UNIT -6
Software Testing

1) Give programming principles, guidelines and programming practices.

- The principle and concepts that guide that guide coding task are closely aligned programming style, programming language and methods.
- Before starting the code, follow some principles and guidelines.
    1. Select data structures that will meet the need of the design.
    2. Keep conditional logic as simple as possible.
    3. Understand the software architecture and create interfaces that are consistent with it.
    4. Select meaningful variable names and follow other local coding standards.
    5. Write code that is self-documenting.
    6. Create a visual layout.
    7. Constrain your algorithm by structured programming practise.

- Programming practices help to avoid common errors.

1. Control construct

The single entry and exit constructs need to be used.

2. Use of goto

The goto statements make the program unstructured. So avoid use of goto statements as possible.

3. Information hiding

Hide complex things form end user it will increase usability of software.

4. Nesting

Structure inside another structure is called as nesting. If there is too deep nesting then it becomes hard to understand the code as well as complex.

5. User defined data types

User can define data type to enhance the readability of the code.

6. Modular size

The size of program may be large or small. There is no rule for size of the program. So as possible generate different module but not of large size.

7. Side effects

Sometimes if some part of code may change then it may generate some kind of problems called as side effects.

8. Robustness

If any kind of exception is generated, the program should generate some kind of output. Then it is called as robustness. In this situation the programs do not crash.

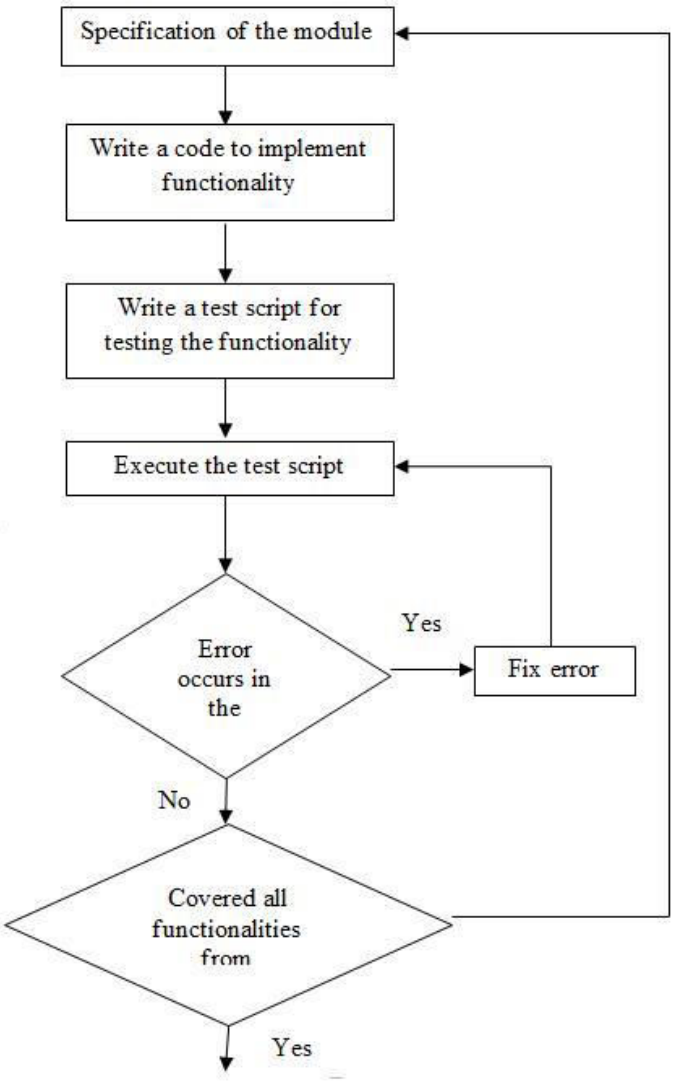
9. Switch case with defaults

Inside the switch case statement if any value which is unpredictable is given as argument then there should be default case to execute it.

10. Module interface Complex module interface must be carefully examined. If there are many parameters inside module interface, it must be broken.

2) Explain coding standards along with incremental development of code.

- To create any kind of good quality software, there is needed to follow some kind of standards.
- These standards should follow at each and every stage. Like for coding there are also some standards called as coding standard which are given below:



Incremental Development of Code

Naming Convention:

- To provide name there are some rules to be followed like, Variable name must not begin with numbers.
- Package name and variable name should begin with lower case.
- Constants must be in upper case.
- Name of variable should be meaningful.
- The variable with large scope must have long name. For example count_total, sum etc.
- The prefix must be used for Boolean type of variables.

Files

- Reader must get an idea about the purpose of the file by its name. in some programming langue there is some extension given to file.
- From the name of file, reader will have idea what is the content of file. Name of class and file name must be same.

Comment

- Comments are non-executable part of programme which increase readability
- Single line comment is given by //.
- Multi line comment is given /* and */.
- For the name of the variables comments must be given.

Statements

- Guidelines about declaration and executable statements:
- Class variable should never be declared.
- Avoid use of break and continue statements in the loop.
- Avoid complex conditional expressions, do...while statements.

- Declare some related variables on some line and unrelated variable in another line.

Advantages:

- The code become readable and can be understood easily.
- It helps in good programming practices.
- It brings uniform appearance in system implementation.

Incremental Development of Code

- Incremental Development of Code is shown in above figure whenever the design has been completed, the coding technique will be started. Before write a code, some specification for the functionalities must be available.
- For that purpose there will be some incremental approach for development.
- Here first of all specify the module and then write code to implement functionality.
- Then test that code based on some cases. Then execute the test script.
- It should be checked that any kind of errors are generated then fix the errors.
- If any kind of errors are not generated then covered all the functionalities mentioned in the specification, the process is terminated.
- Each and every functionality is written and immediately tested is one of its advantages.

3) Give management of code evaluation

Inside any organization each software developer will create a program module. After completion of every module they are integrated with each other. At that time there is needed to manage such code.

- The UNIX makes use of a utility called as CVS using which the source code can be controlled. Windows also provide such type of controlling utility by means of a tool called visual safe.
- The code management system contains a central repository in which there exists a control directory structure which keeps the full revision history of all the files.
- File history should be stored and using the older version new versions can be created. The repository is official source of the files.
- Various commands are performed on the repository are

1. Get a local copy

- The local copy of the repository is obtained by the programmer and programmer works on it. Making local copy is called checkout.

2. Make changes in the file

- The changes are made in local copy of the file. These changes are committed back in the repository. The operation is called as check in.

3. Update local copy

- Changes made in the local copy are not reflected in the local copy of the repository. Hence using the update command these changes are updated.

4. Get report

- A detailed report on the evolution made can be obtained. For example the report containing the old data and corresponding changes made, responses for the changes, corresponding files in which these changes must be reflected and so on.
- After making these changes, these changes must be available to all the team members of the project.
- If two processes try to access the same file at a time then it creates the conflicts. These conflicts must be solved manually.

4) Explain unit testing and load testing

- 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 module interface is tested to ensure that information properly flows into and out of the program unit under test.
- Tests of data flow across a module interface are required before any other test is initiated.
- Selective testing of execution paths is an essential task during the unit test. Test cases should be designed to uncover errors due to erroneous computations, incorrect comparisons, or improper control flow.
- Among the more common errors in computation are (1) misunderstood or incorrect arithmetic precedence, (2) mixed mode operations, (3) incorrect initialization, (4) precision inaccuracy, (5) incorrect

symbolic representation of an expression.

- The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
- Boundary testing is the last task of the unit test step. Software often fails at its boundaries. That is, errors often occur when the nth element of an n-dimensional array is processed, when the ith repetition of a loop with i passes is invoked, when the maximum or minimum allowable value is encountered.
- Test cases should uncover errors such as comparison of different data types, incorrect logical operators or precedence etc.
- All independent paths (basis paths) through the control structure are exercised to ensure that all statements in a module have been executed at least once. And finally, all error handling paths are tested.
- A driver is a "main program" that accepts test case data, passes such data to the component, and prints relevant results. Stubs serve to replace modules that are subordinate 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.

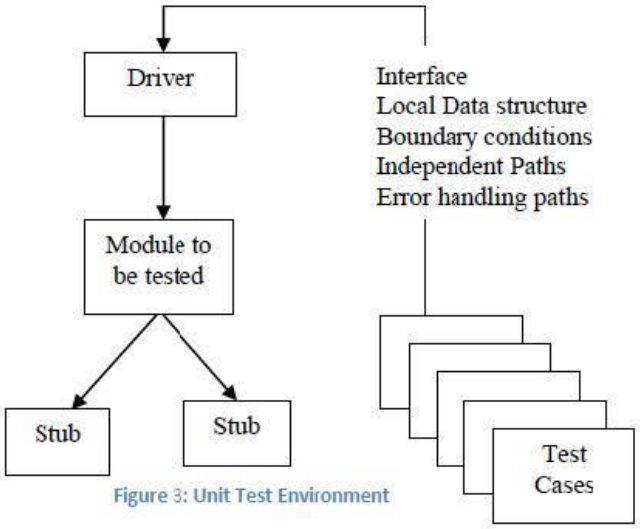


Figure 3: Unit Test Environment

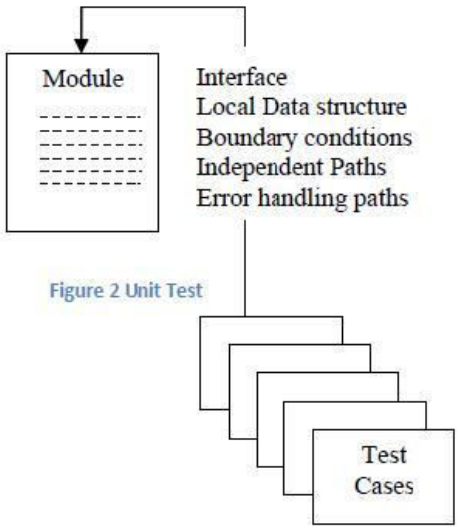


Figure 2 Unit Test

Load Testing

- Load testing is the process of putting demand on a system or device and measuring its response. Load testing is performed to determine a system's behavior under both normal and anticipated load conditions. It helps to identify the maximum operating capacity of an application. Load testing is a type of non-

functional testing.

- The term load testing is used in different ways in the professional software testing community.
- Load testing generally refers to the practice of modeling the expected usage of a software program by simulating multiple users accessing the program concurrently.
- For example, a word processor or graphics editor can be forced to read an extremely large document; or a financial package can be forced to generate a report based on several years' worth of data.
- The most accurate load testing simulates actual use, as opposed to testing using theoretical or analytical modelling.

5) Explain code inspection.

Code Inspection

- The goal of code inspection technique is to find errors and defects in the code. It will compile your code successfully.
- For that purpose there will be a team of member like,

- Moderator: - Manager or the leader of the code inspection team.
- Designer: - The team responsible for the current phase.
- Implementer: - The team responsible for the next phase.
- Tester: - The person who tests the code. This person must be preferably from SQA team.
- The design document and the document containing the code for the review are distributed to the team members during the code inspection.
- It also looks for the quality issue of the code.
- A checklist must be prepared while reviewing the code. A checklist is given below:

- Are the definitions exhibits the typing capacities of the language?
- Are there any dangling pointers?
- Is there any use of NULL pointer?
- Are all indexes within a bound?
- Are all indexes properly initialized?
- Is there any infinite loop?
- Is there any condition such as divide by zero?
- Are all the output variables got assigned with some value?
- Does the code satisfy all the local coding standards?
- Do actual and formal parameters of the function match?
- Is there any undeclared variable?
- Are all labels referred correctly?

Advantages

- It is very effective for finding the defects from the code.

Disadvantages

- The code inspection process is time consuming.
- A large number of people are involved in the code inspection process, it can be turned out to be costly process.
- Due to these drawbacks of the code inspection, a single person may carry out the code inspection and code reviews.

6) Explain metrics. Also give measurements, complexity metrics. Or what is cyclomatic complexity? Define steps to find cyclomatic complexity using flow graph.

Metrics are quantitative measure that the software engineer to gain the efficiency of the process.

- There are various measures available for any software products.

Size measure

- Size oriented measure is derived by considering the size of software that has been produced.
- Any organization builds a simple record of size measure for the software projects. It is built based on past experiences.
- Set of size measure is given below:

Size = Kilo Line of Code
Effort = Person month
Productivity = KLOC/Person-month
Cost = $/KLOC
Quality = Number of faults / KLOC
Documentation = Pages/KLOC

- Size measure is based on line of code computation.

Complexity Measure

- If the complexity is measured in terms of line of code then it may vary from system to system.
- Complexity can be done by various methods such as cyclomatic complexity, Halstead measure and Knot count measure.

Cyclomatic complexity

- Independent path is any path through use of the program that introduces at least one new set of processing statements or a new condition.
- Cyclomatic complexity is software metric that provides a quantitative measure of the logical complexity of a program.
- It defines number of independent paths in the basis of set of program and provides us with an upper bound for the number of tests that must be conducted to ensure all statements have

been executed at least once.

- It can be computed 3 ways:

1) The number of regions corresponds to cyclomatic complexity.

2) Cyclomatic complexity V (G) can be defined as

V (G) =E-N+2

Where E is number of flow graph edges and N is the number of flow graph nodes.

3) Cyclomatic complexity V (G) can also be defined as

V (G) = P + 1

Where P is the number of predicate nodes contained in the graph.

1) Explain various testing concepts. Error, Fault, and Failure

- The term **error** is used in two different ways. It refers to the discrepancy between a computed, observed, or measured value and the true, specified, or theoretically correct value.
- That is, error refers to the difference between the actual output of software and the correct output.
- Error is also used to refer to human actions that result in software containing a defect or fault.
- **Fault** is a condition that causes a system to fail in performing its required function.
- A fault is the basic reason for software malfunction and is synonymous with the commonly used term bug.
- Software has only "design faults"; there is no wear and tear in software.
- **Failure** is the inability of a system or component to perform a required function according to its specifications.
- A software failure occurs if the behavior of the software is different from the specified behavior.
- Failures may be caused due to functional or performance reasons.
- A failure is produced only when there is a fault in the system.
- However, presence of a fault does not guarantee a failure.

Testing Principles

- Before applying methods to design effective test cases, a software engineer must understand the basic principles that guide software testing.

1. All tests should be traceable to customer requirements.
2. Tests should be planned long before testing begins.
3. The Pareto principle1 applies to software testing.
4. Testing should begin “in the small” and progress toward testing “in the large.”
5. Exhaustive testing is not possible.
6. To be most effective, testing should be conducted by an independent third party.

Testability

In ideal circumstances, a software engineer designs a computer program, a system, or a product with
"testability" in mind.
This enables the individuals charged with testing to design effective test cases more easily.

Operability.

"The better it works, the more efficiently it can be tested."

Operability.

"The better it works, the more efficiently it can be tested."

Controllability.

"The better we can control the software, the more the testing can be automated and optimized."

Decomposability.

"By controlling the scope of testing, we can more quickly isolate problems and perform smarter retesting."

Simplicity.

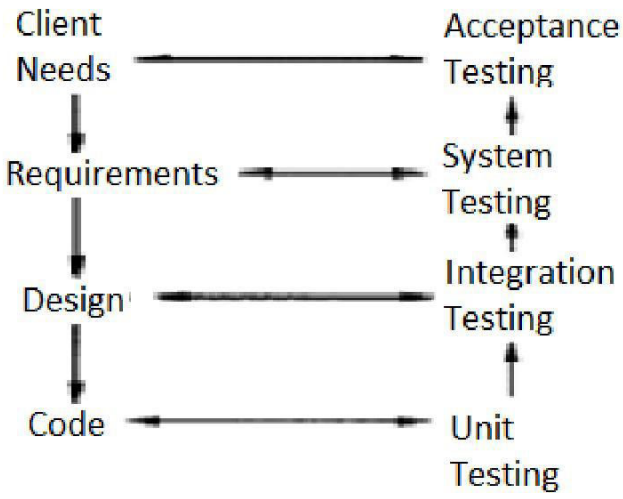
"The less there is to test, the more quickly we can test it."

Stability.

"The fewer the changes, the fewer the disruptions to testing."

2) Explain various Levels of Testing.

- Testing usually relies upon to detect faults remaining from earlier stages, in addition to the faults introduced during coding itself.
- Due to this, different levels of testing are used in the testing process; each level of testing aims to test different aspects of the system.
- The basic levels are **unit testing**, **integration testing**, and **system** and **acceptance testing**.
- These different levels of testing attempt to detect different types of faults.
- The relation of the faults introduced in different phases, and the different levels of testing are shown in Figure below.



- The first level of testing is called **unit testing**.
- In this, different modules are tested against the specifications produced during design for the modules.
- **Unit testing** is essentially for verification of the code produced during the coding phase, and hence the goal is to test the internal logic of the modules.
- It is typically done by the programmer of the module. A module is considered for integration and use by others only after it has been unit tested satisfactorily.
- The next level of testing is often called **integration testing**.
- In this, many unit tested modules are combined into subsystems, which are then tested.
- The goal here is to see if the modules can be integrated properly.
- Hence, the emphasis is on testing interfaces between modules.
- This testing activity can be considered testing the design.
- The next levels are **system testing** and **acceptance testing**.
- Here the entire software system is tested.
- The reference document for this process is the requirements document, and the goal is to see if the software meets its requirements.
- This is essentially a validation exercise, and in many situations it is the only validation activity.
- Acceptance testing is sometimes performed with realistic data of the client to demonstrate that the software is working satisfactorily.
- Testing here focuses on the external behavior of the system; the internal logic of the program is not emphasized.

4) Explain Testing Process in detail.

- The basic goal of the software development process is to produce software that has no errors or very few errors.
- In an effort to detect errors soon after they are introduced, each phase ends with a verification activity such as a review.
- However, most of these verification activities in the early phases of software development are based on human evaluation and cannot detect all the errors.
- This unreliability of the quality assurance activities in the early part of the development cycle places a very high responsibility on testing.
- In other words, as testing is the last activity before the final software is delivered, it has the enormous responsibility of detecting any type of error that may be in the software.
    - Furthermore, we know that software typically undergoes changes even after it has been delivered.
    - And to validate that a change has not affected some old functionality of the system, regression testing is done.
    - In regression testing, old test cases are executed with the expectation that the same old results will be produced.
    - Need for regression testing places additional requirements on the testing phase; it must provide the "old" test cases and their outputs.
    - Testing has its own limitations. These limitations require that additional care be taken while performing testing.
    - As testing is the costliest activity in software development, it is important that it be

done efficiently.

- All these factors mean that testing should not be done on-the-fly, as is sometimes done. It has to be carefully planned and the plan has to be properly executed.
- The testing process focuses on how testing should proceed for a particular project.

5) Explain Test Plan

- In general, testing commences with a test plan and terminates with acceptance testing.
- A test plan is a general document for the entire project that defines the scope, approach to be taken, and the schedule of testing as well as identifies the test items for the entire testing process and the personnel responsible for the different activities of testing.
- The inputs for forming the test plan are: (1) project plan, (2) requirements document, and (3) system design document.
- A test plan should include
  - Test unit specification
  - Features to be tested
  - Approach for testing
  - Test deliverables
  - Schedule and task allocation
- One of the most important activities of the test plan is to identify the test units.
- A test unit is a set of one or more modules, together with associated data, that are from a single computer program and that are the object of testing.
- The identification of test units establishes the different levels of testing that will be performed in the project.
- Generally, a number of test units are formed during the testing, starting from the lower level modules, which have to be unit-tested.
- Then the file input expected by the input module can contain the test cases.
- Features to be tested include all software features and combinations of features that should be tested.
- A software feature is a software characteristic specified or implied by the requirements or design documents.
- These may include functionality, performance, design constraints, and attributes.
- Testing deliverables should be specified in the test plan before the actual testing begins.

6) Explain Test Case Execution in detail.

- With the specification of test cases, the next step in the testing process is to execute them.
- The test case specifications only specify the set of test cases for the unit to be tested.
- However, executing the test cases may require construction of driver modules or stubs.
- It may also require modules to set up the environment as stated in the test plan and test case specifications.

- Only after all these are ready can the test cases be executed. Sometimes, the steps to be performed to execute the test cases are specified in a separate document called the test procedure specification**.**
  - This document specifies any special requirements that exist for setting the test environment and describes the methods and formats for reporting the results of testing.

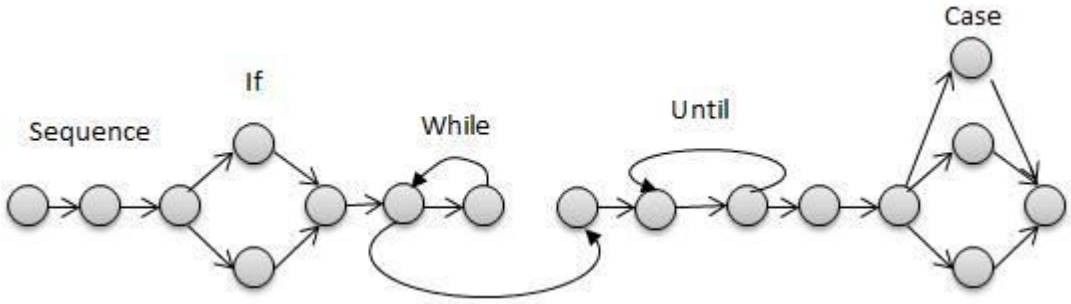
- Measurements, if needed, are also specified, along with methods to obtain them.
- Various outputs are produced as a result of test case execution for the unit under test.
- These outputs are needed to evaluate if the testing has been satisfactory.
- The most common outputs are the *test summary report,* and the *error report.*
- The test summary report is meant for project management, where the summary of the entire test case execution is provided.
- The summary gives the total number of test cases executed, the number and nature of errors found, and a summary of the metrics data collected.
- The error report is the details of the errors found during testing.
- The error report gives the list of all the defects found.
- The defects are generally also categorized into different categories.
- To facilitate reporting and tracking of defects found during testing (and other quality control activities), defects found must be properly recorded.
- This recording is generally done using tools.

7) Using example explain the basic path testing method.

- Basis path testing is a white-box testing technique first proposed by Tom McCabe.
- The basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.

a) Flow Graph Notation

- Before the basis path method can be introduced, a simple notation for the representation of control flow, called a flow graph (or program graph) must be introduced.
- The flow graph depicts logical control flow using the notation illustrated in Figure



To illustrate the use of a flow graph, we consider the procedural design representation in Figure B. • Here, a flowchart is used to depict program control structure.

Cyclomatic complexity

- Independent path is any path through use of the program that introduces at least one new set of processing statements or a new condition.
- Cyclomatic complexity is software metric that provides a quantitative measure of the logical complexity of a program.
- It defines number of independent paths in the basis of set of program and provides us with an upper bound for the number of tests that must be conducted to ensure all statements have

been executed at least once.

- It can be computed 3 ways:
- The number of regions corresponds to cyclomatic complexity.

- Cyclomatic complexity V (G) can be defined as

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

Where E is number of flow graph edges and N is the number of flow graph nodes.

- Cyclomatic complexity V (G) can also be defined as

$$V(G) = P + 1$$

Where P is the number of predicate nodes contained in the graph.

c) Deriving Test Cases

- The basis path testing method can be applied to a procedural design or to source code. So, presenting basis path testing as a series of steps are to be considered here.
- Note here, the average, although an extremely simple algorithm contains compound conditions and loops.
- The following steps can be applied to derive the basis set:

1. Using the design or code as a foundation, draw a corresponding flow graph.
2. Determine the cyclomatic complexity of the resultant flow graph.
3. Determine a basis set of linearly independent paths.
4. Prepare test cases that will force execution of each path in the basis set.

8) Give test case design.

Test cases are used to determine the presence of fault in the program. Sometimes even if there is some fault in our program the correct output can be obtained for some inputs

- Hence, it is necessary to exercise those set of inputs for which fault (if any) can be expressed.
- Executing test cases require money because –
  - i) Machine time is required to execute test cases
  - ii) Human efforts are involved in executing test cases. Hence in the project testing minimum number of test cases should be there as far as possible.
- The testing activity should involve two goals –
  - a Maximize the number of errors detected.
  - b Minimize the number of test cases.
- The selection of test case should be such that faulty module or program segment must be exercised by at least one test case.
- Selection of test cases is determined by some criteria which is called as test selection criterion. Hence the test selection criterion T can be defined as the set of conditions that must be satisfied by the set of test cases.
- Testing criterion is based on two fundamental properties – reliability and validity.
- A test criterion is reliable if all the test cases detected same set of errors.
- A test criterion is valid if for any error in the program there is some set which causes error in the program.

- For testing criteria there is an important theorem – "if testing criteria is valid and reliable if a set satisfying testing criterion succeeds then that means program contains no errors".
- Generating test cases to satisfy criteria is complex task.

9) What is software testing? What is the role of software tester? Compare: black box testing and white box testing.

Software Testing:

Once source code has been generated, software must be tested to uncover (and correct) as many errors as possible before delivery to the customer.

Role of Software Tester :

- The basic goal of testing is to uncover as much errors as possible Hence the intent of testing is to show how the program does not work.
- A tester should find out the possibilities wherein the program does not work. Hence testing should be carried out with the intension of finding out as much errors as possible.
- Testing is a destructive process in which the tester has to treat the program as adversary and should find out the presence of errors.
- One of the reason why organization do not select the developer as tester is depended upon the human psychology. It is quite natural that the man who creates something does not easily dare to find out something wrong in it. Hence the person who have develops the code, would design such test cases that will show that the code works correctly; he will not select the test cases that show errors in his creativity. Hence an independent team is recruited as a tester
- Another reason for testing the program by independent person is that sometimes the programmer does not understand the specification clearly, and then testing the code by independent group will find out the bug easily.
- This approach towards testing reveals as much errors as possible.

Black Box Testing	White Box testing
• It is done by separate Testing team. in this testers validates the developed application by execution of test cases and finding Bugs and send error reports.	• It is done by developers. It typically involves in coding. In this testing developers concentrates on internal structure of the program and justifies whether that program is correct or not.
• Generally black box testing will begin early in the software development i.e. in requirement gathering phase itself.	• But for white box testing approach one has to wait for the designing has to complete.
• We can use black testing strategy almost any size either it may be small or large.	• But white box testing will be effective only for small lines of codes or piece of codes.
• But in Black box testing we can do it.	• In white box testing we cannot test Performance of the application.

11) Explain White box and Black box testing. Discuss all the testing strategies that are available.

White Box Testing

White-box testing, sometimes called glass-box testing is a test case design method that uses the control structure of the procedural design to derive test cases. Using white-box testing methods, the software engineer can derive test cases that

(1) Guarantee that all independent paths within a module have been exercised at least once,
(2) Exercise all logical decisions on their true and false sides,
(3) Execute all loops at their boundaries and within their operational bounds, and
(4) Exercise internal data structures to ensure their validity.

- A reasonable question might be posed at this juncture: "Why spend time and energy worrying about (and testing) logical minutiae when we might better expend effort ensuring that program requirements have been met?" Stated another way, why don't we spend all of our energy on black-box tests? The answer lies in the nature of software defects:
- Logic errors and incorrect assumptions are inversely proportional to the probability that a program path will be executed. Errors tend to creep into our work when we design and implement function, conditions, or controls that are out of the mainstream. Everyday processing tends to be well understood (and well scrutinized), while "special case" processing tends to fall into the cracks.
- We often believe that a logical path is not likely to be executed when, in fact, it may be executed on a regular basis. The logical flow of a program is some-times counterintuitive, meaning that our unconscious assumptions about flow of control and data may lead us to make design errors that are uncovered only once path testing commences.
- Typographical errors are random. When a program is translated into programming language source code, it is likely that some typing errors will occur. Many will be uncovered by syntax and type checking mechanisms, but others may go undetected until testing begins. It is as likely that a typo will exist on an obscure logical path as on a mainstream path.
- Each of these reasons provides an argument for conducting white-box tests. Black-box testing, no matter how thorough, may miss the kinds of errors noted here. White-box testing is far more likely to uncover them.

Black-box Testing

Black-box testing, also called behavioral testing, focuses on the functional requirements of the software. That is, black-box testing enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program. Black-box testing is not an alternative to white-box techniques. Rather, it is a complementary approach that is likely to uncover a different class of errors than white-box methods.

Black-box testing attempts to find errors in the following categories:

(1) Incorrect or missing functions,
(2) Interface errors,
(3) Errors in data structures or external data base access,
(4) Behavior or performance errors, and
(5) Initialization and termination errors.

a) Graph-Based Testing Methods :

The software engineer begins by creating a graph—a collection of nodes that represent objects; links that represent the relationships between objects; node weights that describe the properties of a node (e.g., a specific data value or state behavior); and link weights that describe some characteristic of a link.

- The symbolic representation of a graph is shown in Figure 17.9A.
- Nodes are represented as circles connected by links that take a number of different forms.
- A directed link (represented by an arrow) indicates that a relationship moves in only one direction.
- A bidirectional link, also called a symmetric link, implies that the relationship applies in both directions.
- Parallel links are used when a number of different relationships are established between graph nodes.

b) Equivalence Partitioning

Equivalence partitioning is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived.

- An ideal test case single-handedly uncovers a class of errors (e.g., incorrect processing of all character data) that might otherwise require many cases to be executed before the general error is observed.
- Equivalence partitioning strives to define a test case that uncovers classes of errors, thereby reducing the total number of test cases that must be developed.
- Equivalence classes may be defined according to the following guidelines:

1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
3. If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
4. If an input condition is Boolean, one valid and one invalid class are defined.

c) Boundary Value Analysis

- Boundary value analysis is a test case design technique that complements equivalence partitioning.
- Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the "edges" of the class.
- Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well.
- Guidelines for BVA are similar in many respects to those provided for equivalence partitioning:

1 If an input condition specifies a range bounded by values a and b, test cases should be

designed with values a and b and just above and just below a and b.

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

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

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

- **Differentiate alpha testing and beta testing.**

χ) It is virtually impossible for a software developer to foresee how the customer will really use a program.

δ) Instructions for use may be misinterpreted; strange combinations of data may be regularly used; output that seemed clear to the tester may be unintelligible to a user in the field.

ε) When custom software is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements.

ϕ) Conducted by the end-user rather than software engineers, an acceptance test can range from an informal "test drive" to a planned and systematically executed series of tests.

γ) In fact, acceptance testing can be conducted over a period of weeks or months, thereby uncovering cumulative errors that might degrade the system over time.

η) If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance tests with each one.

ι) Most software product builders use a process called alpha and beta testing to uncover errors that only the end-user seems able to find.

φ) The alpha test is conducted at the developer's site by a customer.

κ) The software is used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems.

λ) Alpha tests are conducted in a controlled environment.

μ) The beta test is conducted at one or more customer sites by the end-user of the software. Unlike alpha testing, the developer is generally not present.

ν) Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer.

ο) The customer records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals.

π) As a result of problems reported during beta tests, software engineers make modifications and then prepare for release of the software product to the entire customer base.