load testing can be done under controlled lab conditions to compare the capabilities of different systems or to accurately measure the capabilities of a single system

Alpha and Beta Testing

1. Alpha Testing

(Question: Explain alpha testing – 4 Marks)

- i. Alpha testing is one of the most common software testing strategy used in software development. It's specially used by product development organizations.
- ii. This **test takes place at the developer's site**. Developers observe the users and note problems.
- iii. Alpha testing is testing of an application when development is about to complete. Minor design changes can still be made as a result of alpha testing.
- iv. Alpha testing is typically performed by a group that is independent of the design team, but still within the company, e.g. in-house software test engineers, or software QA engineers.
- v. Alpha testing is final testing before the software is released to the general public. It has two phases:
 - a. In the **first phase** of alpha testing, the software is tested by in-house developers. They use either debugger software, or hardware-assisted debuggers. The goal is to catch bugs quickly.
 - b. In the **second phase** of alpha testing, the software is handed over to the software QA staff, for additional testing in an environment that is similar to the intended use.

- vi. Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site.
- vii. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.

2. Beta testing

(Question: Explain beta testing. – 4 Marks)

- i. It is also known as field testing.
- ii. It takes place at **customer's site**.
- iii. It sends the system to users who install it and use it under real-world working conditions.

Software Engineering

- iv. A beta test is the second phase of software testing in which a sampling of the intended audience tries the product out.
- v. Beta testing can be considered “pre-release testing.”
- vi. The goal of beta testing is to place your application in the hands of real users outside of your own engineering team to discover any flaws or issues from the user’s perspective that you would not want to have in your final, released version of the application.
- vii. Closed beta versions are released to a select group of individuals for a user test and are invitation only, while
- viii. Open betas are from a larger group to the general public and anyone interested.
- ix. The testers report any bugs that they find, and sometimes suggest additional features they think should be available in the final version.

3.Test Management

In this chapter, we will look at the project management aspect testing. Like SDLC (software development life cycle) testing also needs planning.

Since software project becomes uncontrolled if not planned properly, the testing process is also not efficient if not planned earlier.

If testing is not effective in software project, it also affects the final software product. Therefore, for quality software, testing activities must be planned as soon as the project planning starts.

3.1 Test planning

a) Preparing test plan

- > Testing of any project is driven by a plan.
- > Test plan acts as the baseline for the execution, packing and periodic reporting of the entire testing project.

Test plan covers

1 what needs to be tested including identification of what will be tested and what will not be tested.

- 2 How testing is going to be performed
 - Divide testing task into small and manageable task.

Identifying strategies to be used for carrying out the task.

- 3 What resources are needed for testing
 - Software, Hardware and human resources.

- 4 The time lines by which testing activities will be performed.

- 5 Risk analysis should be done, with appropriate mitigation and contingency plans.

b) Scope management – Deciding features to be scope management defines scope of project. For testing scope management includes:

1. Understanding what supports a release of product
2. Breaking down release into features
3. Prioritizing the features for testing
4. Deciding which features will be tested and which will not be; and
5. Gathering details for estimating resources for testing.

The following factors drive the choice and prioritization of features to be tested

1) Features that are new and critical for the release:

- > New feature will have new program code and thus have higher susceptibility and exposure to defects. Since features are new both development and testing team will have to go through a learning phase.

> So these features are in high priority list for testing.

> Here product marketing team and selected customers participate in identification of features to be tested.

2) Features whose failures can be catastrophic:

Any feature which have highest or adverse impact on business has to be high on the list of features to be tested.

3) Features that are expected to be complex to test:

Early participation by testing team can help identify features that are difficult to test.

This can help in starting the work on these features early.

4) Features which are extensions of earlies features that have defect prone:

In regression testing, certain areas of code tend to be defect prone and such areas need detailed testing so that old defects will not come again or occur again.

c) Deciding test approach:-

After prioritizing feature list, next step is to decide test approach:

This includes identifying.

1. What type testing would you use for testing the functionality.
2. What are the scenarios for testing for testing the features.
3. Which type integration testing will be performed.
4. What type localization would be needed.
5. What “non – functional” test would need to do?
6. Specify the metrics to be collected.
7. Identify significant constraint on testing, such as test-item availability and deadline.

Setting up criteria for testing:

- > For each phase of testing there must be entry and exit criteria.
- > The entry criteria for test specify threshold criteria for each phase or type of test.
- > For entire testing activity entry criteria exists.
- > The exit criteria specify when a test cycle or testing activity can be complete.
- > Suspension criteria specify when a test cycle or test activity can be suspended.
- > Resumption criteria specify when the suspended test can be resumed.

Some of suspension criteria includes:-

1. Encountering more than a certain number of defects.
2. Hitting show stoppers that prevent further progress of testing.
3. Developers releasing a new version which should be used in the place of the product under test.

When solution to above problem is found, then test can resume.

Identifying Responsibilities, Staff and Training needs.

> A testing project requires different people to play different roles.

Responsibilities:

- > Identify the groups responsible for managing, designing, preparing, executing, checking and resolving
- > Identify the groups responsible for providing the test item identified in the test item section
- > Identify the groups responsible for providing the environmental needs

Staffing and Training needs:

- > Specify staffing needs by skill level
- > Identify training needs for providing necessary skill

Resource Requirement:

In test plan project or test manager should provide estimates for the various hardware and software resources required.

Following Points needs to be considered:

1. Machine configuration needed to run product under test.
2. Different configuration or versions of the software that must be present.
3. Need of automation tool like load runner, win runner, QTP etc.
4. Supporting tools such as compilers, test data generators, configuration management tools and so on.
5. Appropriate number of licenses of all software.

Test Deliverable:

Test plan identifies deliverable that should come at the end of testing activity.

The deliverable should be reviewed and approved by appropriate stakeholders.

Following are test deliverable:

- > Test plan
- > Test case design specification

- > Test cases (manual and automated)
- > Test logs produced by running the test
- > Test summary report
- > Bug report written in defect repository

Testing Tasks:- Size and effort estimation.

Estimation happens broadly in three phase.

- (1) Size estimation- It gives the actual amount of testing that needs to be done.

Size estimation can be done as follows:

- 1) line of code- it depends on programming language, style programming, compactes of programming etc.
- 2) Functional point- independent of programming language.
> it depends on function / features
- 3) Application size- It includes number of screens, reports or transaction

- (2) Effort Estimation:

Effort estimate is derived from size estimate by taking the individual work break down structure units and classifying them as “reusable”, “modification” and new development

- (3) Schedule estimation:

Schedule estimation entail translating the effort required into specific time frames.

Also following points are need to be considered:

- * Identify the task necessary to prepare for and perform testing
- * Identify all the task inter dependency
- * Identify any special skills required

3.2 Test Management:-

Choice of standards

> Standards are important part of planning in any organization.

> Standards are of two types:

External standards:-are standards that a product should comply with, are externally visible are usually stipulated by external consortia. Compliance to external standards is usually mandated by external parties. Example: ISO

Internal standards:-are standards formulated by testing organization to bring consistency. It standardize the processes and methods of working within the organization

Internal standards include

1. Naming and storage convention for test artifacts:-

Every test artifact like test cases, test result, test specification and so on have to be named appropriate and meaningfully.

2. Documentation standards:-

Documentation standard specify how to record information about the test within the test script themselves

3. Test coding standards:-

Test coding standards go one level deeper into the test and it feels how the test should be written.

Test infrastructure management:

Testing infrastructure is made up of three essential elements.

1. A test case database (TCDB) It captures all the relevant information about the test cases in organization.

2. Defect Repository capturer all the relevant details of defects reported for product.

The defect repository is tool of communication that influences the workflow within a software organization.

3. Configuration management repository and tools.

An software configuration management repository also known as keeps track of change control and version control of all the files/entities that makeup a software product.

TCDB, defect repository and SCM repository should complement each other and work together in an integrated fashion.

Test people management

People management is an integral part of any project management.

A person relies on his or her own skills to accomplish an assigned activity.

People management also requires the ability to hire, motivate and retain the right people .

Success of testing organization depends on careful people management skills .

Integration with product release:

> Success of a product depends on the effectiveness of integration of the development and testing activities.

> Both team should work in co-ordination with each other.

> Schedule of testing have to be considered or linked to product release.

Following points to be decided for this planning:

1. Synchronization between testing and development as to when different types of testing can start. When unit testing or system testing could start and so on.
2. Services level agreement between development and testing as to how long it would take for testing team to complete the testing.
3. Definition of the various priorities and severities of the defects. Development and testing team should have same vision.
4. Establish communication mechanisms to communication group to ensure that the document is kept in sync with product development and testing.

The purpose of the testing team is to identify the defect in the product and the risk that could be faced by releasing the product with the existing defects.

3.3 Test process

Base lining a test plan:

- > Every organization have template of test plan.
- > The test plan is reviewed by a designated set of competent authority in organization.
- > After this plan is approved by authority who is responsible for testing.
- > After this, the test plan is baseline into the configuration management repository.
- > From then on the baseline test plan becomes the basis for running the testing project.
- > Any significant changes in the testing project should thereafter be reflected in the test plan and changed test plan baseline again the configuration management repository.

Test Case Specification:-

By using test plan as the basis, the testing team designs test case specification which then becomes the basis for preparing individual test cases.

Test case specification should clearly identify

1. The purpose of the test:-This will list which feature the test is intended for
2. Items being tested along with their version release number
3. Environment that needs to be set up for running the test case.eg hardware environment set up, software environment set up
4. Input data to be used for the test case. This choice is depend on technique followed in the case. For example manual or automated or type of testing (equivalence portioning or boundary value analysis)
5. Steps to be followed to execute test

6. The expected result that are considered to be correct results
7. A step to compare the actual result produced with the expected result
8. Any relationship between this test and other tests:-by this we can find dependencies among the test

Update of Tracibility Matrix:

Tracibility matrix is tool to validate that every requirement is tested. It provides mapping between requirement and test cases.

- > The tracibility matrix is created during the requirement gathering phase.
- > Unique identifies is assigned to each requirement.
- > As project proceeds through design phase identifies for design feature is entered in matrix.
- > When project enters into coding phase identifies for program file is entered in the tracibility matrix.
- > When test case specification is complete the row corresponding to the requirement which is being tested by the test case is updated with the test case specification identifies.

4.Defect Management

A Software **Defect / Bug** is a condition in a software product which does not meet a software requirement (as stated in the requirement specifications) or end-user expectations (which may not be specified but are reasonable).

In other words, a defect is an error in coding or logic that causes a program to malfunction or to produce incorrect/unexpected results.

The defect may be introduced in a system due to various reasons as given below:

- Requirements not defined clearly by customer/business analyst/system analyst, or there are understanding related problems as knowledge transfer is not effective.

Sometimes queries are not resolved completely, and development team and designer may make some assumptions.

- Designs are wrong and not capable of implementing requirements. There is missing traceability between requirements and designs.
- People are not trained to do their work in collecting requirements, creating designs, coding, implementing, testing and deploying the product. People lack competency to perform assigned tasks.
- Processes used for software development, testing and deployment are not capable. They are not able to deliver the right product.

The approaches of defect management are:

- Defect Detection: Defect detection techniques identify defect and its origin.
- Defect Prevention: Defect prevention is a process of minimizing defects and preventing them from reoccurrence in future.

4.1 Defect Classification:

The classes of defects are: requirements / specifications, design, code and testing defect.

Requirements and specification defect: Requirement related defects arise in a product when one fails to understand what is required by the customer.

These defects may be due to customer gap, where the customer is unable to define his requirements, or producer gap, where developing team is not able to make a product as per requirements.

Defects injected in early phases can persist and be very difficult to remove in later phases. Since any requirements documents are written using natural language representation, there are very often occurrences of ambiguous, contradictory, unclear, redundant and imprecise requirements. Specifications are also developed using natural language representations.

Over the past several years many organizations have introduced the use of formal specifications language that, when accomplished by tools help to prevent incorrect descriptions of system behaviour.

1. requirements/specifications defects are:

- **Functional Defects:** The overall description of what the product does and how it should behave, is incorrect, ambiguous and/or incomplete. These defects are mainly about the functionalities present / absent in the applications which are expected/not expected by the customer. Generally system testing concentrates on functionality as the first part of testing. Non-working functions and functions not provided as required may represent functional defects.
- **Feature Defects:** Features may be described as different characteristics of a software component or system. Features refer to functional aspects of the software that map to functional requirements as described by the users and clients. Features also maps to quality requirements such as performance and reliability. Feature defects are due to feature descriptions that are missing, incorrect, incomplete or unessential.
- **Interface Defects:** These are defects that occur in the description of how the target software is to interface with external software, hardware and users. These defects talk about various interfaces essential as per the requirement statement. Generally, these defects may be due to user interface problems, problems with connectivity to other systems including hardware etc. Many features interaction and interface description defects are detected using black box based test designs at the integration and system levels.

2 Design Defects: Design defects occur when system components, interactions between system components, interactions between the outside software/hardware, or users are incorrectly designed. This covers in the design of algorithms, control, logic/ data elements, module interface descriptions and external software/hardware/user interface descriptions. Design defects generally refer to the way of design creation or its usage while creating a product.

Design related defects are classified as follows:

Algorithmic and processing Defects: These defects occur when the processing steps in the algorithm as described by the pseudo code are incorrect.

For example, the pseudo code may contain a calculation that is incorrect specified, or the processing steps in the algorithm are written in the pseudo code language may not be in the correct order. Another examples are a step may be missing or a steps may not be duplicated, omission of error condition checks such as divide by zero.

Module Interface Defects: Module interface defects are about communication problem between various modules. If one module gives some parameters which are not recognized by another, it creates module interface defects. For example, incorrect and/or inconsistent parameter types, an incorrect number of parameters or an incorrect ordering of parameters.

System Interface Defects: System interface defects may be generated when application communication with environmental factors is hampered. System may not be able to recognize inputs coming from the environment or may not be able to give outputs which can be used by the environment.

User Interface Defects: User interface defects may be a part of system interface defects where the other system working with the application is a human being. User interface defects may be a part of navigation, look and feel type of defects which affect usability of an application. Other examples of user interface description defects are where there are missing or improper commands, improper sequence of commands, lack of proper messages and/or lack of feedback message for the user.

3 .Coding Defects: Coding defects may arise when designs are implemented wrongly. If there is absence of development/coding standards or if they are wrong, it may lead to coding defects.

Variable Declaration / Initialization Defects: This defect arises when variables are not initialized properly, or variable are not declared correctly. These types of defects refer to wrong coding practices which may arise due to Coding standards of development are not followed.

Database Related Defects: Database related defects may occur when a database is not created appropriately. It may be a part of design defects if database design is wrong, or it may be a coding defect its database is not implemented correctly as per design.

Documentation\Commenting Defects: Coding also needs adequate commenting to make it readable and maintainable in future. Also there must be adequate documentation associated with various stages of development so that it is useful in future for maintenance. It is very difficult to find such defect in black box testing.

Testing Defects: Testing defects are defects introduced in an application due to wrong testing, or defects in the test artifacts leading to wrong testing.

Test Case Design Defects: These would encompass incorrect, incomplete, missing inappropriate test cases and test procedures.

There can be defects in test plans, test cases and test data definition which can lead to defect introduction in software if it is not handled correctly.

- **Test Environment Defects:** Test environment defects may arise when test environment is not set properly. Test environment may be comprised of hardware, software, simulators, and people doing testing. If test environment definition and reality are mismatched, it may lead to defects. Test environment may include operator training also.

- **Test Tool Defect:** Generally it is assumed that there are no defects in a test tool. Any defect introduced by a test tool may be very difficult to find and resolve, as one may have to find the defect using manual tests as against automated tools.

Describe steps in Defect Management Process.

(Diagram-2Marks, Explanation-4Marks)

Defects found during Verification and validation process in SDLC must be recorded so that it helps in further analysis and root causes of the defect. The defects found during these phases are used to find weak areas of project /process so that action can be initiated to strengthening it.

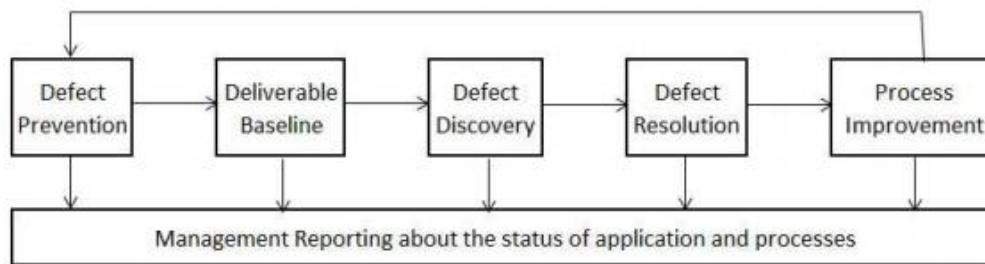


Fig: Defect management Process

Defect Prevention: It is a process of improving quality and productivity by preventing the defects into a software product. It is virtually impossible to eliminate the defects altogether.

Implementation of techniques, methodology and standard processes are used to reduce the risk of defects. Defect prevention is intended to remove the possibility of any defects before it occurs.

Deliverable Baselining :once the defect is fixed, retested and found to be closed, the product is created again .If the newly created product satisfies the acceptance criteria, it is base lined .only base lined work products can go to the next stage.

Defect Discovery: A defect is said to be discovered when it is brought to the attention of the developers and acknowledged (i.e., "Accepted") to be valid one.

Team should find defects before they become major problems. As soon as team finds the defects, they should report them so that those can be resolved.

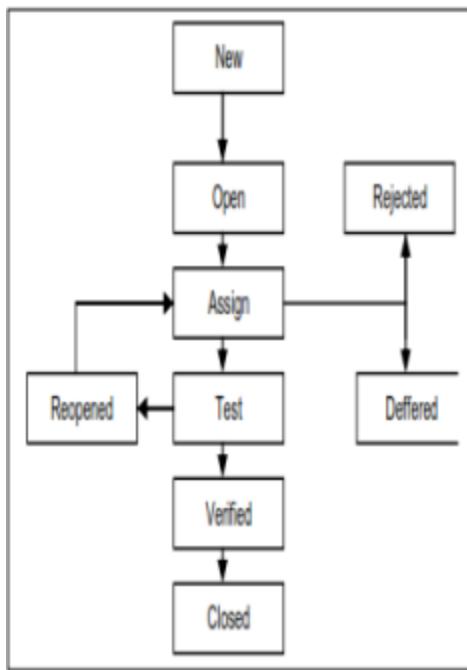
Team should also make sure that defects should be acknowledged by developers and should be valid one.

Defect Resolution: Work by the development team to prioritize, schedule and fix a defect, and document the resolution. This also includes notification back to the tester to ensure that the resolution is verified.

Process Improvement: All problems are due to failure in the process involved in creating software. Defects gives an opportunity to identify the problem with process used and update them. Better processes mean better product with less defect.

- **Management Reporting:** Analysis and reporting of defect information to assist management with risk management, process improvement and project management.

Describe defect life cycle with neat diagram.



The different states of bug life cycle are as shown in the above diagram:

- **New:** When the bug is posted for the first time, its state will be “NEW”. This means that the bug is not yet approved.
- **Open:** After a tester has posted a bug, the lead of the tester approves that the bug is genuine, and he changes the state as “OPEN”.
- **Assign:** Once the lead changes the state as “OPEN”, he assigns the bug to corresponding developer or developer team. The state of the bug now is changed to “ASSIGN”.
- **Test/Retest:** Once the developer fixes the bug, he must assign the bug to the testing team for next round of testing. Before he releases the software with bug fixed, he changes the state of bug to “TEST”. It specifies that the bug has been fixed and is released to testing team.// At this stage the tester do the retesting of the changed code which developer has given to him to check whether the defect got fixed or not.
- **Deferred:** The bug, changed to deferred state means the bug is expected to be fixed in next releases. The reasons for changing the bug to this state have many factors. Some of them are priority of the bug may be low, lack of time for the release or the bug may not have major effect on the software.

- **Rejected:** If the developer feels that the bug is not genuine, he rejects the bug. Then the state of the bug is changed to “REJECTED”.
- **Verified:** Once the bug is fixed and the status is changed to “TEST”, the tester tests the bug. If the bug is not present in the software, he approves that the bug is fixed and changes the status to “VERIFIED”.
- **Reopened:** If the bug still exists even after the bug is fixed by the developer, the tester changes the status to “REOPENED”. The bug traverses the life cycle once again.
- **Closed:** Once the bug is fixed, it is tested by the tester. If the tester feels that the bug no longer exists in the software, he changes the status of the bug to “CLOSED”. This state means that the bug is fixed, tested and approved.
- **Fixed:** When developer makes necessary code changes and verifies the changes then he/she can make bug status as „Fixed“ and the bug is passed to testing team.
- **Pending retest:** After fixing the defect the developer has given that code for retesting to the tester. Here the testing is pending on the testers end. Hence its status is pending retest.

Optional :

- **Duplicate:** If the bug is repeated twice or the two bugs mention the same concept of the bug, then one bug status is changed to “duplicate”.
- **Not a bug:** The state given as “Not a bug” if there is no change in the functionality of the application. For an example: If customer asks for some change in the look and field of the application like change of color of some text then it is not a bug but just some change in the looks of the application.

Defect Tracking Tools Following are some of the commonly used defect tracking tools:

- Bugzilla - Open Source Bug Tracking
- Testlink - Open Source Bug Tracking
- ClearQuest – Defect tracking tool by IBM Rational tools
- HP Quality Center– Test Management tool by HP 5.2.2

Defect Template

After uncovering a defect (bug), testers generate a formal defect report. The purpose of a defect report is to state the problem as clearly as possible so that developers can replicate the defect easily and fix it. Different software management tools offer defect templates, test case template is extended as defect template. Reporting a bug/defect properly is as important as finding a defect, if the defect found is not logged/reported correctly and clearly in bug tracking tools (like Bugzilla, ClearQuest etc.) then it won’t be addressed properly by the developers, so it is very important to fill as much information as possible in the defect template so that it is very easy to understand the actual issue with the software.

Table: Defect Report Template

ID	Unique identifier given to the defect. (Usually Automated)
Project	Project name.
Product	Product name.
Release Version	Release version of the product. (e.g. 1.2.3)
Module	Specific module of the product where the defect was detected.
Detected Build Version	Build version of the product where the defect was detected (e.g. 1.2.3.5)
Summary	Summary of the defect. Keep this clear and concise.
Description	Detailed description of the defect. Describe as much as possible but without repeating anything or using complex words. Keep it simple but comprehensive.
Steps to Replicate	Step by step description of the way to reproduce the defect. Number the steps.
Actual Result	The actual result you received when you followed the steps.
Expected Results	The expected results.
Attachments	Attach any additional information like screenshots and logs.
Remarks	Any additional comments on the defect.
Defect Severity	Severity of the Defect.
Defect Priority	Priority of the Defect.
Reported By	The name of the person who reported the defect.
Assigned To	The name of the person that is assigned to analyse/fix the defect.
Status	The status of the defect.

Attributes of defects:

- **Defect ID:** Unique identification number for the defect. The Defect ID is different for any project. It may be numeric or some distinctive identifier. Some organization use specific methods to describe a defect which is also called 'defect title'.
- **Defect Name:** Name of the defect must explain the defect in brief, its nature and type. It must be short but descriptive enough so that defect type can be identified based on it. This would also be useful to avoid duplicate defect entry.
- **Defect Description:** Detailed description of the defect including information about the module in which defect was found
- **Defect Type:** Definition of defect type may be declared in test plan. Some of the area types may be security defects (SEC), functionality defects (FUN), and user interface defects (GUI). Type of that may not be required, if defect ID already contains this information.

- **Project Name:** Project name indicates the project for which the defect is found.
- Version: Version of the application in which defect was found.

- **Steps:** Detailed steps along with screenshots with which the developer can reproduce the defects.
- **Date Raised:** Date when the defect is raised.

- **Reference:** where you provide reference to the documents like requirements, design, architecture or may be even screenshots of the error to help understand the defect

- **Detected By:** Name/ID of the tester who raised the defect.

- **Status:** Status of the defect. Typically the status of the defect is 'open', 'assigned', 'resolved', 'closed', 'hold', 'deferred', or 'reopened'.

- **Fixed by:** Name/ID of the developer who fixed it

- **Date Closed:** Date when the defect is closed

- **Severity:** Your bug report will also include Severity, which describes the impact of the defect on the application. Severities may be 'critical', 'high', 'medium' or 'low' depending on the impact of a defect. Severity definitions may be as follows:

1. S1= Defect where an application fails or particular functionality is completely missing. There is no work around available.

2. S2= Defect where functionality is not available but work around is possible. This may make customer unhappy but alternate way to achieve same result does exist.

3. S3= Defects of very minor nature (such as cosmetic defects) are in this category. Spelling mistakes, color and font mismatch may be put under least severity defects.

- **Priority:** Priority is defined based on how the project decides a schedule to take the defects for fixing. It is defined based on how many times the defect can be seen by normal users under normal circumstances, or how many customers are being affected due to this.

Priority which is related to defect fixing urgency. Severity Priority could be High/Medium/Low based on the impact urgency at which the defect should be fixed, respectively.

Priority definitions are:

1. P1= Defects having highest probability of affecting the customer. Either it affects most of the users or it appears at a place where maximum users will be visiting.

2. P2= Defects having lesser probability of affecting customer than P1 defects. They may be appearing in circumstances and hitting customer at fewer instances.

3. P3= Defects having minimum probability of affecting customer.

Estimate Expected Impact of a Defect:

Actual impact of a defect can be measured when the risk realizes or becomes a reality in production environment. Estimation may be done by different methods/approaches to find the probability of risk occurrence and impact when it becomes reality.

Some organizations categories risk impact as high, medium and low.

Ways to handle Risk: Risk handling is a management decision, either management from customer/user side or management from development side.

The following methods are some ways of handling risks.

- **Accept the Risk as It Is:** Some risks may not have any solutions. e.g., natural disasters.

The actions to reduce the probability and/or impact of certain risks may be very costly; hence the organization may not be able to implement them.

These risks may be accepted by the organization as it is. Such decisions are generally taken by the senior management.

They may prepare a fallback arrangement but may not define the ways of minimizing/eliminating risks.

This makes a team psychologically prepared to accept the risk so that the impact on the organization/project/user can be reduced to some extent.

Listing of known issues or defects in the product indicates acceptance of risk by the management. Customer user are given information about probable failures and effect of such failures.

This is a methodology of declaring ‘accident prone zone’ to users.

- **Bypassing the Risk:** If the application/approach is very risky to the users/customer, the management may decide to bypass the risk by avoiding the approach.

Bypassing of risk is required when the risk faced by the user cannot be accepted, or no action can be taken to reduce probability/impact arising due to realization of risk.

Minimize Risk Impact or Probability: Risk is a product of probability, impact, and detection ability. Risk minimization has three different methods of handling its probability, impact, or detection ability. The decisions may be driven by organization policy, values, cost-benefit analysis etc.

Minimization of problem due to risk happens in the following manner

- **Eliminate Risk:** Elimination of risk involves taking steps to remove risk from the root. Risk's probability is reduced to almost '0' by removing the causes of risk, so that the risk must not happen at all, or the organization user may be protected from the possible losses arising due to risk. Preventive controls can eliminate the probability of risk to a large extent.

Preventive controls are management-decided controls. Preventive controls are applied if the probability of occurrence is very high. Preventive controls are useless if the probability of happening is already negligible.

- **Mitigation of Risk:** Actions initiated by an organization to minimize the possible damage due to realization of risk are considered ‘mitigation actions’.

Mitigation actions are planned by an organization/project so that if the risk is realized, the impact due to it can be reduced to minimum possible.

The corrective controls used from the mitigation action. Corrective controls may be auto corrective or suggestive.

- **Detection Ability Improvements:** Impact of a risk is more if it catches the user unprepared. If people are aware of the risks, they can be well prepared to handle them. Generally, detective controls are used to increase the visibility towards risks. Sometimes, detective controls give threshold to corrective controls.

- **Contingency Planning:** Contingency planning refers to the actions initiated by an organization, when preventive or corrective actions fail, and risk actually occurs.

They are previously planned ways of tackling risks when all other planned activities for reducing probability and impact of the risk fail, and the risk becomes reality.

Techniques for finding Defects:

No system in the world is defect free.

Testers/developers need definition of methods and processes to find the defects and fix them, to make software stronger and reduce the number of defects in the product delivered to customer. Defects are found either by pre planned activities specifically intended to uncover defects (e.g., quality control activities such as inspections, testing, etc.) or by accident (e.g., users in production). Techniques to find defects can be divided into three categories:

- **Static techniques:** Testing that is done without physically executing a program or system.

A code review is an example of a static testing technique. Static techniques of quality control define checking the software product and related artifacts without executing them.

It is also termed ‘desk checking/verification/white box testing’. It may include reviews, walkthroughs, inspection, and audits. Here, the work product is reviewed by the reviewer with the help of a checklist, standards, any other artifact, knowledge, and experience, to locate the defect with respect to the established criteria.

Static technique is so named because it involves no execution of code, product, documentation etc. This technique helps in establishing conformance to requirements view.

- **Dynamic techniques:** Testing in which system components are physically executed to identify defects. Execution of test cases is an example of a dynamic testing technique.

Dynamic testing is a validation technique which includes dummy or actual execution of work products to evaluate it with expected behavior.

It includes black box testing methodology such as system testing.

The testing methods evaluate the product with respect to requirements defined; designs created and mark it as ‘pass’ or ‘fail’. This technique establishes ‘fitness for use’ view.

- **Operational techniques:** An operational system produces a deliverable containing a defect found by users, customers, or control personnel i.e., the defect is found because of a failure.

Operational techniques typically include auditing work products and projects to understand whether the processes defined for development/testing are being followed correctly or not, and whether they are effective or not.

It also includes revisiting the defects before and after fixing and analysis. Operational technique may include smoke testing and sanity testing of a work product.

While it is beyond the scope of this study to compare the various static, dynamic, and operational techniques, the research did arrive at the following conclusions:

- **Both static and dynamic techniques** are required for an effective defect management program. In each category, the more formally the techniques were integrated into the development process, the more effective they were.
- Since static techniques will generally find defects earlier in the process, they are more efficient at finding defects.

Reporting a Defect:

Defect finding and reporting are the important steps in software development life cycle.

When we find defect, analyses it, take corrective and preventive actions for further steps.

Points for defects reporting are: Give Complete Record of Discrepancies: Complete description of a defect indicates the symptom observed by the tester, when the defect is located, corrective actions and preventive actions.

Project manager must conduct complete analysis of how the defect occurred and what was not done correctly.

They should perform root-cause analysis to identify and correct the defect and responsible processes which have resulted into the defect. Defect finding must lead to process improvement.

Defect Report Forms the Base for Quality Measurement: Nobody can find all the defects present in an application, and if defect is not found in testing, it does not mean that the software is good. But one may use ‘number of defects’ as a measure of lack of quality in a software program which would be delivered to the customer. It must include definition of severity, priority, and category of defects. More defects indicate that the software quality is not good, while fewer defects may not be able to certify that the product as good. The major purpose of defect management is to get information about the capability of the processes and project management in totality. Detect management may go through the following stages.

- **Correct the Defect:** Correcting the defect logged by the reviewer/tester in verification/validation during software development life cycle is the primary activity done by the author or developer responsible for defect fixing. Corrected detect may undergo review, unit testing and retesting. In case it

is not possible to correct the defect or it is not advisable to do so, it may be noted as known issue in release note.

- **Report Status of System:** The status of system with respect to organizational standards and acceptance criteria fulfillment can be assessed through status reporting. Number of defects found and fixed is a major measurement to establish the status of software. It helps in informing users, customer, and management about the readiness of an application for delivery, or requirements of any efforts needed for fixing them, and any level of retesting and regression testing required.
- **Gather Statistics to Predict Future:** Defect statistics is used by an organization to plan corrective and preventive actions at project/organizational level. It can be used for benchmarking a product as well as an organization and assessing the quality improvement programs initiated by the organization. Matured organizations use defect calculators for predicting future and initiate actions to improve the process capabilities.
- **Process Improvement:** Testing is not the most efficient way of improving software quality. Verification and validation activities are costly. Finding and fixing defects cannot improve quality of software as well as productivity of team. An organization must stress on quality improvement programs through process improvements to ensure that defects are not introduced in the system

5. Testing Tools and Measurements

How to select a testing tool? Explain in detail.

(Explanation – 4 Marks) [Note: Criteria or guidelines can be considered as an answer]

Ans: Criteria for Selecting Test Tools:

The Categories for selecting Test Tools are,

1. Meeting requirements.
2. Technology expectations.
3. Training/skills.
4. Management aspects.

1. Meeting requirements: There are plenty of tools available in the market but rarely do they meet all the requirements of a given product or a given organization.

Evaluating different tools for different requirements involve significant effort, money, and time.

2. Technology expectations: Test tools in general may not allow test developers to extend/modify the functionality of the framework.

So extending the functionality requires going back to the tool vendor and involves additional cost and effort.

3. Training/skills: While test tools require plenty of training, very few vendors provide the training to the required level. Organization level training is needed to deploy the test tools, as the user of the test suite are not only the test team but also the development team and other areas like configuration management.

4. Management aspects: A test tool increases the system requirement and requires the hardware and software to be upgraded. This increases the cost of the already-expensive test tool.

OR

Guidelines for selecting a tool:

1. The tool must match its intended use. Wrong selection of a tool can lead to problems like lower efficiency and effectiveness of testing may be lost.
2. Different phases of a life cycle have different quality-factor requirements. Tools required at each stage may differ significantly.

3. Matching a tool with the skills of testers is also essential. If the testers do not have proper training and skill, then they may not be able to work effectively.
4. Select affordable tools. Cost and benefits of various tools must be compared before making final decision.
5. Backdoor entry of tools must be prevented. Unauthorized entry results into failure of tool and creates a negative environment for new tool introduction

Enlist any four benefits of automation testing.

(Any four benefits, Each benefit -1 Mark)

Ans:

- 1. Save Time /Speed:** Due to advanced computing facilities, automation test tools prevail in speed of processing the tests. Automation saves time as software can execute test cases faster than human.
- 2. Reduces the tester's involvement in executing tests:** It relieves the testers to do some other work.
- 3. Repeatability/Consistency:** The same tests can be re-run in exactly the same manner eliminating the risk of human errors such as testers forgetting their exact actions, intentionally omitting steps from the test scripts, missing out steps from the test script, all of which can result in either defects not being identified or the reporting of invalid bugs (which can again, be time consuming for both developers and testers to reproduce)
- 4. Simulated Testing:** Automated tools can create many concurrent virtual users/data and effectively test the project in the test environment before releasing the product.
- 5. Test case design:** Automated tools can be used to design test cases also. Through automation, better coverage can be guaranteed than if done manually.
- 6. Reusable:** The automated tests can be reused on different versions of the software, even if the interface changes.
- 7. Avoids human mistakes:** Manually executing the test cases may incorporate errors. But this can be avoided in automation testing
- 8. Internal Testing:** Testing may require testing for memory leakage or checking the coverage of testing. Automation can have done this easily.
- 9. Cost Reduction:** If testing time increases cost of the software also increases. Due to testing tools time and therefore cost is reduced.

What is software test automation? State types of test automation tools.

(Test automation - 2 marks; Types – 2 marks)

Ans: Test automation is the use of special software to control the execution of tests and the comparison of actual outcomes with predicted outcomes. The objective of automated testing is to simplify as much of the testing effort as possible with a minimum set of scripts. Test automation can automate some repetitive but necessary tasks in a formalized testing process already in place, or add additional testing that would be difficult to perform manually.

Types of test automation tools:

Static automation tools: These tools do not involve actual input and output. Rather, they take a symbolic approach to testing, i.e. they do not test the actual execution of the software.

e.g. Static analyzers(examine program systematically and automatically)

Code Inspectors(Inspect programs automatically to make sure they adhere to minimum quality standards)

Standard enforcers(Imposes simple rules on the developers)

Dynamic automation tools: These tools test the software system with live data.

e.g. Coverage Analyzer (Measure the extent of coverage)

Test Generators (To set up test inputs)

Output comparators (It is used to check predicted and actual output)

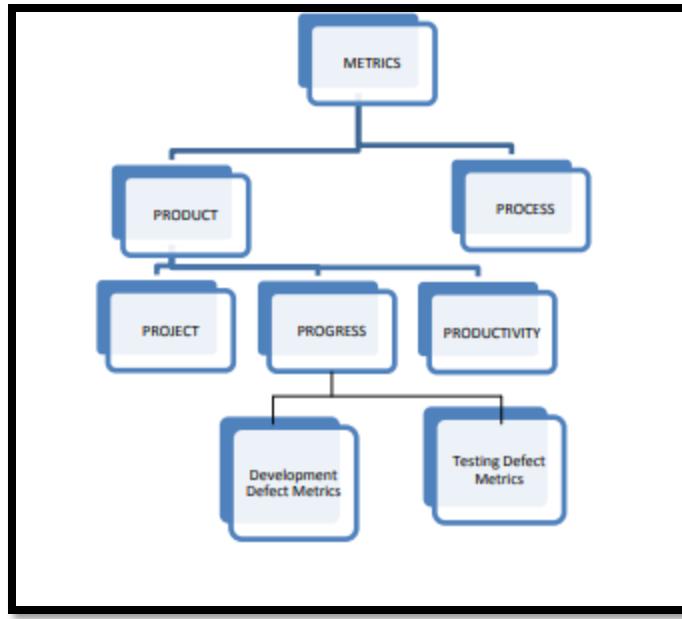
State the different metrics types with its classification.

(Stating metrics - 1 mark; Classification with explanation - 3 marks)

Ans: Metrics are basically classified as:

1. Product Metrics: Product metrics are measures of software product at any stage of its development, from requirements to installed system.

2. Process Metrics: Process metrics are measures of the software development process such as the overall development time, type of methodology used or the average level of experience of the programming staff.



Project Metrics: A set of metrics that indicates how the project is planned and executed.

o **Progress:** A set of metrics that tracks how the different activities of the project are progressing.

Progress Metrics is classified as

1. Test defect metrics 2. Development defect metrics

1. Test defect metrics: help the testing team in analysis of product quality and testing

2. Development defect metrics: help the development team in analysis of development activities.

o **Productivity:** A set of metrics that takes into account various productivity numbers that can be collected and used for planning and tracking testing activities.

OR

Other type of classification is: 1. Product vs. Process Metrics 2. Objective vs. Subjective Metrics

3. Primitive vs. Computed Metrics 4. Private vs. Public Metrics

Elaborate the advantages (any four) of using the test automation tools.

(Any four appropriate advantages of the test automation tools - 4 marks; 1 mark each)

Ans: Advantages of using the test automation tools are as given below:

1. Speed. The automation tools tests the software under tests with the very faster speed. There's a vast difference between the speed of user entering the data and the automated tools generating and entering the data required for the testing of the software. Speed of these software also completes the work faster.

2. Efficiency. While testers are busy running test cases, testers cannot be doing anything else. If the tester have a test tool that reduces the time it takes for him to run his tests, he has more time for test planning and thinking up new tests

3. Accuracy and Precision. After trying a few hundred cases, tester 's attention span will wane, and he may start to make mistakes. A test tool will perform the same test and check the results perfectly, each time.

4. Resource Reduction. Sometimes it can be physically impossible to perform a certain test case. The number of people or the amount of equipment required to create the test condition could be prohibitive. A test tool can be used to simulate the real world and greatly reduce the physical resources necessary to perform the testing.

5. Simulation and Emulation. Test tools are often used to replace hardware or software that would normally interface to your product. This "fake" device or application can then be used to drive or respond to your software in ways that you choose and ways that might otherwise be difficult to achieve.

6. Relentlessness. Test tools and automation never tire or give up. they can keep going and going and on and on without any problem; whereas the tester gets tired to test again and again.

State the contents of standard template of a test plan.

(template contents of a test plan - 4 marks) [**Note - any other relevant template shall also be considered**]

Ans: TEST PLAN TEMPLATE

1. Introduction

1.1 Scope What features are to be tested and what features will not be tested what combinations of environment are to be tested and what not.

2. References

3. Test Methodology and Strategy/Approach

4. Test Criteria

4.1 Entry Criteria

- 4.2 Exit Criteria
- 4.3 Suspension Criteria
- 4.4 Resumption Criteria
- 5. Assumptions, Dependencies, and Risks
 - 5.1 Assumption
 - 5.2 Dependencies
 - 5.3 Risk and Risk Management Plans
- 6. Estimations
 - 6.1 Size Estimate
 - 6.2 Effort Estimate
 - 6.3 Schedule Estimate
- 7. Test Deliverables and Milestones
- 8. Responsibilities
- 9. Resource Requirement
 - 9.1 Hardware Resources
 - 9.2 Software Resources
 - 9.3 People Resources
 - 9.4 Other Resources
- 10. Training Requirements
 - 10.1 Detail of Training Required
 - 10.2 Possible Attendees
 - 10.3 Any Constraints
- 11. Defect Logging and Tracking Process
- 12. Metrics Plan
- 13. Product Release Criteria

Differentiate between manual testing & automation testing.

Automated Testing	Manual Testing
<ul style="list-style-type: none"> If you have to run a set of tests repeatedly automation is a huge gain 	<ul style="list-style-type: none"> If Test Cases have to be run a small number of times it's more likely to perform manual testing
<ul style="list-style-type: none"> Helps performing "compatibility testing" - testing the software on different configurations 	<ul style="list-style-type: none"> It allows the tester to perform more ad-hoc (random testing)
<ul style="list-style-type: none"> It gives you the ability to run automation scenarios to perform regressions in a shorter time 	<ul style="list-style-type: none"> Short term costs are reduced
<ul style="list-style-type: none"> It gives you the ability to run regressions on a code that is continuously changing 	<ul style="list-style-type: none"> The more time tester spends testing a module the greater the odds to find real user bugs
<ul style="list-style-type: none"> It's more expensive to automate. Initial investments are bigger than manual testing 	<ul style="list-style-type: none"> Manual tests can be very time consuming
<ul style="list-style-type: none"> You cannot automate everything, some tests still have to be done manually 	<ul style="list-style-type: none"> For every release you must rerun the same set of tests which can be tiresome

Define software metrics. Describe product Vs process & objective Vs subjective metrics.

Ans:

Metrics are necessary to provide measurements of such qualities.

Metrics can also be used to gauge the size and complexity of software and hence are employed in project management and cost estimation.

Process quality: Activities related to the production of software, tasks or milestones.

1. Process metrics are collected across all projects and over long periods of time.
2. They are used for making strategic decisions.
3. The intent is to provide a set of process indicators that lead to long-term software process improvement.
4. The only way to know how/where to improve any process is to:
 - Measure specific attributes of the process.
 - Develop a set of meaningful metrics based on these attributes.
 - Use the metrics to provide indicators that will lead to a strategy for improvement.

Product quality: Explicit result of the software development activity, deliverables, products.

1. Product metrics help software engineers to better understand the attributes of models and assess the quality of the software.
2. They help software engineers to gain insight into the design and construction of the software.
3. Focus on specific attributes of software engineering work products resulting from analysis, design, coding, and testing.
4. Provide a systematic way to assess quality based on a set of clearly defined rules.
5. Provide an “on-the-spot” rather than “after-the-fact” insight into the software development.

Objective Metrics:

1. They are non-negotiable – that is the way they are defined does not change with respect to the type of endeavor they are being applied to.
2. Actual cost or AC is always the total cost actually incurred in accomplishing a certain activity or a sequence of activities.

Subjective Metrics:

1. These metrics are a relatively new precept and are more flexible than the rigid framework of the objective metrics.
2. Subjective metrics do deal with performance, but the approach is more tailored. For some enterprises, the niche in which they function forces project management to change in order to adapt to the demands of the workplace.

Give any two disadvantages of using testing tools.

Disadvantages of using testing tools:

- It is more expensive to automate. Initial investments are bigger than manual testing.
- You cannot automate everything; some tests still must be done manually.
- You cannot rely on testing tools always.

What do you mean by Software Metrics? List any three types of metrics.

Ans: Software metrics: Metrics are necessary to provide measurements of such qualities. Metrics can also be used to gauge the size and complexity of software and hence are employed in project management and cost estimation.

OR

Metrics can be defined as —STANDARDS OF MEASUREMENT

. Software Metrics are used to measure the quality of the project. Simply, Metric is a unit used for describing an attribute. Metric is a scale for measurement.

Types of software metrics are: 1. Project metrics 2. Progress metrics 3. Productivity metrics

State limitations of manual testing. Write any four.

(Any 4 appropriate limitations of manual testing - 4Marks; 1 Mark each)

Ans: Limitations of Manual Testing are as given below:

- i. Manual testing is slow and costly.
- ii. It is very labor intensive; it takes a long time to complete tests.
- iii. Manual tests don't scale well. As the complexity of the software increases the complexity of the testing problem grows exponentially. This leads to an increase in total time devoted to testing as well as total cost of testing.
- iv. Manual testing is not consistent or repeatable. Variations in how the tests are performed are inevitable, for various reasons. One tester may approach and perform a certain test differently from another, resulting in different results on the same test, because the tests are not being performed identically.
- v. Lack of training is the common problem, although not unique to manual software testing.
- vi. GUI objects size difference and color combinations are not easy to find in manual testing.
- vii. Not suitable for large scale projects and time bound projects. Batch testing is not possible, for each and every test execution Human user interaction is mandatory.
- viii. Comparing large amount of data is impractical.

Which different benefits help to recommend automated testing? Write advantages of switching to automated testing.

(Need / Benefit – 2 Marks, Advantages – 2 Marks)

Ans: **NEED of automated testing**

- i. An automated testing tool can playback pre-recorded and predefined actions, compare the results to the expected behavior and report the success or failure of these to a test engineer.
- ii. Once automated tests are created, they can easily be repeated, and they can be extended to perform tasks impossible with manual testing
- iii. Automated Software Testing Saves Time and Money. Software tests must be repeated often during development cycles to ensure quality. Every time source code is modified software tests should be repeated.

- iv. For each release of the software it may be tested on all supported operating systems and hardware configurations. Manually repeating these tests is costly and time consuming. Once created, automated tests can be run over and over again at no additional cost and they are much faster than manual tests.
- v. Testing Improves Accuracy Even the most conscientious tester will make mistakes during monotonous manual testing. Automated tests perform the same steps precisely every time they are executed and never forget to record detailed results
- vi. Increase Test Coverage Automated software testing can increase the depth and scope of tests to help improve software quality. Lengthy tests that are often avoided during manual testing can be run unattended. They can even be run on multiple computers with different configurations. Automated software testing can look inside an application and see memory contents, data tables, file contents, and internal program states to determine if the product is behaving as expected.

Structural Testing:

- The structural testing is the testing of the structure of the system or component.
- Structural testing takes into account the code, code structure, internal design and how they are coded.
- Structural testing is often referred to as ‘white box’ or ‘glass box’ or ‘clear-box testing’ because in structural testing we are interested in what is happening ‘inside the system/application’.
- The fundamental difference between structural testing and static testing is that in structural testing tests are actually run by the computer on the built product, whereas in static testing the product is tested by humans using just the source code and not the executable.
- In structural testing the testers are required to have the knowledge of the internal implementations of the code. Here the testers require knowledge of how the software is implemented, how it works.

During structural testing the tester is concentrating on how the software does it. For example, a structural technique wants to know how loops in the software are working. Different test cases may be derived to exercise the loop once, twice, and many times.

- Structural testing can be used at all levels of testing. Developers use structural testing in component testing and component integration testing, especially where there is good tool support for code coverage.
- Structural testing can be classified into unit/code testing, code coverage and code complexity testing.

Unit/Code Functional Testing: This part of structural testing is related to some quick checks that a developer performs before subjecting the code to more extensive code coverage testing or code complexity testing.

This happens by several methods:

- Initially developers can perform certain understandable test to know the input variable and the corresponding expected output variables. These are the quick test that checks out any obvious mistakes. These tests are repeated for multiple values of input variable therefore the confidence level of the developer are going to the next higher level increases. This can be done prior to formal reviews of static testing so that the review mechanism does not waste time catching obvious errors.
- For modules with complex logic or conditions, the developer can build a “debug version” of the product by putting intermediate print statements and making sure the program is passing through the right loops and iterations the right number of times. It is important to remove the intermediate print statements after the defects are fixed.
- Another approach to do the initial test is to run the product under a debugger or an Integrated Development Environment (IDE). These tools allow single stepping of instructions (allowing the developer to stop at the end of each instruction, view or modify the contents of variables, and so on), setting break points at any function or instruction, and viewing the various system parameters or program variable values. All the above methods are comes under the “debugging” category of activities than under the “testing” category of activities. All are familiarly related to the knowledge of code structure and hence we have included these under the “white box testing”

Code Coverage Testing:

- Code coverage is dynamic white box testing because it requires you to have full access to the code to view what parts of the software you pass through when you run your test cases. The simplest form of code coverage testing is using your compiler’s debugger to view the lines of code you visit as you single step through the program.
- Code coverage testing involves designing and executing test cases and finding out the percentage of code that is covered by testing. The percentage of code covered by a test is found by adopting a technique called instrumentation of code.
- Following are the types of coverage’s:

1) Statement Coverage: The statement coverage is also known as line coverage or segment coverage. The statement coverage covers only the true conditions. Through statement coverage we can identify the statements executed and where the code is not executed because of blockage. In this process each and every line of code needs to be checked and executed.

The statement coverage can be calculated:

Statement Coverage: $\frac{\text{Number of statement exercised}}{\text{Total number of statements}} * 100$

Statement coverage criteria call for having adequate number of test cases for the program to ensure execution of every statement at least once. In spite of achieving 100% statement coverage, there is every likelihood of having many undetected bugs.

Example-1

```
READ X  
READ Y  
IF X>Y  
PRINT "X is greater than Y"  
ENDIF
```

We can have 100% coverage by just one test set in which variable X is always greater than variable Y.

TEST SET 1: X=10, Y=5

Example-2

```
1 READ X  
2 READ Y  
3 Z =X + 2*Y  
4 IF Z> 50 THEN 5  
PRINT large Z  
6 ENDIF
```

TEST SET 1

Test 1_1: X= 2, Y = 3

Test 1_2: X =0, Y = 25

Test 1_3: X =47, Y = 1

- In Test 1_1, the value of Z will be 12, so we will cover the statements on lines 1 to 4 and line 6.
- In Test 1_2, the value of Z will be 50, so we will cover exactly the same statements as Test 1_1.
- In Test 1_3, the value of Z will be 49, so again we will cover the same statements.

Test 1_4: X = 20, Y = 25 This time the value of Z is 70, so we will print 'Large Z' and we will have exercised all six of the statements, so now statement coverage = 100%.

Note that Test 1_4 on its own is more effective which helps in achieving 100% statement coverage, than the first three tests together

Advantage of statement coverage:

- It verifies what the written code is expected to do and not to do .
- It measures the quality of code written.
- It checks the flow of different paths in the program and it also ensure that whether those paths are tested or not.

Disadvantage of statement coverage:

- It cannot test the false conditions.
- It does not report that whether the loop reaches its termination condition.
- It does not understand the logical operators.

2) Branch Coverage

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

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

Consider the program Listed below

```
1.Void main()
{
2.int A,B;
3.printf("enter two number");
4.Scanf("%d %d", &A,&B);
5..If(A>B)
{
6.Printf("%d is greater",A);
}
7.else
8.Printf("%d is greater",B);
}
```

If you test this program with the goal of 100 per cent branch coverage, you would need to run two test case with the A>B(i.e A=20 and B=10). The program would then execute the following path: Lines 1, 2, 3, 4, 5, 6, and if A<B (i.e A=10 and B=20). The program would then execute the following path: Lines 1, 2, 3, 4, 5, 7,8

Most code coverage analysers will account for code branches and report both statement coverage and branch coverage results separately, giving you a much better idea of your test's effectiveness

Advantages of Branch coverage:

- To validate that all the branches in the code are reached
- To ensure that no branches lead to any abnormality of the program's operation
- It eliminate problems that occur with statement coverage testing

Disadvantages of branch coverage: • This metric ignores branches within boolean expressions

3) Condition Coverage

Condition coverage is also known as predicate coverage.

An extra condition is added to the IF statement in line 5 that checks A>B as well as the A>C. Condition coverage testing takes the extra conditions on the branch statements into account.

Program

```
1.Void main()
{
2.int A,B,C;
3.printf("enter three numbers");
4.Scanf("%d %d", &A,&B,&C);

5.If((A>B)&& (A>C))
{
```

```

6.Printf("%d is greater",A);
}
7.elseif( B>A &&B>>C)
{
8.Printf("%d is greater",B);
}
9.else
{
10.Printf("%d is greater",C);
}

```

In this sample program, to have full condition coverage testing, you need to have the three sets of test cases shown in Table 3. 1. These cases assure that each possibility in the IF statement are covered.

Table 3.1

Sr No	Input	Line execution	
1	A=10 B=5 C=7	1,2,3,4,5,6	
2	A=10 B=12 C=9	1,2,3,4,5,7,8	
3	A=10 B=11 C=12	1,2,3,4,5,7,9,10	

In condition coverage testing, all three cases are important because they exercise different conditions of the IF statement.

As with branch coverage, code coverage analysers can be configured to consider conditions when reporting their results. If you test for all the possible conditions, you will achieve branch coverage and therefore achieve statement coverage.

4) Function coverage This is a new addition to structural testing to identify how many program functions (similar to functions in "C" language) are covered by test cases.

The requirements of a product are mapped into functions during the design phase and each of the functions form a logical unit. For example in a database software. "inserting a row into the database" could be a function. Or, in a payroll application, "calculate tax" could be a function. Each function could in turn be implemented using other function.

While providing function coverage, test cases can be written so as to exercise each of the different functions in the code.

Advantages of functional coverage:

- Functions are easier to identify in a program and hence it is easier to write test cases to provide function coverage Since functions are at a much higher level of abstraction than code, it is easier to achieve 100 percent function coverage than 100 percent coverage in any of the earlier methods.
- Functions have a more logical mapping to requirements and hence can provide a more direct correlation to the test coverage of the product. Functions provide one means to achieve traceability.
- Since functions are a means of realizing requirements, the importance of functions can be

prioritized based on the importance of the requirements they realize. Thus, it would be easier to prioritize the functions for testing. This is not necessarily the case with the earlier methods of coverage.

Function coverage provides a natural transition to black box testing.

We can also measure how many times a given function is called. This will indicate which functions are used most often and hence these functions - become the target of any performance testing and optimization.

As an example, if in networking software, we find that the function that assembles and disassembles the data packets is being used most often, it is appropriate to spend extra effort in improving the quality and performance of that function. Thus, function coverage can help in improving the performance as well as quality of the product.

Code Complexity Testing

BASIS PATH TESTING

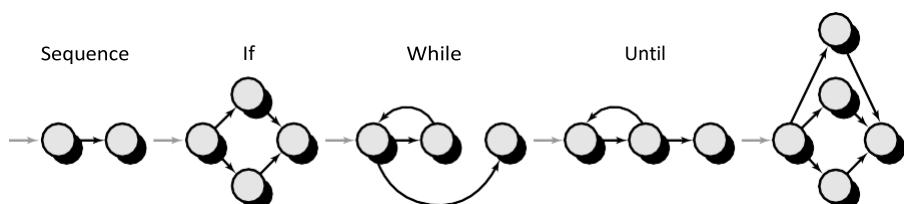
Basis path testing is a white-box testing technique first proposed by Tom McCabe.

Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

1. Flow Graph Notation

The flow graph depicts logical control flow using the notation. Each structured construct has a corresponding flow graph symbol. Here, a flowchart is used to depict program control structure.

The structured constructs in flow graph form:



Where each circle represents one or more non branching source code statements. Each circle, called a **flow graph node**, represents one or more procedural statements. A sequence of process boxes and a decision diamond can map into a single node. The arrows on the flow graph, called **edges** or links, represent flow of control and are analogous to flowchart arrows. Areas bounded by edges and nodes are called **regions**. Each node that contains a condition is called a **predicate node** and is characterized by two or more edges emanating from it.

Fig A: Flowchart

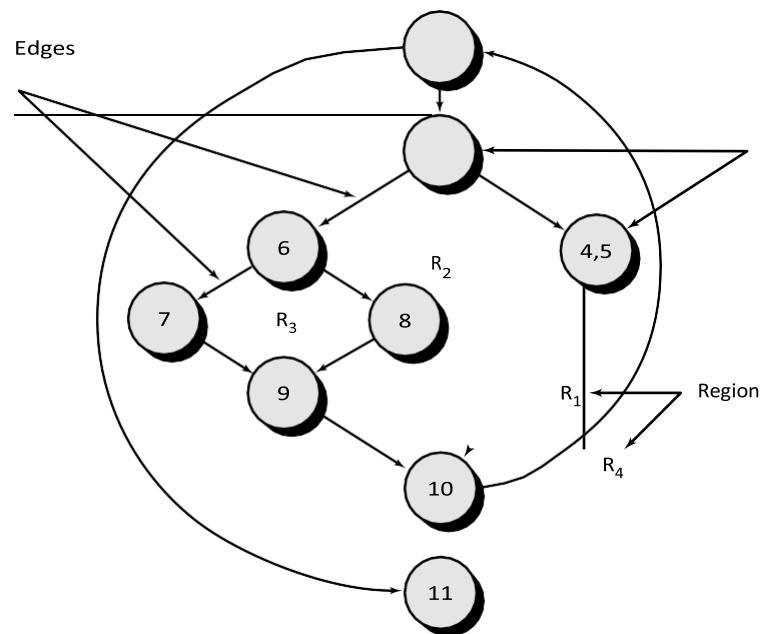
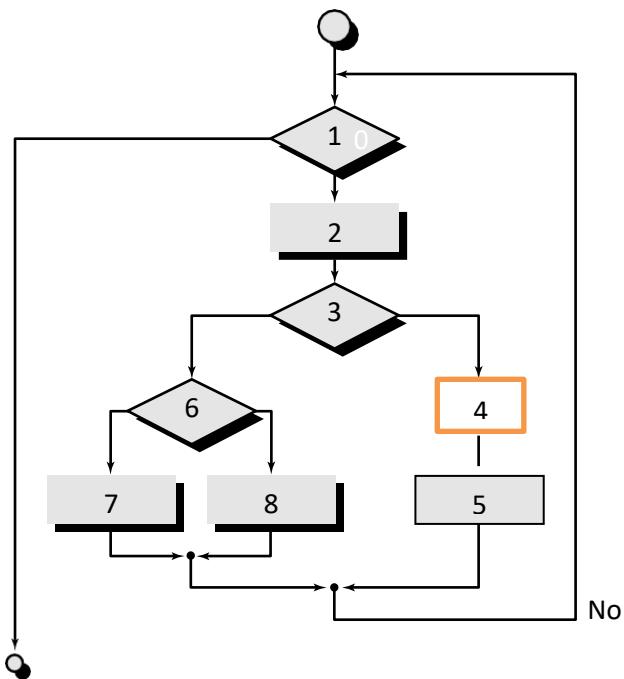


Fig B: Flow graph

2. Compound Logic

When compound conditions are encountered in a procedural design, the generation of a flow graph becomes slightly more complicated. A compound condition occurs when one or more Boolean operators (logical OR, AND, NAND, NOR) is present in a conditional statement.

Cyclomatic Complexity

Cyclomatic complexity is software metric that provides a quantitative measure of the logical complexity of a program.

When used in the context of the basis path testing method, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.

An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.

When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.

For example, a set of independent paths for the flow graph illustrated in Figure 17.2B is

path 1: 1-11

path 2: 1-2-3-4-5-10-1-11

path 3: 1-2-3-6-8-9-10-1-11

path 4: 1-2-3-6-7-9-10-1-11 . Paths 1, 2, 3, and 4 constitute a basis set for the flow graph.

If tests can be designed to force execution of these paths (a basis set), every statement in the program will have been guaranteed to be executed at least one time and every condition will have been executed on its true and false sides.

Complexity is computed in one of three ways:

1. The number of regions of the flow graph corresponds to the cyclomatic complexity.

2. Cyclomatic complexity, $V(G)$, for a flow graph, G, is defined as $V(G) = E - N + 2$

where E is the number of flow graph edges, N is the number of flow graph nodes.

3. Cyclomatic complexity, $V(G)$, for a flow graph, G, is also defined as $V(G) = P + 1$

where P is the number of predicate nodes contained in the flow graph G.

Referring once more to the flow graph in Figure B, the cyclomatic complexity can be computed using each of the algorithms just noted: 1. The flow graph has four regions.

2. $V(G) = 11 \text{ edges } 9 \text{ nodes } + 2 = 4$.

3. $V(G) = 3$ predicate nodes + 1 = 4.

Therefore, the cyclomatic complexity of the flow graph in Figure B is 4.