Unit 6

Software Coding & Testing

Unit Outcomes (UOs) ;

After completion of this unit, students will be able to :

6.1 Follow coding standards

6.2 Test the software with proper test cases.

❖ Concept of coding

- In general, coding means set of guidelines for a specific programming language. Coding is what makes it possible for us to crate computer software, applications, websites etc.
- In the simple term, coding is →telling a computer what you want it to do, which involves typing in step-by-step commands for the computer to follow.
- Objective of coding → transform the design document into high level language code and unit test this code.
- Input to the coding phase → design document (module structure and data structure algorithms).
- The purpose of coding standards :
  - It gives a uniform appearance to the codes written by different engineers.
  - It enhances code understanding.
  - It encourages good programming practices.
- Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop.

6.1.1 Coding standards and guidelines

❖ Coding standards

- Software coding standards are language-specific programming rules that greatly reduce the probability of introducing errors into your applications, regardless of which software development model (iterative, waterfall, extreme programming, and so on) is being used to create that application.
- Good software development organizations normally require their programmers to have some well-defined and standard style of coding that is*coding standards.*
- Coding standards, sometimes referred to as programming styles or coding convention.
- Coding standards list several rules to be followed and coding guidelines provide general suggestions.
- The use of global should be limited.
- Contents of the headers preceding codes for different modules
  - Name of the module

- Date in which module is created
- Author's history
- Modification history
- Global variables used in the module

- Naming conventions for global variables, local variables, and constant identifiers.
- Error returns conventions and exception handling mechanisms.

❖ **Coding guidelines**

Coding guideline is a way to ensure that an entire team is building a cohesive code base. General coding guidelines provide the programmer with a set of the best methods which can be used to make programs more comfortable to read and maintain.

Some of the coding guidelines are listed below.

- Do not use a coding style that is too clever or too difficult to understand.
- Do not use an identifier for multiple purposes.
- Each variable should be given a descriptive name indicating its purpose.
- The code should be well-documented.
- The appropriate use of spaces within a line of code can improve readability.
- The length of any function should not exceed 10 source lines.
- Do not use 'goto' statements.
- Avoid deep nesting.

➥ **Characteristics of good coding**

- *Simplicity* – a code should be simple and easy to understand.
- *Readability* – a programmer should write simple code, which is easily readable by the user.
- *Good documentation* – code should be properly documented with header, comments, meaningful variable names etc. Software documentation can be classified into internal and external documentation.
- *Transportability* – a program should be easy to transport to different environments and computers.
- *Usability* – a code should have error recovery techniques that help the users for using the software or program.

6.1.2 Code review

- Conducting reviews is one of the important activities of system verification.
- Code review for a model is carried out after the module is successfully compiled and the all the syntax errors have been eliminated.
- According to IEEE standards, **a code review is→ "a process or meeting during which code is presented to project personnel, managers, users, customers or interested parties for comments or approval."**
- Vulnerabilities in the code are found in this meeting and the quality of the code should be improved.
- Code reviews are extremely cost-effective strategies for reducing errors and to produce high quality code.

Advantages of code review

- It improves code quality.
- It improves code readability.
- It improves the planning and estimation of project.
- Knowledge sharing is possible by code review.
- It helps us in improving domain expertise.

Classification of code review

- Code reviews can be classified into → Formal review and Informal review.

i. ***Formal reviews*** are conducted at the end of each life cycle phase. They are held when the author believes that the product is error free.
Results of the formal reviews are documented.

ii. ***Informal reviews*** are conducted on as-needed basis. They may be held at any time, serving the purpose of brainstorming session and no agenda set.
Results of the informal reviews are not documented.

Techniques of code review

- Two main techniques for code review are carried out on the code of module :

i. Code walkthrough
ii. Code inspection

(i) Code walkthrough

- Code walk through is an informal code analysis technique proposed by Fagan.
- This technique can be used throughout the software lifecycle to assess and improve the quality of the software products.
- A Code Walkthrough is an informal meeting where the programmer leads the review teamthrough his/her code and the reviewers try toidentify faults.
- In the process of code walkthrough, after a module has been coded, successfully compiled and all syntax errors eliminated, a few members of the development team are given the code few days before the walk through meeting to read and understand code. Each member selects some test cases and simulates (બનાવટી) execution of the code by hand.
- Members note down their findings and discuss them in the meeting.
- Several guidelines are produced in the meetings and accepted as examples.
- Some of the prerequisite guidelines are the following.

→ The team performing code walk through should not be either too big or too small. Ideally, it should consist of between three to seven members.

→ Discussion should focus on discovery of errors and not on how to fix the discovered errors.

Advantages of code walkthrough

- It is useful for the people if they are not from the software discipline; they are not used to or cannot easily understand software development process.
- It improves project team communication and morale of team members.
- It's an excellent educational medium for new team members.
- If it is done right, it can save time and improve quality over the project lifecycle.

➥ **Limitations of code walkthrough**

- The main disadvantage is that it takes more time.
- As this technique is informal, the documentation of this process is not done.

(ii) Code inspection

- Code inspection is a formal, efficient and economical methodof finding faults in design and code proposed by Fagan [1976].
- The goal of code inspection is → to identify and remove bugs before testing the code and to discover the algorithmic and logical errors.
- During code inspection, the code is examined in the presence of certain kinds of errors, in contrast to the hand simulation of code execution done in code walk through.
- Coding standards are also checked during code inspection.
- Good software companies list some common types of errors and these are discussed during code inspection to look out for possible errors.
- An inspection team consists of four persons who play the role of moderator, reader, recorder and author.
  - *Moderator* : he is a technically knowledgeable person. He leads the inspection process.
  - *Reader* : the reader takes the team through the code.
  - *Recorder* : he notes each error on a standard form.
  - *Authors* : he understands the errors found and try to solve unclear areas.

➥ **Following is a list of some general programming errors which can be checked during code inspection**

- ✓ Jumps into loop (in case of nested loop)
- ✓ Non-terminating loops.
- ✓ Array includes out of bound.
- ✓ Improper storage allocation.
- ✓ Use of uninitialized variables.
- ✓ Mismatch between actual and formal parameters.
- ✓ Use of incorrect logical operations.
- ✓ Control flows and computational expressions.

➥ **Advantages of code inspection**

- List all potential design flows. That makes software code maintainable and less costly.
- A detailed error feedback is provided to individual parameters.
- It makes easier to change in the code.

➥ ***Code walkthrough is informal technique lead by an author while code inspection is formal technique lead by moderator.***

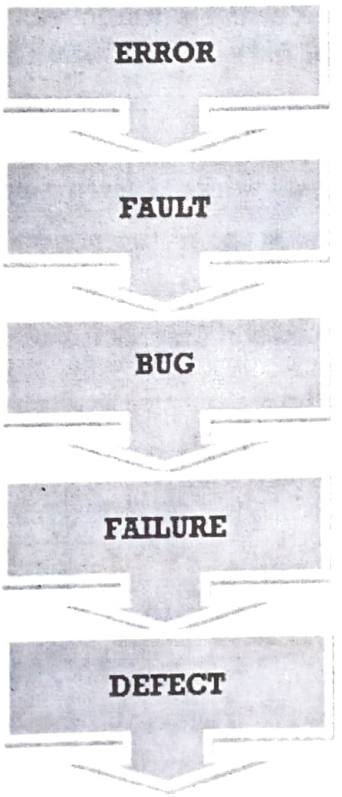
➥ ***Code walkthrough and code inspection both are static testing techniques.***

As this technique is formal, proper documentation of this method is done.

6.2.1 Testing

- The basic goal of any software development is to produce software that has no errors or has few errors. As we know that faults can occur during any phase of software development cycle.
- Verification is performed at output of each phase, but some faults are likely to remain undetected and they can affect the whole software.
- Testing is relied on to detect these faults. Testing is itself an expensive activity.
- If program fails to behave as expected, it needs to be debugged and corrected. For that testing is done.
- Testing is the process of executing a program to locate an error.
- Testing a program consists of providing the program with a set of test inputs (or test cases) and observing if the program behaves as expected.
- Aim of testing → to identify all defects existing in a software product.
- A good test case is one that has a high probability of finding undiscovered errors.

❖ **Some commonly used terms associated with testing are**



- ***Error*** **:** Error is a kind of mistake. This may be a syntax error or misunderstanding of specifications. Sometimes it may be logical error.
- ***Fault*** **:** an error may lead to one or more faults. More precisely a fault is the representation of an error.
- ***Bug*** **:** A software bug is an error or fault in a computer program or system that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

- ***Failure*** : When software is unable to perform as per the requirements, it is called failure.Failure occurs when a fault executes. This is a demonstrationof an error (or defect or bug).
- ***Defect*** : Defect is defined as the deviation from the actual and expected result of application or software

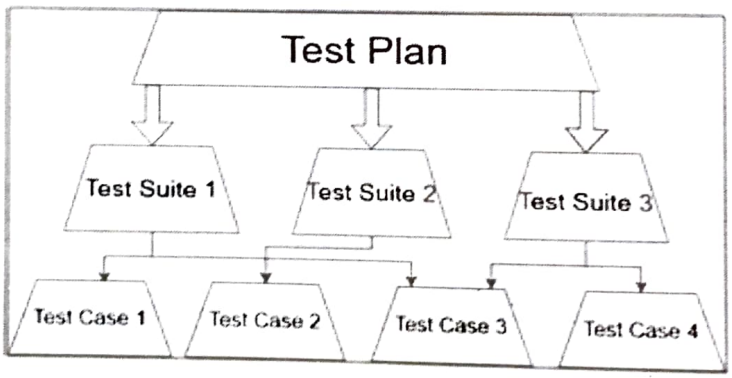
Some other testing terms are (test case and test suit)

❖ **Test case**

- Test cases define how to test a system, software or an application.
- Test case is a set of conditions or variables under which a tester will determine whether a system under test works correctly or satisfy requirements.
- This is the triplet [I,S, O], where ***I*** is the data input to the system, ***S*** is the state of the system at which the data is input, and ***O*** is the expected output of the system.
- Generally test cases are written by a tester or QA professionals.
- Test cases are run after the implementation of a software product to confirm whether the product meets its software requirements.
- There are mainly two types of test cases → Formal and informal.
- In formal test cases in which tester writes a test in which the inputs are all known and predefined inputs. And in case of informal test cases, tester don't have known inputs and outputs.

❖ **Test suite**

- This is the set of all test cases with which a given software product is to be tested.
- Test suite is created based on the scope of the project.
- The collection of test suite is referred as test plan.
- The concept of test case, test suite and test plan can be understood by the following figure



- ***Testing is four stage process:***

  Unit testing → subsystem testing → system testing → acceptance testing.

➥ **Design of test cases**

- We must design an optimal test suite that is of reasonable size and can uncover as many errors existing in the system as possible.
- Testing a system using a large collection of test cases that are selected at random does not guarantee that all of the errors in the system will be uncovered.

Example :

```
int x, y;
take values of x and y
from user
if (x>y) then
    x is greater;
else
    y is greater;
```

- ❖ Invalid test suit:
  {(x=3, y=2), (x=8, y=4), (x=18, y=12)}
- ❖ Valid test suit :
  {(x=3, y=2), (x=4,y=5)}

- Systematic approach should be followed to design an optimal test suit. Each test case is design to detect different errors.
- There are mainly two approaches to systematically design test cases :

Black box testing	White box testing

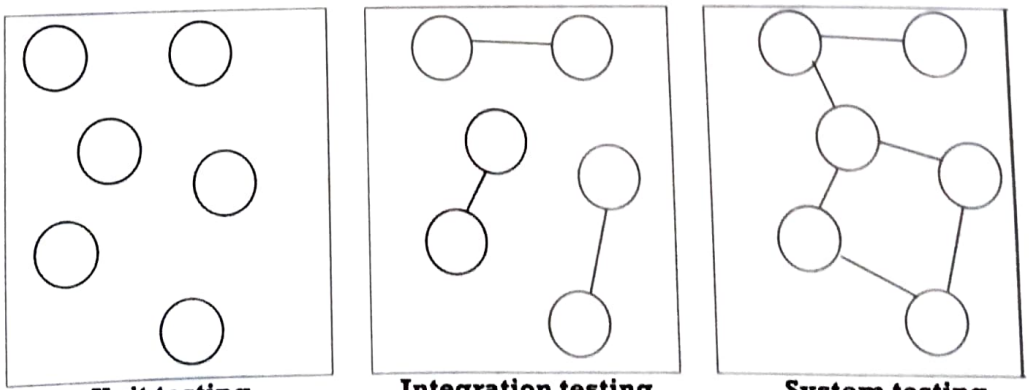
- In the ***black-box testing approach***, test cases are designed using only the functional specification of the software, i.e. without any knowledge of the internal structure of the software.
- So, black-box testing is known as *functional testing.*
- In the ***white-box testing approach***, designing test cases requires thorough knowledge about the internal structure of software.
- So, the white-box testing is called *structural testing.*

➥ **Testing in the large vs. testing in the small**

- Software products are normally tested first at the individual component (or unit) level. This is referred to as testing in the small.
- After testing all the components individually, the components are slowly integrated and tested at each level of integration (integration testing). Finally, the fully integrated system is tested (called system testing).
- Integration and system testing are known as testing in the large.

➥ **Levels of testing**

- There are three levels of testing:
  - **Unit testing :** it is first level of testing and individual module is tested.
  - **Integration testing :** combine the modules and test their detailed structures.
  - **System testing :** the whole system is tested ignoring the internal structure of the system.



Unit testing **Integration testing** **System testing**

Level of testing

- Initially all the modules are tested individually by their programmers, then interactions between these modules are tested (integration testing), then whole system as a single component is tested and finally the system is tested against users' data for final approval from the users.

❖ Verification and Validation

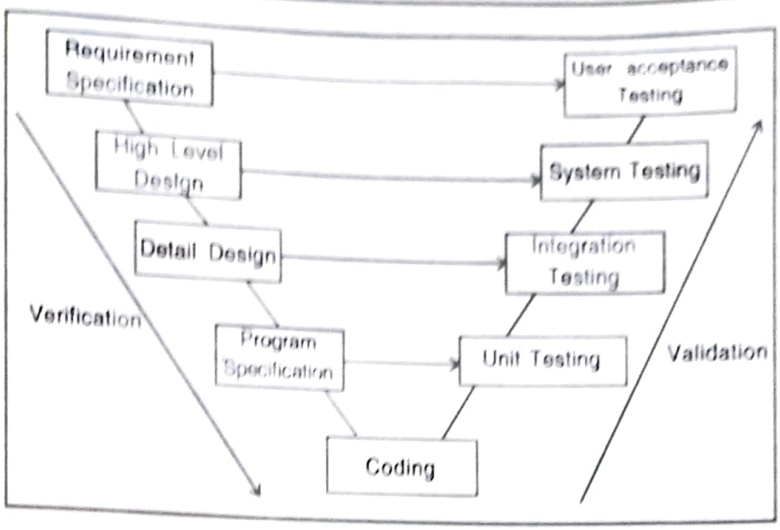
➥ Verification

- Verification is the process of checking that a software achieves its goal without any bugs.
- Verification is the process of determining whether the output of one phase of software development confirms to that of its previous phase.
- We can say that this is the process where we ensure that we are on track to complete the final result or not.
- These may comprise documentation like requirements specifications, design documents, database table design, ER diagrams, test cases, traceability matrix, and so on that are created throughout the development stages.
- It is a static testing.
- The cost of verification is less compare with validation.
- When we are doing verification testing, execution of code does not happen.
- Verification is concerned with *phase containment of errors.*
- For an organization verification and validation both are important, but verification comes before validation.
- In one statement we can say → verification is "are we doing right?"

➥ Validation

- Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements.
- Validation is the process of determining whether a fully developed system confirms to its requirements specification.
- It is dynamic testing.
- The cost of validation is more compare with verification.
- Validation is concerned with *final product be error free.*
- In case of validation testing, execution of code does happen.
- In one statement we can say → validation is "have we done right?"
- Validation comes after verification.
- Different activities involved in validation are:
  - ✓ Black box testing
  - ✓ White box testing
  - ✓ Unit testing
  - ✓ Integration testing

Verification and validation in software engineering can be illustrated using below figure



❖ Difference between verification and validation

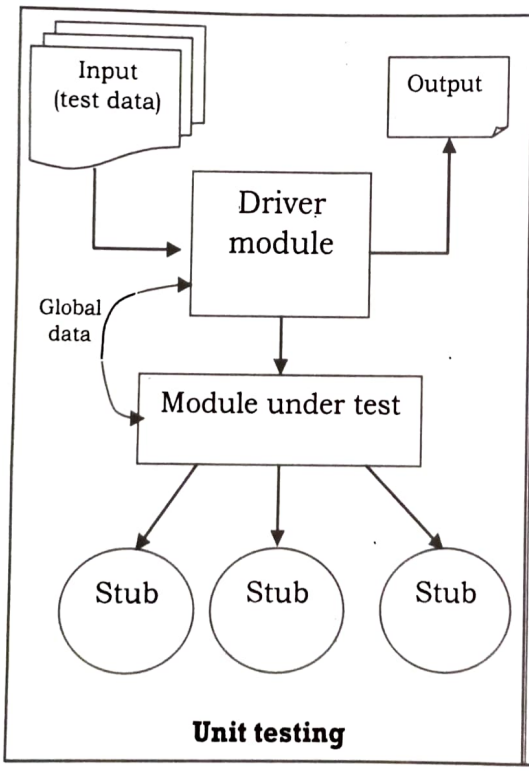
Verification	Validation
→ Verification is the process of confirming that software meets its specification.	→ Validation is the process of confirming that software meets customers' requirements.
→ Verification is the process of determining whether the output of one phase of software development confirms to that of its previous phase.	→ Validation is the process of determining whether a fully developed system confirms to its requirements specification.
→ Verification is concerned with phase containment of errors.	→ Validation is concerned with final product be error free.
→ Verification '"are we doing right?"	→ And Validation '"have we done right ?"
→ Verification comes before validation.	→ Validation comes after verification.
→ It is static testing.	→ It is dynamic testing.
→ Cost of verification is less.	→ Cost of validation is more.
→ Verification does not include the execution of code.	→ Validation includes the execution of code.

Both strategies are used to finds defects in software project, but in different way, Verification is used to identify the errors in requirement specifications & Validation is used to find the defects in the implemented software application.

❖ Unit testing

- Unit testing is the first level of testing.
- It is the process of taking a program module and running it in isolation form from the rest of the product.
- Unit testing is undertaken after a module has been coded and successfully reviewed.
- Unit testing (or module testing) is the testing of different units (or modules) of a system in isolation.

- This testing is a white box testing methodology.
- Purpose of unit testing is to validate that each unit of the software code performs as expected.
- Unit testing is done during development phase.
- A complete environment is needed to execute the unit testing on the module.
- Following steps are needed in order to test the module
  - The procedures belonging to other driver module which contains non-local data structure.
  - A procedure belonging to the module which is working as stub module.
  - A procedure belonging to the module which is under test.
- The calling module and called module should be unit tested.
- An issue with the unit testing is that unit testing is done as a part, not the whole system. But in execution, one module may use other modules that have not been developed yet. For that dummy modules are needed.
- Due to this, unit testing often require driver and/or stub modules. The role of driver and stub modules is described in below figure.



Unit testing

- Driver plays the role of "calling" module and getting test data and generating the result. It contains non-local data structure.
- Stub modules are "dummy" modules that are used in placed of actual modules to facilitate unit testing.
- So, if module X uses services of another module Y that hasn't been developed, then for unit testing of X, some stub module N is written for module X.
- The need of stub module can be avoided.

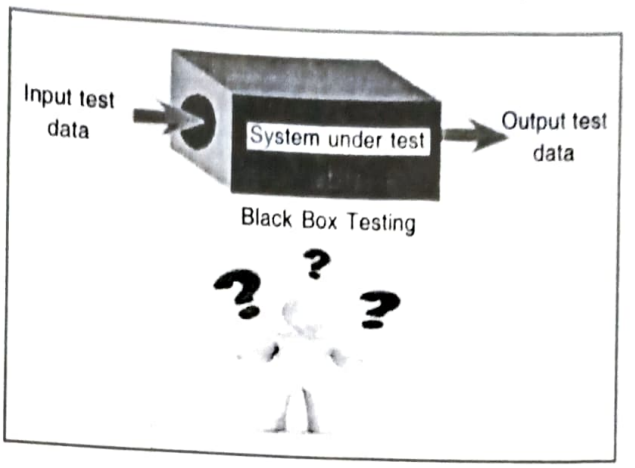
➥ **Advantages of unit testing**

- Improve the quality of the code.
- Fids the software bugs early.
- One of the main benefits of unit testing is that it makes the coding process more agile.
- It simplifies integration testing.

❖ **Black box testing**

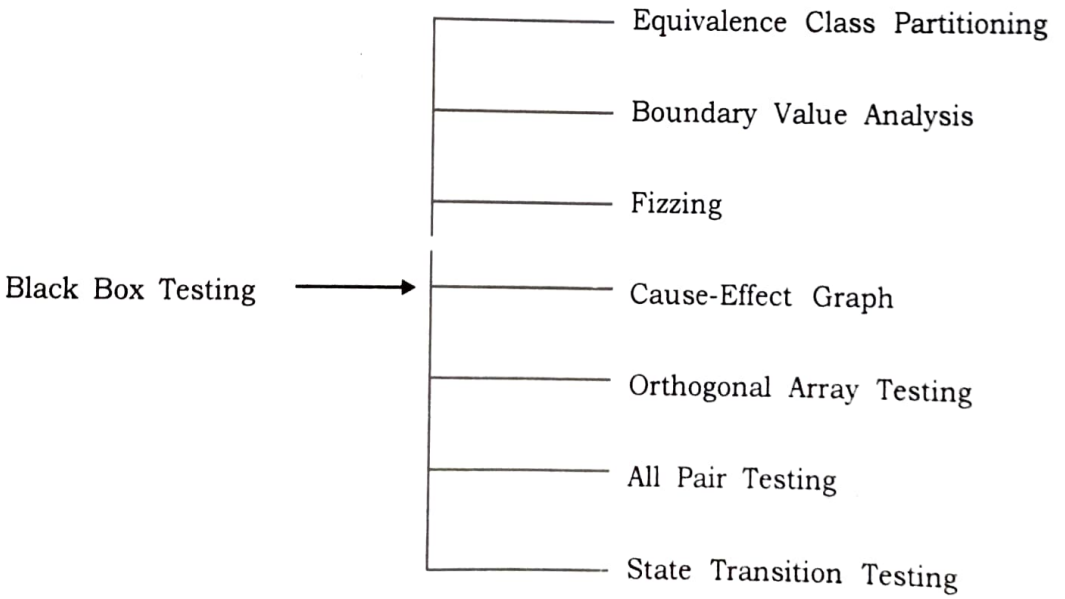
- This method is also called behavioural testing or functional testing.

- It is a testing method where test cases are designed using only the functional specification of the software, i.e. without any knowledge of the internal structure of the software.
- Functionality of the black box is understood completely in terms of its inputs and output.



Black box testing

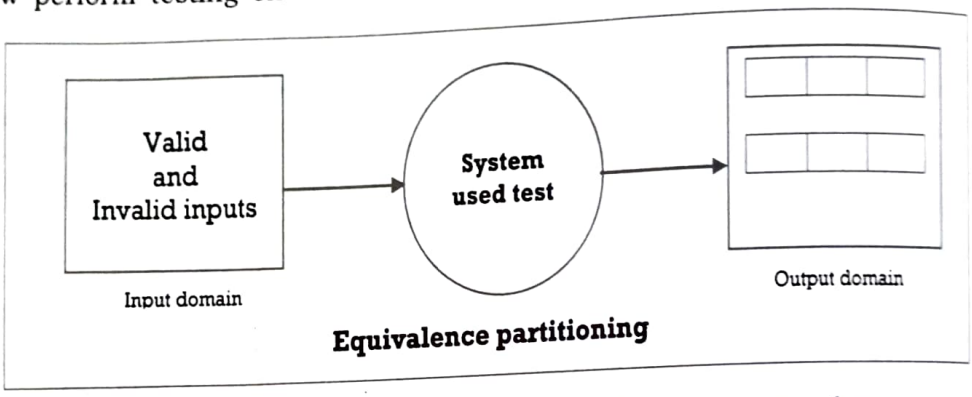
- Black box testing treats the software as a "Black Box" – without any knowledge of internal working and it only examines the fundamental aspects of the system. While performing black box test, a tester must know the system architecture and will not have access to the source code.
- The example of black box testing is search engine. You enter text that you want to search in the search bar, and results are returned to you. You don't know the specific process or algorithm of searching.
- There are number of techniques that can be used to design test cases in black box testing method, which are listed below.



Among all of the above, we will discuss only first two techniques which are very successful in detecting errors.

Equivalence class partitioning

- In it, the domain of input values (input test data) to a program is partitioned into a finite number of equivalence classes.
- So the behaviour of the program is similar for every input data belonging to the same equivalence class.
- The main idea behind defining the equivalence classes is that if one test case in a class detects an error all other test cases in the class would be expected to find same error.
- To implement this technique two steps are required.
  i. The equivalence classes are identified by taking each input condition and partitioning in into valid and invalid classes. For example, if an input condition specifies a range of values from 1 to 100, we identify one valid equivalence class (1 to 100); and two invalid equivalence classes (< 1) and (>100).
  ii. Generate test cases using the equivalence classes identified in the previous step. Now perform testing on both valid and invalid equivalence classes.



Equivalence partitioning

- The aim is to choose at least one element from each equivalence class.
- The following are some general guidelines for designing the equivalence classes:
- If the input data values to a system can be specified by a range of values, then one valid and two invalid equivalence classes should be defined.
- If the input data assumes values from a set of discrete members of some domain, then one equivalence class for valid input values and another equivalence class for invalid input values should be defined.
- If the input value is Boolean, then one valid and one invalid equivalence classes should be defined.
- For example, for a program that supposes to accept any number between 1 and 99, there are at least four equivalence classes from input side. Like:
  i. Any number between 1 and 99 is valid input.
  ii. Any number less than 1 (includes 0 and all negative numbers).
  iii. Any number greater than 99.
  iv. If it is not a number, it should not be accepted.

Boundary value analysis (BVA)

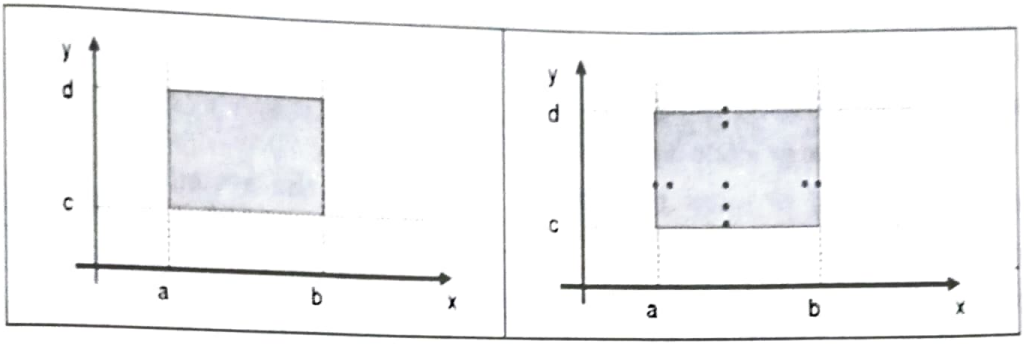
- Generally errors are occurred at boundary of domains rather than centre of domain.
- This is because test cases closer to boundary have more chance to detect errors.
- For this reason, boundary value analysis technique has been developed.

- In this technique, selection of test cases performed at the edges of the class. Suppose we have an input variable x with a range from 1 to 100. The boundary values are: 1, 2, 99, 100.
- Consider a program with two input variables x and y. These input variables have specified boundaries as :

$$a \le x \le b$$

$$c \le y \le d$$

- Both the inputs x and y are bounded by two intervals [a,b] and [c,d] respectively. For input x, we may design test cases with values a and b, just above a and also just below b. for input y, we may have values c and d, just above c and just below d. These test cases have more chance to detect an error. Like :



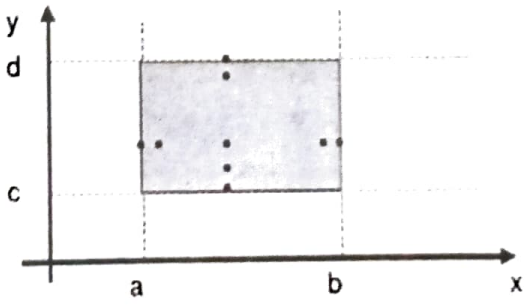
"Each dot represents a test case and inner rectangle is the input domain."

- Basic idea of BVA is to use input variables values at their minimum, just above minimum, a nominal value, just below maximum and at their maximum.
- BVA test cases are obtained by holding the values of all but one variable at their nominal value.

Example :

- The test cases for BVA in a program with two input variables x and y that may have any value from 100 to 300 are :

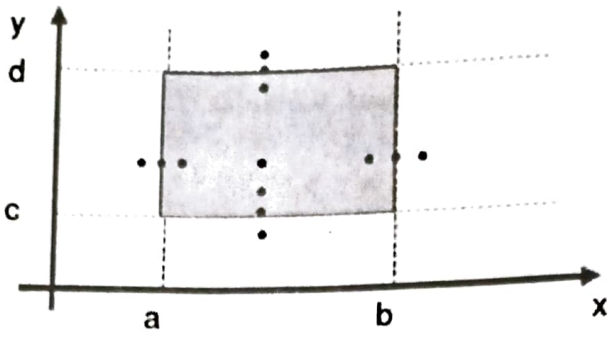
  (200,100), (200,101), (200,200), (200,299), (200,300), (100,200), (101,200), (299,200) and (300,200). (Given in below figure)



"Thus for a program of n variables, BVA yields 4n + 1 test cases."

Robust testing

- It is an extension of boundary value analysis.
- In this type of testing, the extreme values are exceeded with a slightly greater than the maximum and a value slightly less than the minimum. This is shown in below figure.



- For this technique, if a program of n variables, then BVA yields 6n + 1 test cases.

  Test cases are: (200,99), (200,100), (200,101), (200,200), (200,299), (200,300), (200,301), (99,200), (100,200), (101,200), (299,200), (300,200), (301,200).

➥ **Advantages of black box testing**

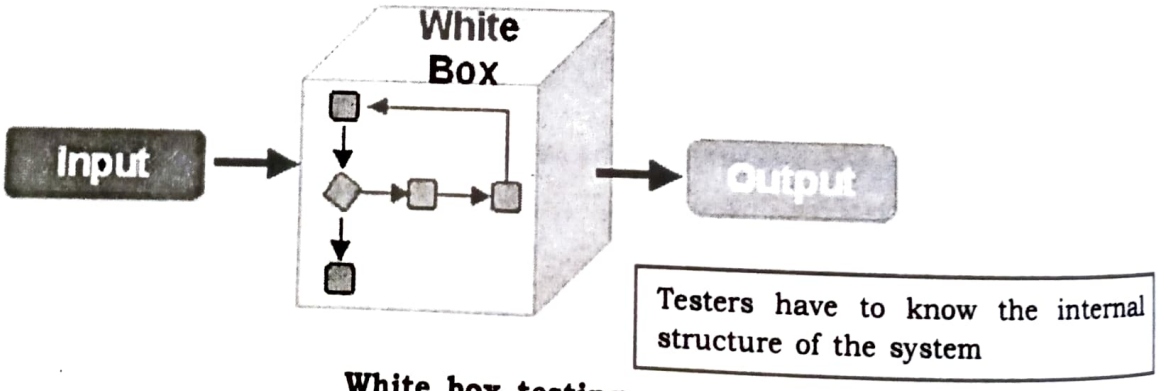
- It is Efficient for large code segment.
- Tester doesn't need to know the internal structure of the system.
- Tester perception is very simple.
- Programmer and tester are independent of each other.
- Quicker test case development.

➥ **Disadvantages of black box testing**

- It is inefficient testing.
- Without clear specification test cases are difficult to design.
- Only a selected number of test scenarios are actually performed. As a result, there is only limited coverage.

*"Black box testing is also known as **close box testing** and **opaque testing**."*

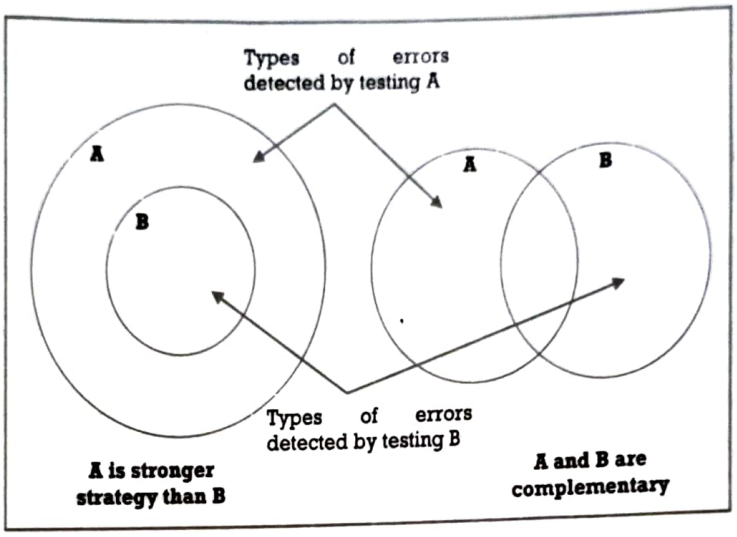
❖ **White box testing**



White box testing

- This method is concerned with testing the implementation of the program.
- The aim of this testing is to investigate the internal logic and structure of the code. That is why white box testing is also called structural testing.
- In white box testing it is necessary for a tester to have full knowledge of source code.
- Some of the synonyms of white box testing are *glass box testing, clear box testing, open box testing, transparent box testing, structural testing, logic driven testing* and *design based testing*.

- There are six basic types of testing: unit, integration, function/system, acceptance, regression,and beta. White-box testing is used for three of these six types: Unit, integration and regression testing.
- There are many white box strategies available, in which one is *stronger* than another. When two testing strategies detect errors that are different, then they are called *complementary*.
- The concepts of stronger and complementary testing are schematically illustrated in given figure.



- If we test program while it is running, it is called dynamic white box testing, and if we test the program without running it, only by examining and reviewing it then it is called static white box testing.
- Test cases are designed in this method are based on program structure or logic. Example of white box testing is circuit testing. As in electric circuit testing, the internal structure is checked.
- Test cases generated using white box testing can:
  - Guarantee that all independent paths within a module have been exercised at least once.
  - Exercise all decisions whether they are true or false.
  - Exercise internal data structure of the program.
- The different methods of white box testing are illustrated in below figure.

White Box Testing →
- Statement coverage
- Branch coverage
- Condition coverage
- Path coverage
- Data flow based testing
- Mutation testing

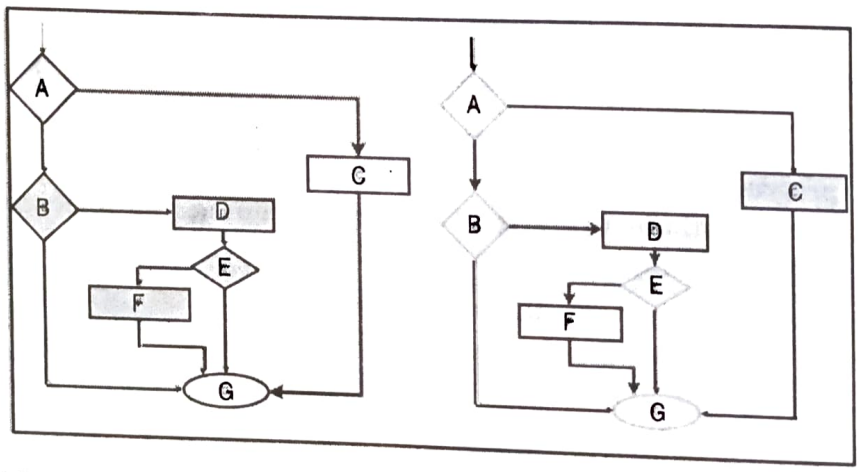
Statement coverage

- The statement coverage is also known as line coverage or segment coverage.
- The principal idea behind the statement coverage strategy is that unless a statement is executed, it is very hard to determine if an error exists in that statement.
- It aims to design test cases so that every statement in a program is executed at least once.
- Statement coverage is also known as *node testing.*
- However, executing some statement once and observing that it behaves properly for that input value is no guarantee that it will behave correctly for all other input values.

Example :

```
int x, y;
if x > y
        printf("x is greater");
else
        printf("y is greater");
```

For this code test case should be:
{(5,4), (4,5)}



In above figure, we can see that all the nodes (A to G) have been tested by designing test cases in two different ways. All nodes are covered, that's why this strategy is called *node testing.*

Branch coverage

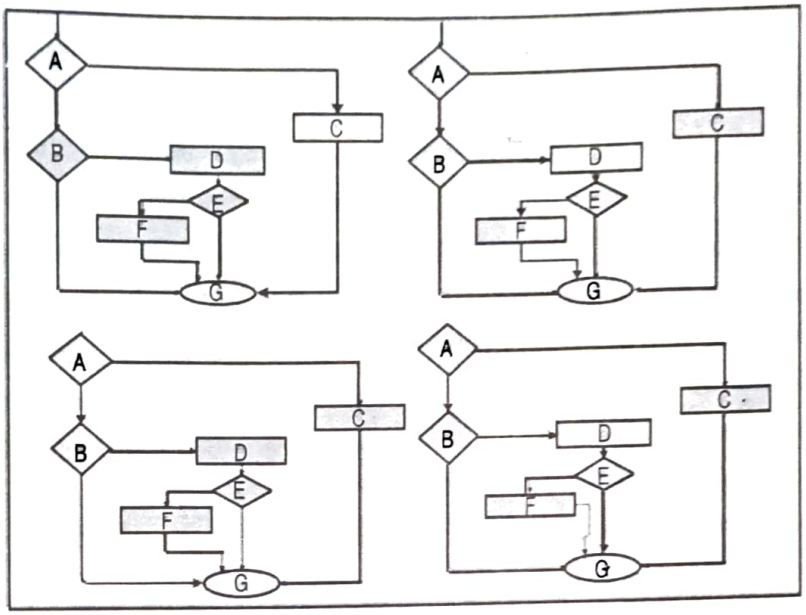
- In it, test cases are designed to make each branch condition to assume true and false values in turn.
- Branch testing is also known as *edge testing* as in it, each edge of a program's control flow graph is traversed at least once.
- Branch testing guarantees statement coverage, so it is stronger testing strategy than statement coverage.
- We can take above example for branch testing as well.

Example :

```
    intcompute_gcd(x, y)
    int x, y;
    {
1   while (x! = y){
2       if (x>y) then
3           x= x - y;
4       else y= y - x;
5   }
6    return x;
    }
```

For this code test case should be :

{(x=3,y=3), (x=4,y=3), (x=3,y=4)}



In above figure, we can see that test cases are designed in a way that every edge has been covered at least one. All the edges are covered that's why this testing strategy is called edge testing. At a particular time the edge which is examined is shown using bold arrow.

Note : *Using statement and branch coverage user generally attains 80-90% code coverage, which is sufficient.*

Condition coverage

- In this method test cases are designed to make each component of acomposite conditional expression to assume both true and false value.
- For example, consider the following code:

```
1. READ X, Y
2. IF(X == 0 || Y == 0)
3. PRINT '0'
```

In this example, there are two conditions → X = 0 and Y = 0. Now test cases should be designed in a way that all conditions get TRUE and FALSE.

Test case 1: (X=0, Y=5)

Test case 2: (X=5, Y=0)

- Condition testing is stronger than branch testing.
- For a composite conditional expression, if n components are there, for condition coverage → 2^n test cases are required.
- In condition coverage, test cases are increasing exponentially with the number of components. Therefore, a condition coverage based testing technique is practical only if n (the number of conditions) is small.

➥ **Path coverage**

- Path testing is used for module or unit testing.
- It requires complete knowledge of the program structure.
- The effectiveness of path testing deteriorates as the size of the software under test increases.
- It is not useful for system testing.
- This type of testing involves:
  - Generating a set of paths that will cover every branch in the program.
  - Finding a set of test cases that will execute every path in this set of program path.

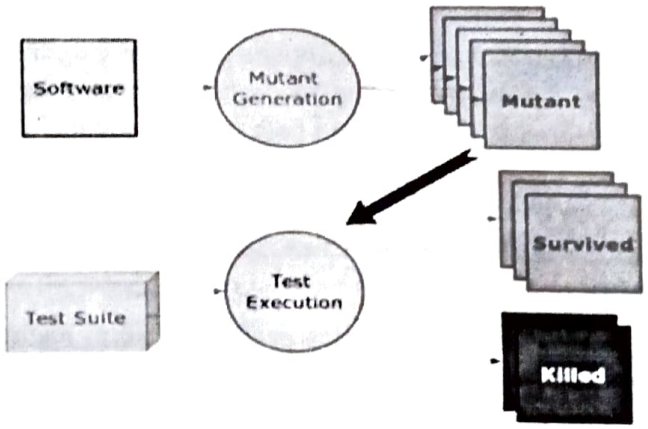
➥ **Data flow based testing**

- Data flow-based testing method selects test paths of a program according to thelocations of the definitions and uses of different variables in a program.
- For a statement numbered S, let

  DEF(S) = {X/statement S contains a definition of X}, and

  USES(S) = {X/statement S contains a use of X}
- Ex → For the statement S:**a=b+c;, DEF(S) = {a}. USES(S) = {b,c}**

➥ **Mutation testing**



Mutation Testing

- Mutation testing is a type of white box testing which is mainly used for unit testing.
- In it, software is first tested by initial test suit built from different white box techniques, and then mutation testing is taken up.
- Main idea behind this technique is → to make few changes to program at a time. In this type of software testing where we mutate (change) certain statements in the source code and check if the test cases are able to find the errors.

- Each time a program is changed, it is called mutated program and effectedchange is called a mutant.
- After comparison of the results of original and mutant programs, if the original program and mutant programs generate the same output, then that the mutant is killed by the test case. Hence the test case is good enough to detect the change between the original and the mutant program.If the original program and mutant program generate different output, Mutant is kept alive. In such cases, more effective test cases need to be created that kill all mutants.
- Major disadvantageof mutation testing is → it is very expensive to compute as large number of mutant can be generated.
- It is not suitable for manual testing.

➥ **Advantages of white box testing methods**

- Reveal hidden errors in the code.
- White box testing is very thorough as the entire code and structures are tested.
- Testing can start early in SDLC even if GUI is not available.
- Out product will be qualitative if white box testing is successful.

➥ **Disadvantages of white box testing methods**

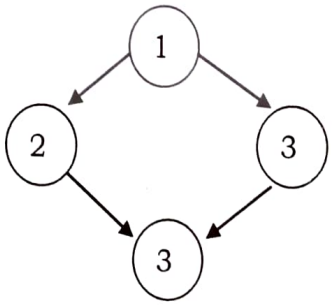
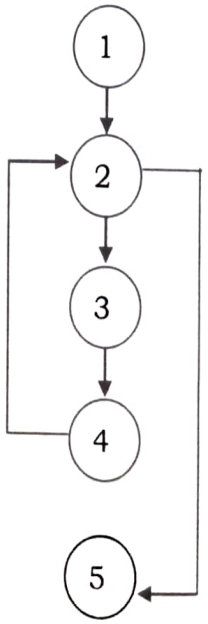
- White box testing can be quite complex and expensive.
- It is time consuming as it takes more time to test fully.
- It requires professional resources.
- In-depth knowledge about the programming language is necessary to perform white box testing.

❖ **Difference between Black box testing and White box testing.**

Black box testing	White box testing
→ Synonyms of black box testing are functional testing, close box testing, data driven testing and opaque testing.	→ Synonyms of white box testing are structural testing, glass box testing, clear box testing, open box testing, logic driven testing.
→ No need to know internal structure of the system.	→ Internal structure of the system must be known.
→ It is concerned with results.	→ It is concerned with details and internal workings of the system.
→ Performed by end users and also by testers and developers.	→ Normally done by testers and developers.
→ Granularity is low.	→ Granularity is low.
→ It is least exhaustive and time consuming.	→ Potentially most exhaustive and time consuming.
→ Not suited for algorithm testing.	→ It is suited for algorithm testing.
→ Example: search engine.	→ Example: electrical circuit testing.

❖ Control Flow Graph (CFG)(Extra topic)

- A control flow graph describes the sequence in which the different instructions of a program get executed.In other words, a control flow graph describes how thecontrol flows through the program.
- In order to draw the control flow graph of a program, all the statements of a program must be numbered first. The differentnumbered statements serve as nodes of the control flow graph.
- An edge from one node to another node exists if the execution of the statement representing the first node can result in the transfer of control to theother node.
- The CFG for any program can be easily drawn by knowing how torepresent the sequence, selection, and iteration type of statements in the CFG.
- Below figure summarizes how the CFG for these three types of statements can be drawn.

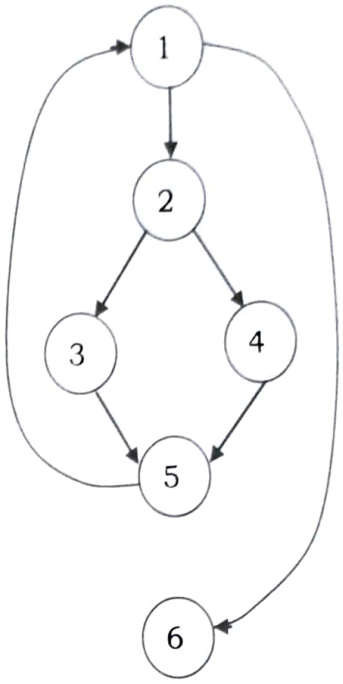
Sequence	Selection	Iteration
1. a=5 2. b=a*5	1. if (a>b) 2. print(a is greater) 3. else print(b is greater) 4. print(a+b)	1. a=5 2. while(a>0) { 3. print(a) 4. a=a-1 } 5. b=a;
		

- Using these basic concepts, the CFG of Euclid's GCD computationalgorithm can be drawn as shown.

GCD algorithm

```
1.    while (x != y)
2.    {     if (x > y) then
3.                x = x - y
4.          else  y = y - x
5.    }
6.    return (x)
```

CFG for GCD algorithm



CFG

❖ **Path**

- A path through a program is a node and edge sequence from the starting node to a terminal node of the control flow graph of a program. There can be more thanone terminal node in a program.
- Path coverage requires the coverage of linearly independent path.

❖ **Linearly independent path**

- Linearly independent path is defined as a path that has at least one new edge which has not been traversed before in any other paths.
- If a path has one new node compared to all other linearly independentpaths, thenthe path is also linearly independent.
- Sub path is not considered as a linearly independent path.

❖ **Cyclomatic complexity**

- McCabe's cyclomatic complexity defines an upper bound for the number of linearly independent paths through a program.
- Cyclomatic complexity is software metric used to measure the complexity of a program. This metric measures independent paths through program source code.
- It is very simple to compute.

Notes : Cyclomatic complexity is software metric used to measure the complexity of a program. This metric measures independent paths through program source code.

- It is practical way to determine maximum number of independent path in the program.

There are three different ways to compute the cyclomatic complexity.

➥ **Method 1:**

- Given a control flow graph G of a program, the cyclomatic complexity V(G) can be computed as :

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

(where N is the number of nodes of the control flow graph and E is the number of edges in the control flow graph.)

- In above figure 4.6cyclomatic complexity according to method 1 is:

$$V(G) = 7 - 6 + 2 = 3$$

➥ **Method 2:**

- Another way of computing the cyclomatic complexity V(G) is

V(G) = Total number of bounded areas + 1

any region enclosed by nodes and edges can be called as a bounded area.

- In above figure 4.6, we can see two bounded area. So cyclomaticcomplexity is:

$$V(G) = 2 + 1 = 3$$

➥ **Method 3 :**

- The cyclomatic complexity of a program can also be easily computed by computing the number of decision statements of the program.If N is the number of decision statement of a program, then the McCabe's metric is equal to N+1. In our example, we can clearly see there are two decision statements available (while loop and if statement). So the cyclomatic complexity is: 2 + 1 = 3.

ASSIGNMENT

MCQs

1 Open box testing is a synonym of which testing method ?
(a) unit testing (b) black box **(c) white box** (d) system testing

2 White Box techniques are also classified as :
(a) design based testing **(b) structural testing**
(c) error guessing technique (d) closed box testing

3 Which of the following testing is related to the boundary value analysis ?
(a) white box testing (b) red box testing
(c) unit testing **(d) black box testing**

4 First level of testing is ..... .
(a) white box testing (b) red box testing
(c) unit testing (d) black box testing

5 Data flow based testing is one of the techniques of ..... .
(a) white box testing (b) red box testing
(c) unit testing (d) black box testing

6 Code inspection technique of code review is led by ..... .
(a) reader (b) recorder **(c) moderator** (d) author

True / False

1. White box testing method is also called opaque testing. **Answer : False**
2. Result of the formal type of code review technique is documented. **Answer : True**
3. Validation is dynamic testing. **Answer : True**
4. Verification comes after validation. **Answer : False**
5. Structural testing is also called black box testing. **Answer : False**

Short Questions

1. What is code review ?
2. Give the differences between validation and verification.
3. What is validation ?
4. What is verification ?
5. Define unit testing.
6. In unit testing, _____ module plays the role of dummy module. **Answer : stub**
7. Define test case.
8. Match the following :

1. Testing in the small	a. Black box testing method
2. Dummy module	b. Verification
3. Search engine	c. Unit testing
4. Phase containment of errors	d. Stub

Answer : 1 – c, 2 – d, 3 – a, 4 – b

9. Define black box testing.
10. Give the definition of white box testing.
11. List various white box testing methods.
12. In a particular situation, if we provide input data as Boolean value in equivalence class partitioning method. How many valid and invalid classes can be generated ?

    Answer : Two classes (one valid and one invalid)
13. Justify the need of boundary value analysis.
14. In BVA (Boundary Value Analysis), if a program has n variables then how many test cases yields ? **Answer : 4n + 1**

Descriptive Questions

1. Explain code walkthrough.
2. Write a short note on code inspection.
3. Write a short note on unit testing.
4. Compare (OR differentiate) white box and black box testing methods.
5. Write a short note on equivalent class partitioning.
6. Explain in brief white box testing methods. (individual can be asked)
7. Write a short note on black box testing.

* * *

MODEL QUESTION PAPER - 1

Instructions :

1. Attempt all questions.
2. Make suitable assumptions wherever necessary.
3. Figures to the right indicate full marks.

Q. 1. (a) List different umbrella activities. 03

(b) Match the following : 04

1.	Testing in the small	a.	Black box testing method
2.	Dummy module	b.	Verification
3.	Search engine	c.	Unit testing
4.	Phase containment of errors	d.	Stub

(c) Give the IEEE definition of software. Give two examples of system software and application software. Explain different characteristics of a good software. 07

OR

(c) Define cohesion. Explain classification of cohesion. 07

Q. 2. (a) Differentiate between functional and non-functional requirements. 03

(b) Draw and explain software engineering layered approach. 04

(c) Explain different components of activity diagram. 07

OR

Q. 2. (a) Give the full form of : FTR, SQA, TQM 03

(b) State true / false for the following.

(i) PERT chart is a special type of bar chart used for resource allocation.

(ii) 'wishful thinking' is one of the characteristics of a bad SRS.

(iii) Structural testing is also called black box testing.

(iv) Iterative waterfall model is based on agile principles.

(c) Write a short note on data dictionary. 04

Q. 3. (a) Explain feasibility study of software development. 07

(b) Define UI. Explain CLI. 03

(c) Explain with example: how to find critical path for a given activity network diagram. 04

OR

Q. 3. (a) Explain agile methodology. 07

(b) Explain requirement gathering activities. 03

(c) Draw activity diagram for the following. And find critical path for the same. 04

Activity	Description	Time	Immediate Predecessor
A	Build internal components	2	None
B	Modify roof and floor	3	None
C	Construct collection stack	2	A
D	Pour concrete and install frame	4	B
E	Build high-temperature burner	4	C
F	Install control system	3	C
G	Install air pollution device	5	D,E
H	Inspection and testing	2	F,G

Q. 4. (a) Discuss job responsibilities of software project manager. **03**

(b) Explain code inspection. **04**

(c) Draw and explain spiral model with advantages and disadvantages. **07**

OR

Q. 4. (a) Explain path statement coverage. **03**

(b) Write a short note on scrum model. **04**

(c) Draw the figure of waterfall model and explain requirement analysis & specification and design phase. **07**

Q. 5. (a) Give the full form of: WBS, PMC, PERT **03**

(b) Write a short note GUI. **04**

(c) Define coupling. Explain classification of coupling. **07**

OR

Q. 5. (a) Explain activity network diagram. **03**

(b) Give one word answer for the following.

(i) Lowest cohesion is

(ii) Highest coupling is

(iii) The slack time of a critical activity is

(iv) The phase which consumes maximum time in waterfall model: **04**

(c) Explain risk management. **07**

❁ ❁ ❁

MODEL QUESTION PAPER - 2

Q. 1 Answer any seven out of ten. **14**

1. Define software and software engineering.
2. State in which situation we can use incremental model.
3. Write any two advantages and two disadvantages of scrum model.
4. List various SRS users.
5. Define scheduling.
6. How agile principles are important.
7. Define cohesion. List classification of cohesion.
8. Give the full form of: GUI, UML, CLI, ERD
9. Define black box testing. Give name of any two techniques of it.
10. In BVA (Boundary Value Analysis), if a program has n variables then how many test cases yields ?

Q. 2. (a) State true or false for the following.

(i) Waterfall model is meta model.

(ii) Activity network diagram used to determine critical path.

(iii) White box testing method is also called opaque testing. 03

OR

(a) Give one word answer for the following : **03**

(i) ..... chart is a special type of bar chart used in resource allocation.

(ii) In scrum, the cycle or stage of development is known as ..... .

(iii) In a particular situation, if we provide input data as Boolean value in equivalence class partitioning method. How many valid and invalid classes can be generated?

(b) Write a short note on menu based UI. 03

OR

(b) Write a short note on manipulative UI. 03

(c) Write a short note on web based application. 04

OR

(c) Differentiate between cohesion and coupling. 04

(d) Write a short note on prototype model. 04

OR

(d) Explain DFD. 04

Q. 3. (a) Explain code inspection technique. 03

OR

(a) Discuss agile principles. 03

(b) Explain software process framework activities.. 03

OR

(b) Write a short note on design activities. 03

(c) Write a short note on software engineering layered approach. 04

OR

(c) Write a short note on design methodology. 04

(d) Write a short note on data dictionary. 04

OR

(d) Explain boundary values analysis technique. 04

Q. 4. (a) Write a short note on requirement analysis. 03

OR

(a) Explain RAD model with figure. 03

(b) Define coupling. Explain any two coupling classification. 04

OR

(b) List and explain various components of activity diagram. 04

(c) Explain risk management in software engineering. 07

Q. 5. (a) Justify why spiral model is also called meta model. 04

(b) Explain GUI in brief. 04

(c) Explain PERT. 03

(d) Draw use case diagram for ATM system. 03

❁ ❁ ❁