Unit : 4

Software Coding and Testing

Prepared By: Mr. U. N. Parmar

Lecturer, CUSP, Surendranagar

4.1 Code review

- **Coding** means set of guidelines for a specific programming language. That creates applications, software and websites.
- **Coding** → telling a computer what to do (typing step by step commands).
- **Coding objectives** → transform the design document into high level language code and unit test this code.
- Input is → design document.
- **Coding standards** → well defined and standard style of coding.
- Purpose of coding standards:
  - *Uniform appearance.*
  - *Enhance understanding.*
  - *Encourage programming practices.*
- Coding standards list several rules.
- Good s/w development organizations develop their own coding standards and guidelines.

4.1 Code review

❖ Coding standards

- Software coding standards are language-specific programming rules that greatly reduce the probability of introducing errors into your applications.
- s/w development models are not considered into action.
- The use of global should be limited.
- Contents of the header.
- Naming conventions for variables.
- Error return conventions and exception handling mechanism.

❖ Coding guidelines

- Don’t use too clever or too difficult coding style.
- Do not use an identifier for multiple purposes.
- Proper variable use.

4.1 Code review

- The code should be well-documented.
- The length of any function should not exceed 10 source lines.
- Do not use goto statements.

➥ **Characteristics of coding:**

- Simplicity
- Readability
- Good documentation
- Transportability
- Usability

❖ Code Review

- Is a kind of verification.
- Code review for a model is carried out after the module is successfully compiled and the all the syntax errors have been eliminated.

4.1 Code review

- **IEEE standards, a review is →** “a process or meeting during which a work product or a set of work products is presented to project personnel, managers, users, customers or interested parties for comments or approval.”
- It is extremely cost effective process to reduce errors and produce high quality code.

↪ Advantages of code review

- Improves product quality.
- It helps us in improving domain expertise.
- We can have all the executions of path.

↪ Classification of review

- ***Formal reviews:*** conducted at the end of life cycle. Results of the formal reviews are documented.

4.1 Code review

- ***Informal reviews:*** are conducted on as-needed basis. It may be held at any time without any agenda. Results of the informal reviews are not documented.

➥ Techniques of code review

- *Two main techniques: Code walkthrough and code inspection.*

❑ Code walkthrough

- Is an **informal** analysis technique.
- Is used to assess and **improve the quality** of the software products.
- A Code Walkthrough is an informal meeting where the programmer leads the review team through his/her code and the reviewers try to identify faults.
- Process of code walkthrough.

4.1 Code review

- Members note down their findings and discuss them in the meeting.
- Several guidelines are produced in the meetings and accepted as examples.
- Several guidelines:
  - ✓ The **team** performing code walk through should not be either too big or too small
  - ✓ Discussion should **focus on discovery of errors** and not on how to fix the discovered errors.

⤷ **Advantages of code walkthrough**

- Useful for non software discipline people.
- It improves project team communication and morale.
- Provide educational medium for new team members.
- It can save project time and improve quality.

⤷ **Disadvantage:** it takes more time.

4.1 Code review

❑ Code inspection

- Code inspection is a formal, efficient and economical method of finding faults in design and code proposed by Fagan [1976].
- **Goal** → to identify and remove bugs before testing the code and to discover the algorithmic and logical errors.
- In it, the code is examined for the presence of certain kinds of errors.
- Coding standards are also checked during code inspection.
- An inspection team consists of four persons who play the role of moderator, reader, recorder and author.

➥ **List of some general programming errors:**

- – Jumps into loop *(in case of nested loop)*
- – Non terminating loops.

4.1 Code review

- – Array includes out of bound.
- – Improper storage allocation.
- – Use of uninitialized variables.
- – Mismatch between actual and formal parameters.

↪ **Advantages of code inspection**

- It makes software code maintainable and less costly.
- A detailed error feedback is provided.
- 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.

4.2 Software Documentation

- It is an important aspect. It could be **paper or electronic**.
- Software documentation can be defined as → *an artefact whose purpose is to communicate information about the software system to which it belongs.*
- It can work as an **information repository**.
- It could be the part of the software **(internal)** and it can be available offline **(external)**.
- **Kinds of software documents:** SRS, users' manual design documents, test documents, installation manual etc.
- **Two main requirements** for good documentation are that → it is complete and up-to-date.

4.2 Software Documentation

↪ **Advantages of good software documentation**

- Good documents enhance understandability and maintainability.
- Reduce effort and time for maintenance.
- It helps the users in effectively using the system.
- It helps in handling manpower turn over.
- It helps the manager effectively track the progress of the system.

○ **Classification of software documentation:**

❑ **Internal documentation**

- It’s a **part of the source code** itself.
- It is provided through appropriate module **headers and comments** embedded in the source code.
- **Main objective** → is to provide help to the user and the programmer to get a quick understanding of the program and the problem to modify the program as early as possible.

4.2 Software Documentation

- Included in the syntax of programming language through variable names, function header, code structuring, code indention, user defined functions etc.
- Internal documents not only explain the programs, or program statements, but also help programmers to know before any action is taken for modification.
- Good internal documentation appropriately formulating by coding standards and coding guidelines.

❑ External documents

- It is **outside the source code** through users' manual, SRS, design documents, test documents etc.
- It is **general description of the code** not concerned with detail, like algorithm, code dependencies, output format etc.
- External documents have **two types**: one for the users and one for those who wants to understand how the program works.

4.3 Testing

- The **basic goal** of any software development is to produce software that has no errors or has few errors.
- Testing is relied on **to detect the 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.***
- In testing, program is provided a set of test inputs (test cases).
- **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.

4.3 Testing

↪ **Some commonly used terms associated with testing are:**

- Error: *a kind of mistake (syntax or logical error).*
- Bug: *mistake done by programmer at the time of coding.*
- Fault: *it is representation of an error.*
- Failure: *occurs when fault executes, it is demonstration of an error.*
- Test case: *it is a Triplet [I,S,O].*
- Test suit: *set of all test cases.*

↪ Testing is four stage process:

Unit testing → subsystem testing (integration testing) → system testing → acceptance testing.

4.3 Testing

⤷ Difference between verification and 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.
- Thus **verification** is concerned with *phase containment of errors*, the aim of **validation** is that the *final product be error free*.
- **Verification** → are we doing right? And **Validation** → have we done right?

4.3 Testing

➥ Design of test cases

- Test cases must be of reasonable size and should cover as many errors.
- A large collection of test suit doesn't guarantee to cover all errors.
- Example of find greater number from X and Y.
- Systematic approach should be followed to design an optimal test suit.
- There are mainly two approaches to systematically design test cases:

I. **Black box testing** : 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*.

4.3 Testing

II. **White box testing:** 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 is first tested at unit level. This is testing in the small.
- Then integration testing is done, then finally system testing is executed.
- Integration and system testing are known as testing in the large.

↪ **Levels of testing**

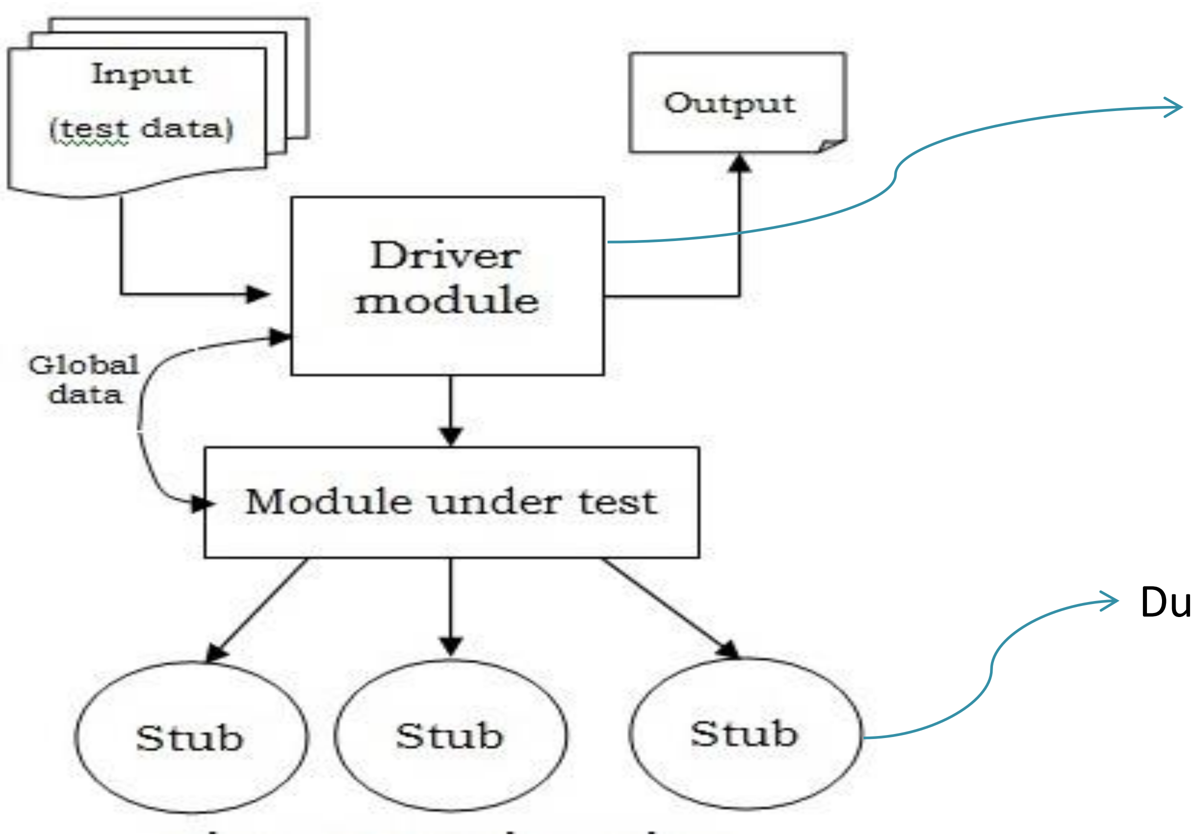
- Unit testing, Integration testing and System testing.
- Figure of above testing strategies.

4.3 Testing

❑ Unit testing

- Unit testing is the **first level of testing**.
- Running a program module as isolation.
- 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.
- A **complete environment is needed** to execute the unit testing on the module.
- The **calling module and called module** should be unit tested.
- Concept of **dummy module**.
- Due to this, unit testing often require **driver and/or stub modules**.

4.3 Testing



Calling module

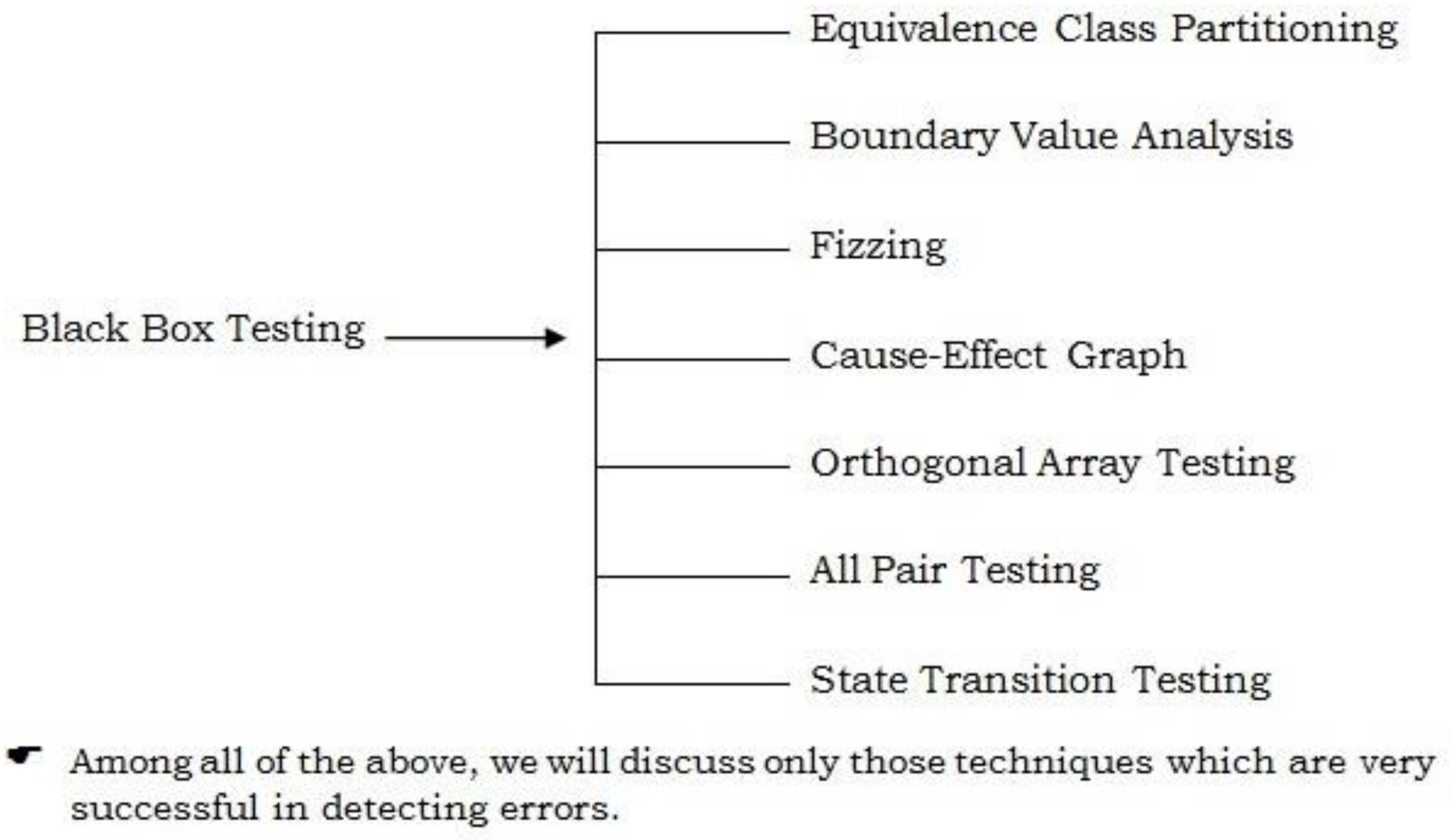
Dummy modules

4.3 Testing

❖ Black box testing

- This method is also called **behavioral testing or functional testing.**
- It is a technique of testing without having any knowledge of the internal working of the application.
- Figure of black box testing.
- Concept of black box (without any knowledge of internal working and it only examines the fundamental aspects) and a tester must know the system architecture.
- Example is → search engine.
- There are number of techniques that can be used to design test cases in black box testing method, are:

4.3 Testing



Black Box Testing →

- Equivalence Class Partitioning
- Boundary Value Analysis
- Fizzing
- Cause-Effect Graph
- Orthogonal Array Testing
- All Pair Testing
- State Transition Testing

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

4.3 Testing

❑ 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 behavior of the program is similar for every input data belonging to the same equivalence class.
- **Main idea →** if one test case in a class detects an error all other test cases in the class would be expected to find same error.
- **Two steps are required to implement this technique.**

I. Equivalence classes are identified by partition data into valid and invalid class.

II. Generate test cases using equivalent classes.

- Figure of equivalence partitioning.
- **Aim →** is to choose at least one element from each equivalence class.

4.3 Testing

➥ Some general guidelines for designing the equivalence classes:

- Input data values specified **in range** → one valid and two invalid classes.
- If input data from a set of **discrete number** → one for valid and one for invalid classes.
- If input data value is **Boolean** → one valid and one invalid classes.
- Example → accept any number between 1 to 99.

❑ 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.

4.3 Testing

- For this reason, boundary value analysis technique has been developed.
- In it, selection of test cases performed at the edges of the class.
- A simple example of X and Y with figure.
- 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.
- ***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. (with figure)
- ***For this technique, if n variables, then BVA yields 6n + 1 test cases.***

4.3 Testing

⤷ **Advantages of Black box testing**

- It is Efficient for large code.
- Tester perception is very simple.
- Programmer and tester are independent of each other.
- Quicker test case development.

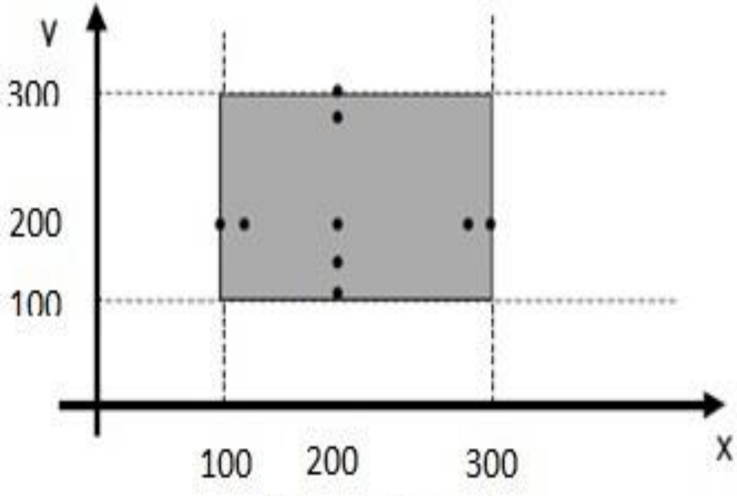
⤷ **Disadvantages of Black box testing**

- It is inefficient testing.
- Without clear specification test cases are difficult to design.
- There is only limited coverage due to selected numbers of test scenario.

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

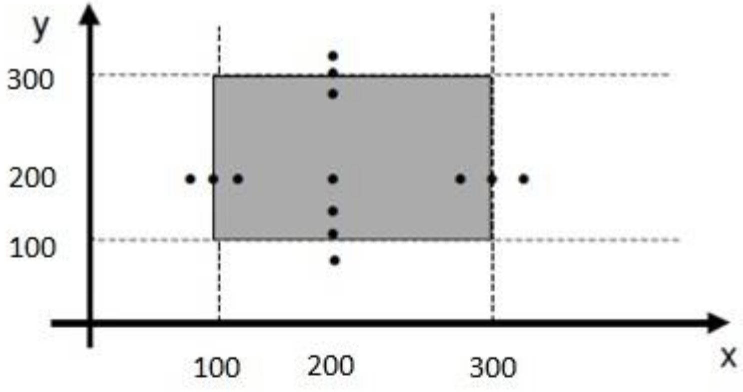
- **Example:**

- 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).



Input domain

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



☛ *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).

4.3 Testing

❖ White box testing

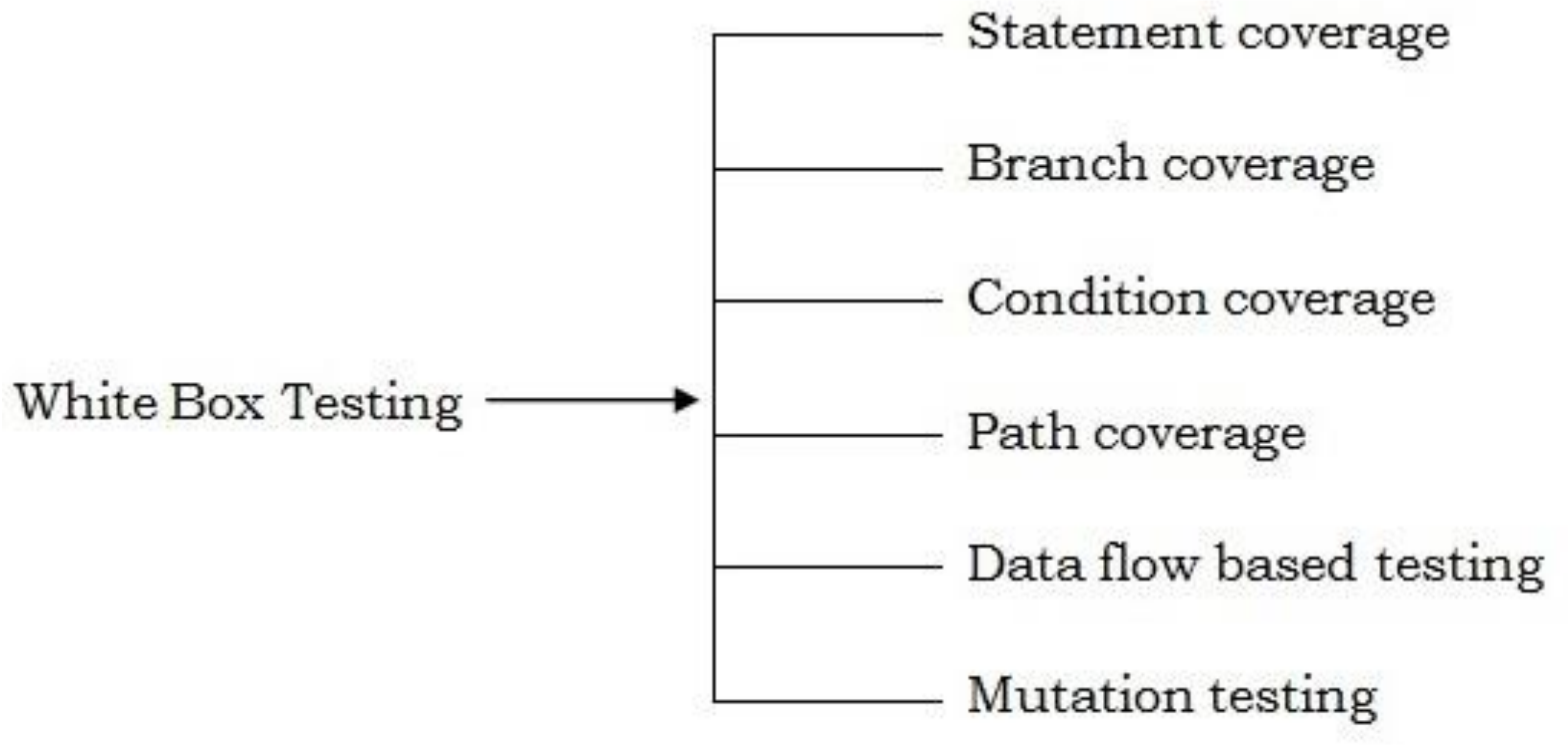
- This method is concerned with **testing the implementation** of the program.
- **The aim** of this testing is to investigate the internal logic and structure of the code. That is why white box testing is also called ***structural testing***.
- In white box testing it is necessary for a tester to have **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.

4.3 Testing

- Concept of stronger and complementary (with figure).
- Static and dynamic white box testing method.
- Test cases are based on program structure or logic.
- Example of white box testing is → circuit testing.
- 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.

4.3 Testing

The different methods of white box testing are:



White Box Testing

- Statement coverage
- Branch coverage
- Condition coverage
- Path coverage
- Data flow based testing
- Mutation testing

4.3 Testing

❑ Statement coverage

- Also known as **line coverage** or **segment coverage**.
- **Aim** → to design test cases so that every statement in a program is executed at least once.
- 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 input values.
- **Ex** → greater number of X and Y.

❑ 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. **Ex**→ GCD of X and Y.

4.3 Testing

❑ Condition coverage

- In this method test cases are designed to make each component of a composite conditional expression to assume both true and false value.
- For example, in the conditional expression ((c1.and.c2).or.c3), the components c1, c2 and c3 are each made to assume both true and false values.
- Condition testing is stronger than branch testing.
- If *n* components then 2^n test cases are required.
- In it, test cases are increasing with number of components, so it is practical if *n* is small.

❑ Path coverage

- Path testing is used for module or unit testing.
- It requires complete knowledge of the program structure.

4.3 Testing

- It is not useful for system testing.
- In this type of testing:
  - ✓ Generating a set of paths that will cover every branch
  - ✓ Finding a set of test cases that will execute every path

❑ Data flow based testing

- It selects test paths of a program according to the locations 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}**

4.3 Testing

❑ Mutation 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 →** to make few changes to program at a time.
- Each time a program is changed, it is called mutated program and change effected is called a mutant.
- If at least one test case in the test suit for which mutant gives an incorrect result, then the ***mutant is said to be dead***. If a mutant remains alive even after all the test, the ***test data kill the mutant***.
- **Major disadvantage →** very expensive when large number of mutant.
- It is not suitable for manual testing.

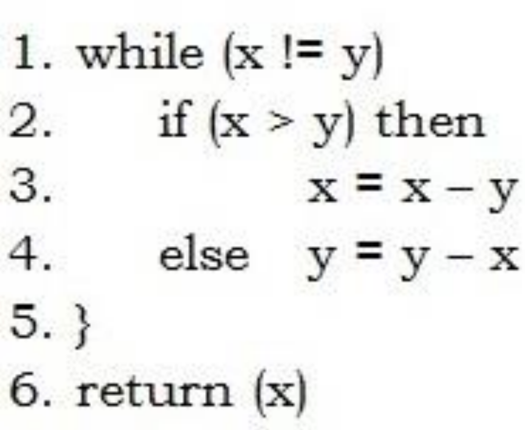
4.3 Testing

❖ Control Flow Graph (CFG)

- 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 the control flows through the program.*
- **To draw CFG** → all the statements of a program must be numbered first. They serve as nodes.
- 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 the other node.
- The CFG for any program can be easily drawn by knowing how to represent the sequence, selection, and iteration type of statements in the CFG.
- Summary of sequence, selection and iteration statements.

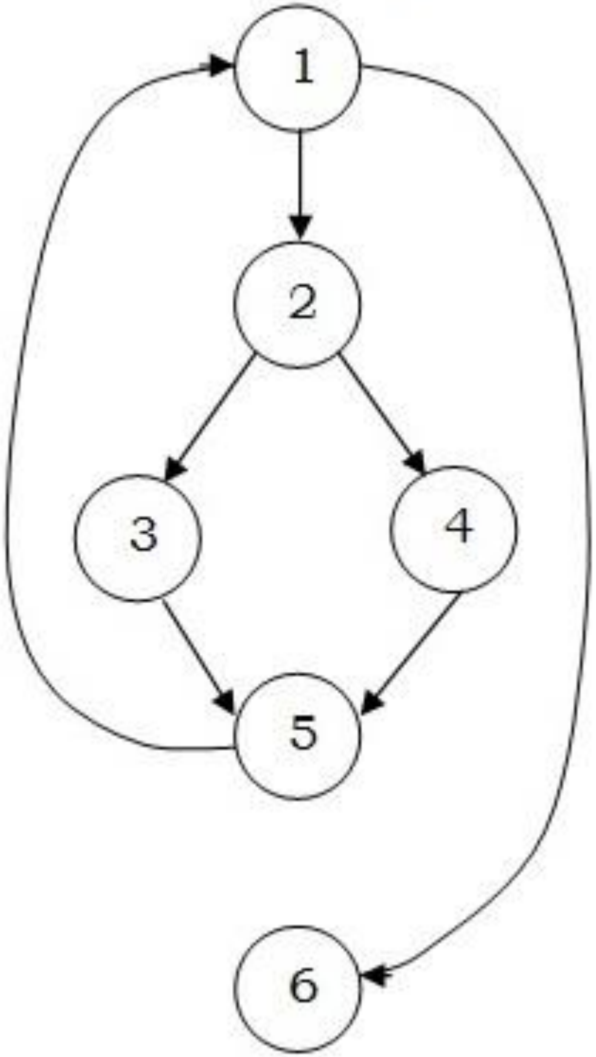
4.3 Testing

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



4.3 Testing

- **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 than one terminal node in a program.
- **Linearly independent path :** A linearly independent path is any path through the program that introduces at least one new edge that is not included in any other linearly independent paths.
- If a path has one new node compared to all other linearly independent paths, then the 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.
- It is very simple to compute.

4.3 Testing

- It defines maximum number of independent path in the program.

➥ There are three different methods 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

❑ ***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.

4.3 Testing

❑ *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.
- Difference between black box 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.

4.4 Test documentation

- The documentation which is generated towards the end of testing is the test summary reports.
- It provides summary of test suits which has been applied to the system.
- It specify how many test suits are successful, how many are unsuccessful and what is the degree of successful and unsuccessful.
- *A test design specification*
- *A test case specification*
- *A test procedure specification*
- *A test item transmittal report*
- *A test log*
- *A test incident report*
- *A test summary report*

Thank You...