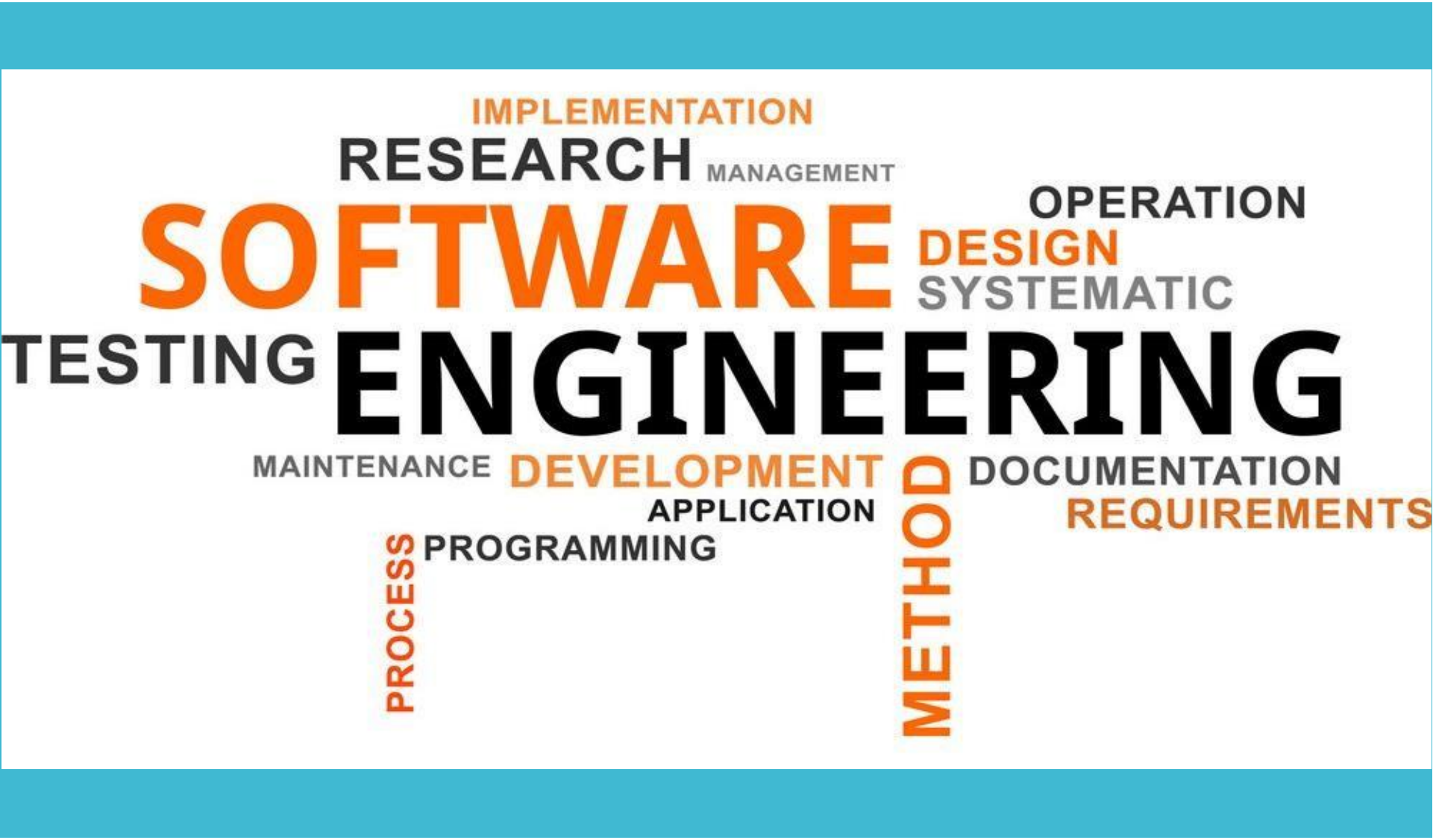


Marwadi University

Department of Information Communication and Technology

Chapter 4
Software Coding & Testing

Prof. Suhag Baldaniya

Contents

- Coding standards & Coding Guidelines
- Code Review
- Software Testing Techniques,
- Software Testing Fundamentals
- White Box Testing Techniques in detail
- Black Box Testing Techniques in detail

Coding Standard

- Good software development organizations normally require their programmers to adhere to some well-defined and standard style of coding called **coding standards.**
- Most software development organizations formulate their own coding standards that suit them most.

Coding Standard

- Purpose of following a standard style of coding is:
  - A coding standard gives a **uniform appearance** to the codes.
  - It enhances code **understanding**.
  - It encourages **good programming practices.**

Coding Standard

1. **Rules for limiting the use of global**
   - These rules list what types of data can be declared global and what cannot.
2. **Naming conventions for global & local variables & constant identifiers**
3. **Contents of the headers preceding codes for different modules**
   - The information contained in the headers of different modules should be standard for an organization.

Coding Standard

4. Error return conventions and exception handling mechanisms

- The way error conditions are reported in by a function program should be consistent with an organization standard or should be handled

Coding guidelines

- **Avoid using a coding style that is too difficult to understand:** Code should be easily understandable. The difficult coding makes maintenance and debugging expensive.
- **Avoid using an identifier for multiple purposes:** Each variable should be given a descriptive and meaningful name indicating the reason behind using it. This is not possible if an identifier is used for multiple purposes and thus it can lead to confusion to the reader. Moreover, it leads to more difficulty during future enhancements.
- **Code should be well documented:** The code should be properly commented for understanding easily. Comments regarding the statements understandability of the code..

Coding guidelines

- **Lengthy functions are very difficult to understand.** That's why functions should be small enough to carry out small work and lengthy functions should be broken into small ones for completing small tasks.
- **Do not use goto statements:** Use of GOTO makes the program unstructured, thus it reduces understandability of the program and also debugging becomes difficult

Coding guidelines

- **Advantages of Coding Guidelines:**
- Coding guidelines increase the efficiency of the software and reduces the development time.
- Coding guidelines help in detecting errors in the early phases, so it helps to reduce the extra cost incurred by the software project.
- If coding guidelines are maintained properly, then the software code increases readability and understandability thus it reduces the complexity of the code.
- It reduces the hidden cost for developing the software.

Code Review

- Code Review is carried out after successfully compiled and all the syntax errors have been eliminated.
- Code Reviews are extremely cost-effective strategies for reduction in coding errors and to produce high quality code.
- Two types of reviews are carried out on the code of a module.
  1. **Code Walk Through**
  2. **Code Inspection**

Code Walk Through

- Code Walk Through is an informal technique.
- A few members of the team are given the code before the walk through meeting to read and understand code.
- Each member selects some test cases and simulates execution of the code.
- The walk through meeting is conducted in the presence of the coder of the module is present.
- Objectives of the walk through are to **discover the algorithmic and logical errors**.

Code Inspection

- The aim of Code Inspection is to discover some common types of errors caused due to improper programming.
- In other words, during Code Inspection the code is examined for the presence of certain kinds of errors.
- For instance, consider the classical error of writing a procedure that modifies a parameter while the calling routine has a corresponding parameter that is also modified.
- It is more likely that such an error will be discovered by looking for these kinds of mistakes in the code.
- In addition, commitment to coding standards is also checked.

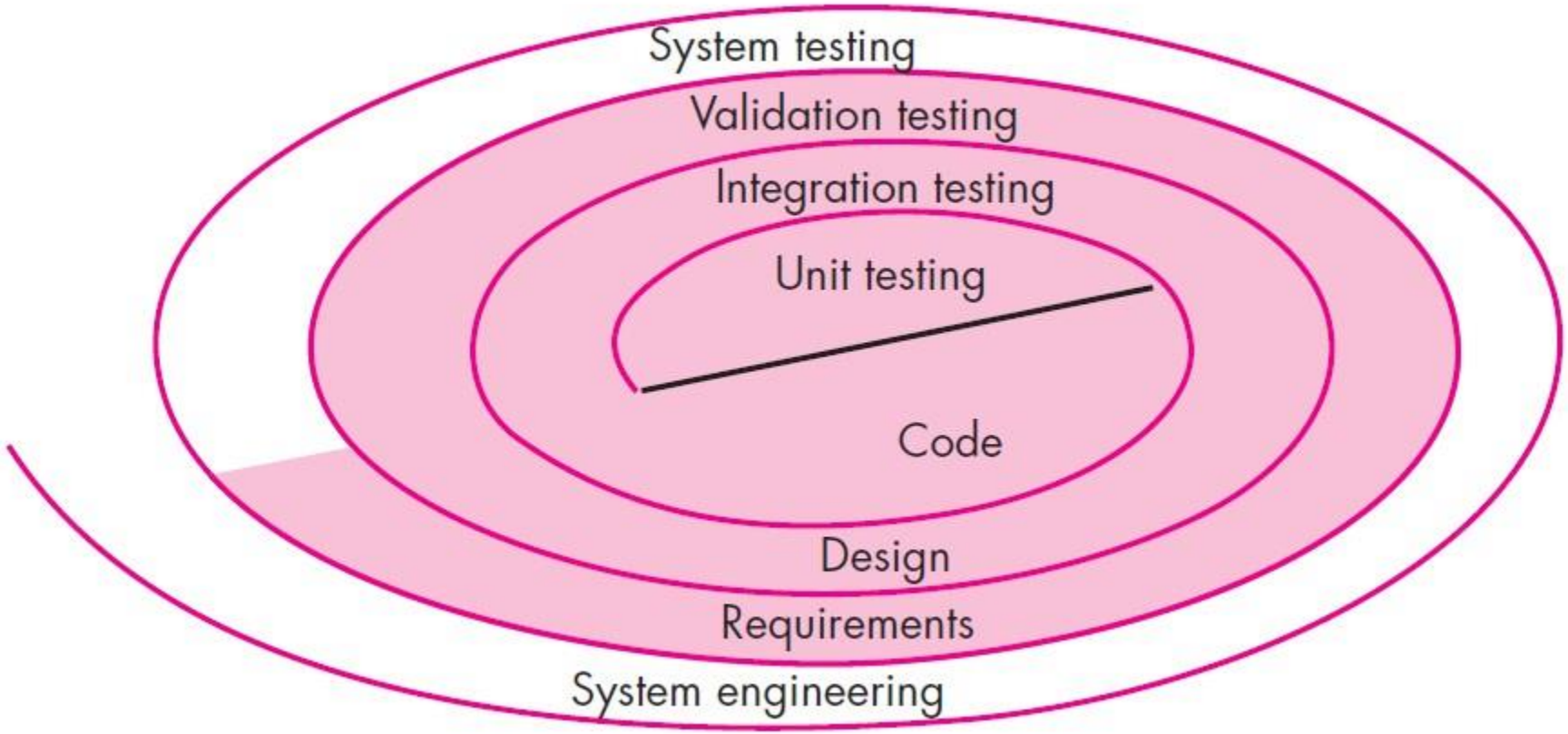
Software testing

- Softwaretesting is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation.

Testing Strategies

- It is a critical element of software quality assurance and represents the ultimate review of specification, design, and code generation.
- Usually a software development organization pays 30 to 40% of total project effort on testing.
- The engineer creates a series of test cases to “defeat” the software that has been built.
- In fact, testing is the one step in the software process that could be viewed as destructive rather than constructive.
- Destructive testing is testing method where application is intentionally made to fail to check the robustness of the application and identify the point of failure.

Testing Strategies



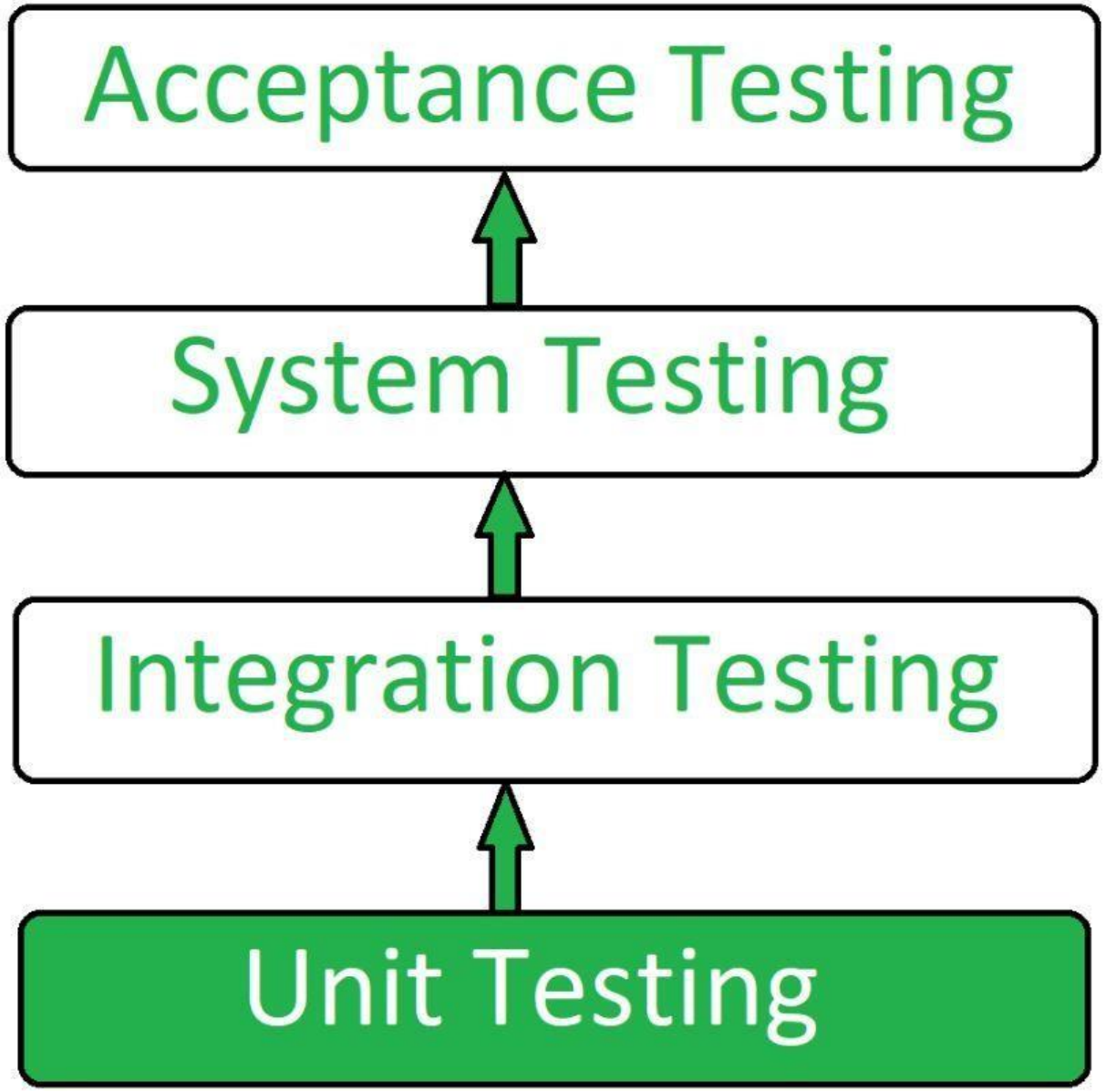
Testing Strategies

- Moving inward along the spiral, you come to design and finally to coding.
- To develop computer software, you spiral inward (counterclockwise) that decrease the level of abstraction on each turn.
- But to test computer software, you spiral outward in a clockwise direction along streamlines that increase the scope of testing with each turn.

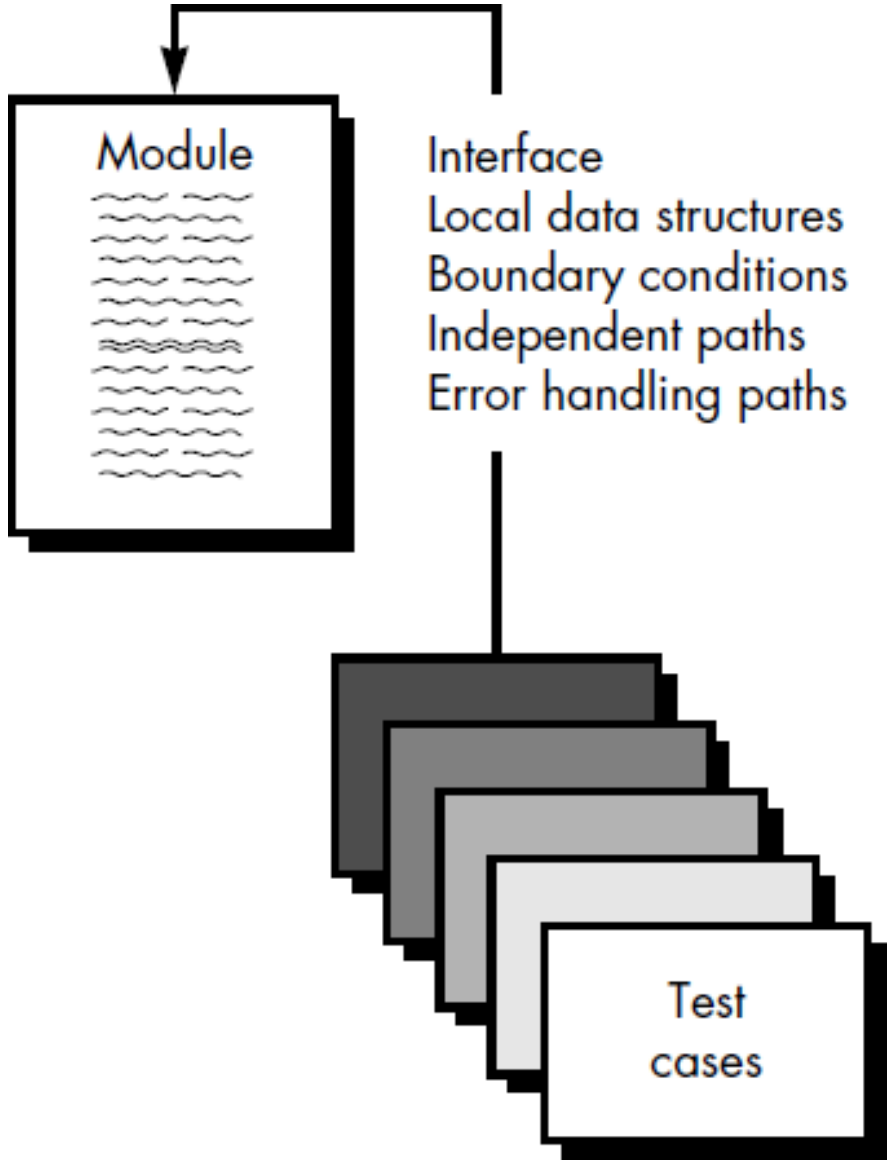
Test Strategies for Conventional Software

1. **Unit Testing**
2. **Integration testing**
   - Top-down Integration Testing
   - Bottom-up Integration Testing
3. **Regression Testing**
4. **Smoke Testing**
5. **Validation Testing**
   - Alpha Testing
   - Beta Testing
6. **System Testing**
   - Recovery testing
   - Security testing
   - Stress testing
   - Performance testing
   - Deployment testing
7. **Acceptance**

Test Strategies for Conventional Software- Levels of Testing



Unit Testing
Testing

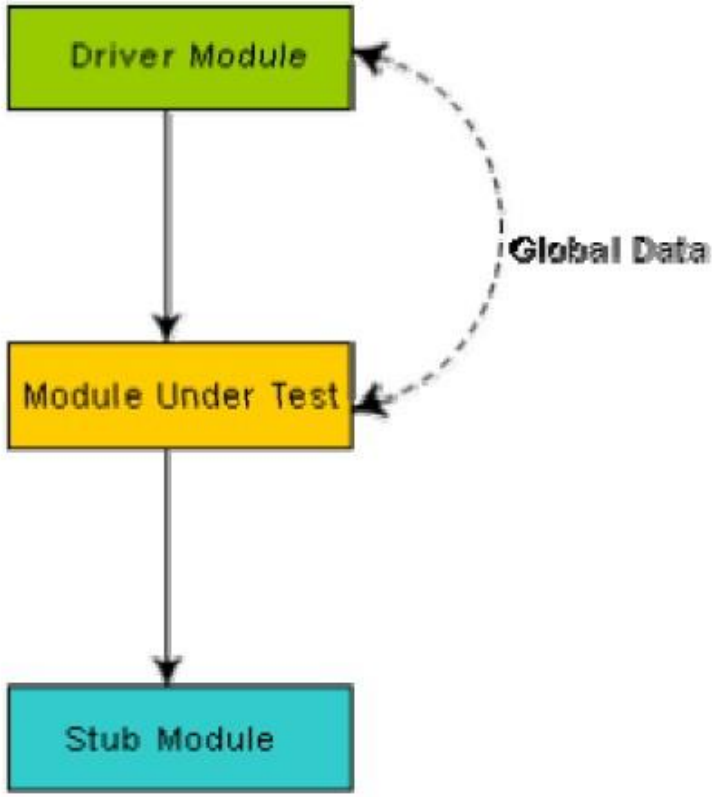


Unit Testing

- Unit is the smallest part of a software system which is testable.
  - It may include code files, classes and methods which can be tested individually for correctness.
- Unit Testing validates small building block of a system before testing an integrated large system or whole system.
- The module is tested to ensure that information properly flows into and out of the program unit
- Local data structures are examined to ensure that temporarily stored data maintains its integrity during execution
- All independent paths through the control structures are exercised to ensure that all statements in module have been executed at least once

Unit Testing

- They replace the modules that aren't yet been developed but are till needed in the testing of other modules functionality tested against expected results.
- Driver and/or Stub software must be developed for each unit test.
- A **driver** is nothing more than a "main program" that accepts test case data, passes such data to the component, and prints relevant results.
- Stub serves to replace such modules which are called by the component to be tested.
- A **stub** or "dummy subprogram" uses the subordinate module's



Unit Testing- Stubs Vs Drivers

Stubs	Drivers
Stubs are used in Top-Down Integration Testing.	Drivers are used in Bottom-Up Integration Testing.
Stubs are basically known as a “called programs” and are used in the Top-down integration testing.	While, drivers are the “calling program” and are used in bottom-up integration testing.
Stubs are similar to the modules of the software, that are under development process.	While drivers are used to invoking the component that needs to be tested.
Stubs are basically used in the unavailability of low-level modules.	While drivers are mainly used in place of high-level modules and in some situation as well as for low-level modules.
Stubs are taken into use to test the feature and functionality of the modules.	Whereas the drivers are used if the main module of the software isn’t developed for testing.
The stubs are taken into concern if testing of upper-levels of the modules are done and the lower-levels of the modules are under developing process.	The drivers are taken into concern if testing of lower-levels of the modules are done and the upper-levels of the modules are under developing process.
Stubs are used when lower-level of modules are missing or in a partially developed phase, and we want to test the main module.	Drivers are used when higher-level of modules are missing or in a partially developed phase, and we want to test the lower(sub)- module.

Integration Testing

- **INTEGRATION TESTING** is defined as a type of testing where software modules are integrated logically and tested as a group.
- Modules are typically; code modules, individual applications, client and server applications on a network, etc.
- Testing of the interactions between the interactions with other system externally is called **Integrated Testing.**
- **Types of integration testing are:**
  1. **Top-down Integration Testing**
  2. **Bottom-up Integration**

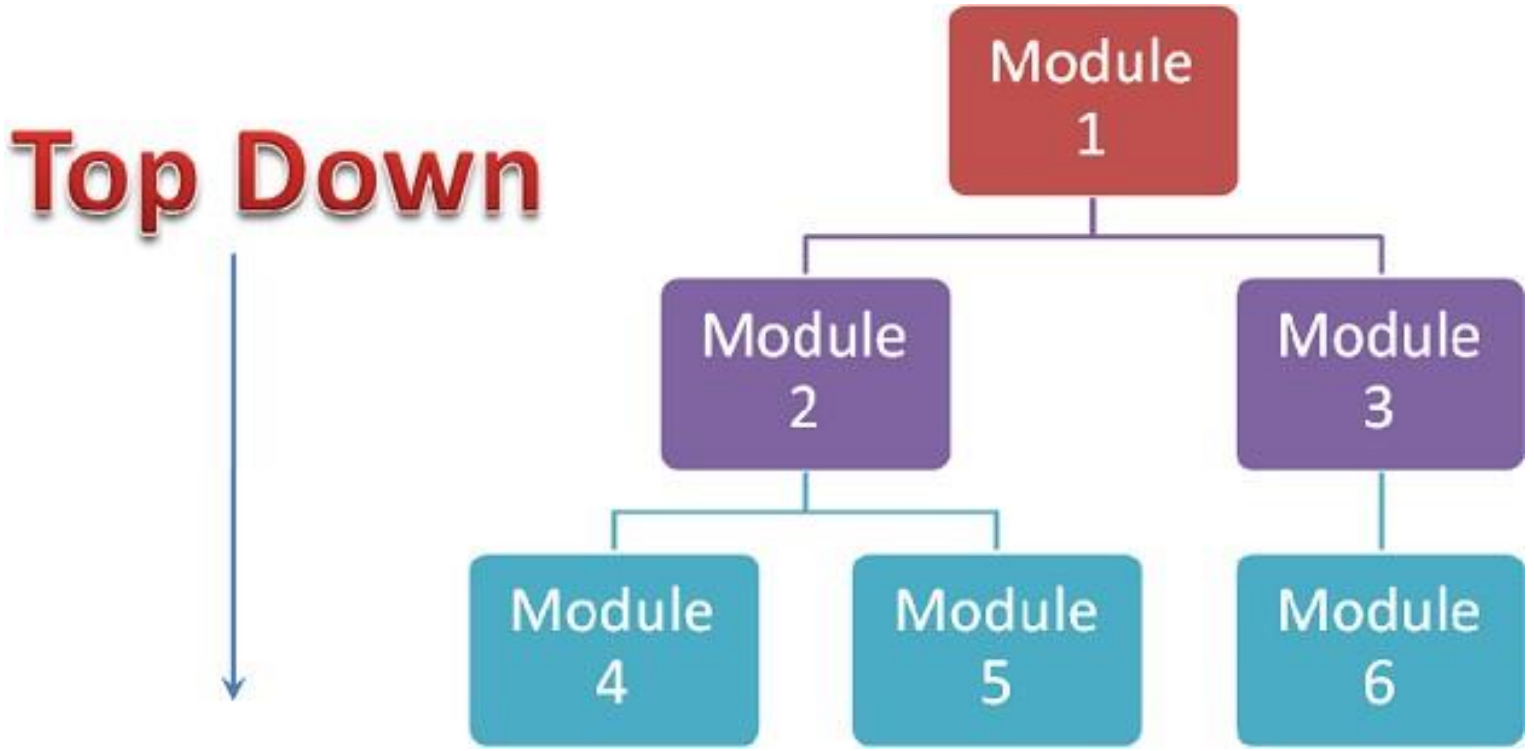
Types of Integration Testing

1. Top-down integration testing

Top Down Integration Testing is a method in which integration testing takes place from top to bottom following the control flow of software system.

- The higher level modules are tested first and then lower level modules are tested and integrated in order to check the software functionality
- Modules are integrated by moving downward beginning with the main program.

Types of Integration Testing

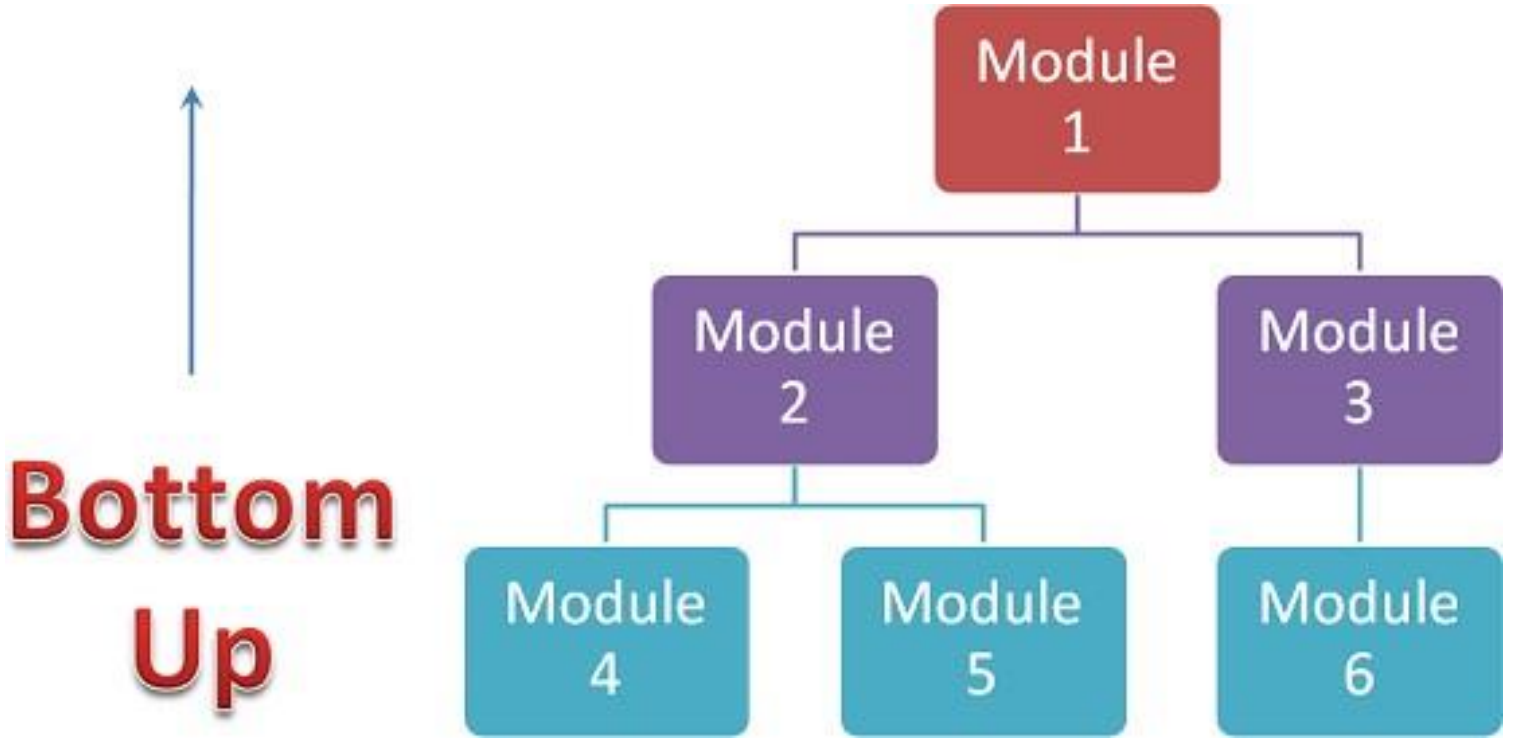


Types of Integration Testing

2. Bottom-up Integration Testing

- Bottom-up Integration Testing is a strategy in which the lower level modules are tested first.
- These tested modules are then further used to facilitate the testing of higher level modules.
- The process continues until all modules at top level are tested.
- Once the lower level modules are tested and integrated, then the next level of modules are formed.

Types of Integration Testing



Regression Testing

- Repeated testing of an already tested program, after modification, to discover any defects introduced as a result of the changes in the software being tested
- Regression testing is done by re-executing the tests against the modified application to evaluate whether the modified code breaks anything which was earlier working
- Anytime we modify an application, we should do regression testing
- It gives confidence to the developers that unexpected side effects after modification

Smoke Testing

- Smoke Testing is an integrated testing approach that is commonly used when product software is developed
- This test is performed after each Build Release
- Smoke testing verifies – Build Stability
- This testing is performed by “Tester” or “Developer”
- Smoke Testing is also referred as ‘Surface Level Testing’ as it takes place before actual process testing is performed does deep testing just verifies not functionalities are working fine or not.
- It also calls ‘Build Verification Testing’ as it verifies the initial builds of software.

Validation Testing

- The process of evaluating software to determine whether it satisfies specified business requirements (client's need).
- It provides final assurance that software meets all informational, functional, behavioral, and performance requirements.
- If software is developed as a product to be used by many customers, it is impractical to perform formal acceptance tests with each one.
- Most software product builders use a process called **alpha and beta testing** to detect errors that only the end user seems able to find.

Types of Validation Testing

Alpha Test

- The alpha test is conducted at the developer's site by a representative group of end users.
- The software is used in a natural setting with the developer "looking over the shoulder" of the users and recording errors and usage problems.
- Alpha tests are conducted in a controlled environment.

Beta Test

- The beta test is conducted at one or more end-user sites.
- Unlike alpha testing, the developer generally is not present.
- Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer.

Types of Validation Testing

Alpha Testing	Beta Testing
Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black box testing.
Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Alpha testing is performed at developer's site.	Beta testing is performed at end-user of the product.
Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.
Alpha testing may require long execution cycle.	Beta testing requires only a few weeks of execution.
Developers can immediately address	Most of the issues or feedback collected

System Testing

- In system testing, the software elements and other system elements are tested.
- System testing verifies that all elements mesh properly and overall system function/performance is achieved.
- System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system.

Types of System Testing

1. Recovery testing
2. Security testing
3. Stress testing
4. Performance testing
5. Deployment testing

Types of System Testing

1. Recovery testing

- It is a system test that forces the software to fail in a variety of ways and verifies that recovery is properly performed.
- If recovery is automatic (performed by the system itself)
  - Re-initialization, check pointing mechanisms, data recovery, and restart are evaluated for correctness.
- If recovery requires human intervention
  - The mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.

Types of System Testing

2. Security testing

- It attempts to verify software's protection mechanisms, which protect it from improper penetration (access).
- During this test, the tester plays the role of the individual who desires to penetrate the system.

3. Stress testing (Sensitivity Testing)

- It executes a system in a manner that demands resources in abnormal quantity, frequency or volume.

4. Performance testing

- It is designed to test the run-time performance of software.
- It occurs throughout all steps in the testing process, even at the unit testing level.

Types of System Testing

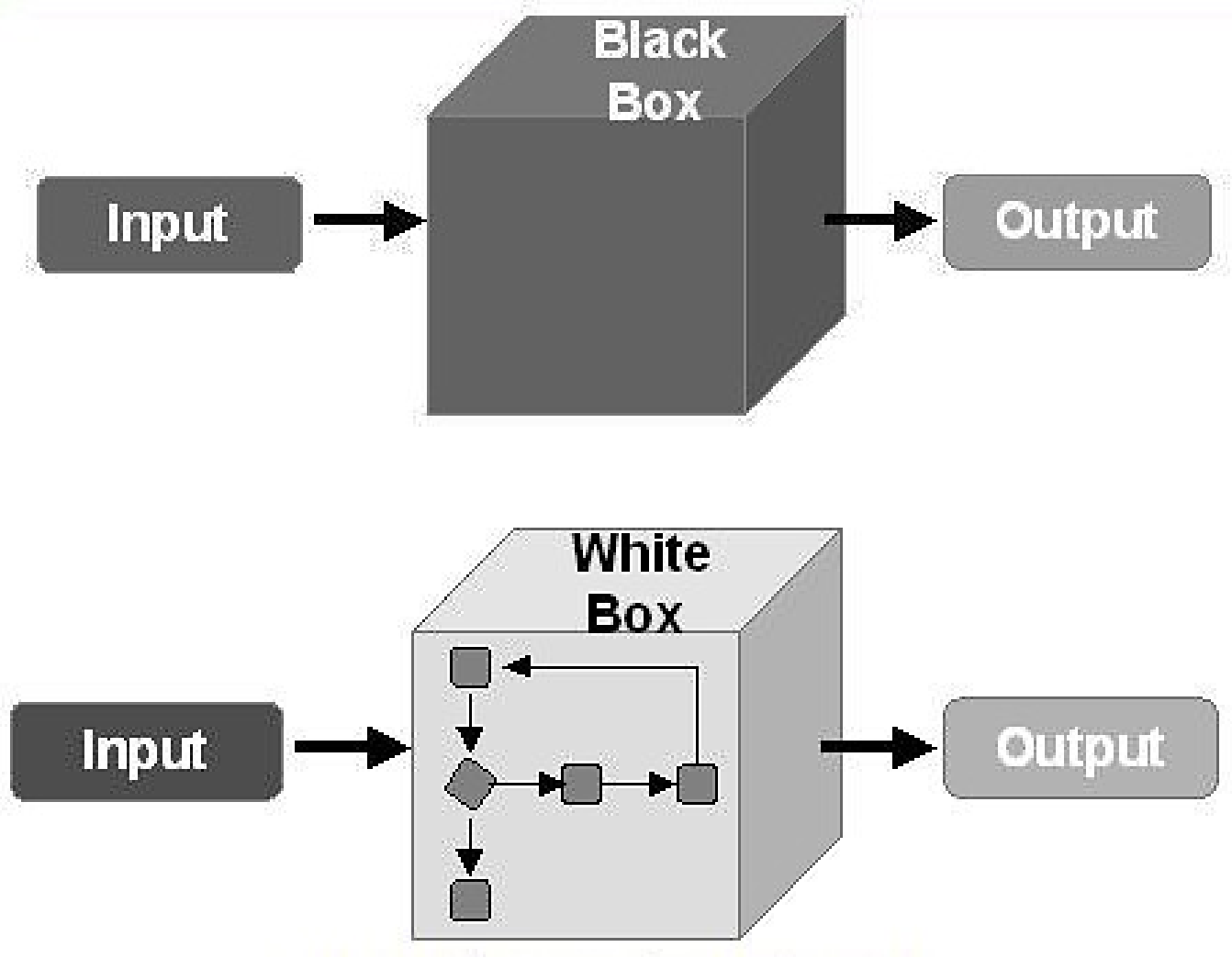
5. Deployment testing (Configuration Testing)

- It tests the software in each environment in which it is to operate.
- In addition, it examines
  - all installation procedures
  - specialized installation software that will be used by customers
  - all documentation that will be used to introduce the software to end users

Acceptance Testing

- It is a level of the software testing where a system is tested for acceptability.
- The purpose of this test is to evaluate the system's **compliance with the business requirements**.
- It is a formal testing conducted to determine whether or not a system satisfies the acceptance criteria with respect to user needs, requirements, and business processes
- It enables the customer to determine, whether or not to accept the system.
- It is performed after System Testing and before making the system available for actual use.

Black Box and White Box Testing



Black Box Testing (Behavioral Testing)

- The testing is done without the internal knowledge of the products.
- Tester has access only to running code and it is supposed to satisfy specification.
- Black Box Testing is a software testing method in which internal structure/design/implementation of the module being tested is not known to the tester.
- It attempts to find errors in the following categories:
  1. Incorrect or missing functions
  2. Interface errors
  3. Errors in data structures or external database access
  4. Behavior or performance errors
  5. Initialization and termination errors

Advantages of Black Box Testing (Behavioral Testing)

- Efficient when used on large systems.
- Since the tester and developer are independent of each other, testing is balanced and neutral. There is no need for the tester to have detailed functional knowledge of system.
- Tests will be done from an end user's point of view, because the end user should accept the system.
- Testing helps to identify contradictions in functional specifications.
- Test cases can be designed as soon as the functional specifications are complete
- Test cases can be developed in parallel with code

Disadvantages of Black Box Testing (Behavioral Testing)

- Only a small number of possible inputs can be tested and many program paths will be left untested.
- Test cases will be difficult to design without having clear specifications, which is the situation in many projects.

The special techniques are needed which **select test-cases smartly** from the **all combination of test-cases** in such a way that all scenarios are covered.

Black Box Testing Techniques

1. Equivalence Partitioning:

- Input data for a program unit usually falls into a number of partitions, e.g. all negative integers, zero, all positive numbers
- Each partition of input data makes the program behave in a similar way
- Example :Assume that we have to test field which accepts SPI (Semester Performance Index) as input (SPI range is 0 to 10)
- **Valid Class:** SPI = 0 – 10
- **Invalid Class 1:** SPI <=-1
- **Invalid Class 2:** SPI >=11

Black Box Testing Techniques

2. Boundary Value Analysis (BVA)- Range Testing

- It arises from the fact that most program fail at input boundaries
- Boundary Testing comes after the Equivalence Class Partitioning
- Suppose system asks for “a number between 100 and 999 inclusive”

Lower boundary Upper boundary

99 100 101 998 999

White box testing (Glass-box Testing)

- White Box Testing is a software testing method in which internal structure/design/implementation of the module being tested is known to the tester.
- Using white-box testing, we can derive test cases that guarantee that

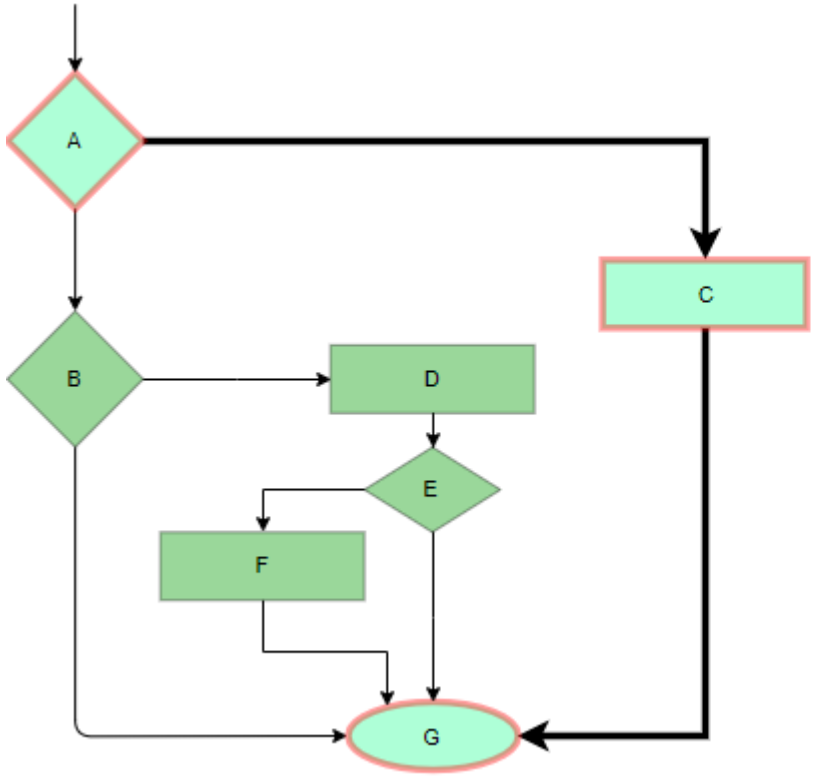
1. All independent paths within a module have been exercised at least once.
2. All logical decisions on their true and false sides have been exercised
3. All loops at their boundaries have been executed.
4. Internal data structures have been exercised to ensure their validity.

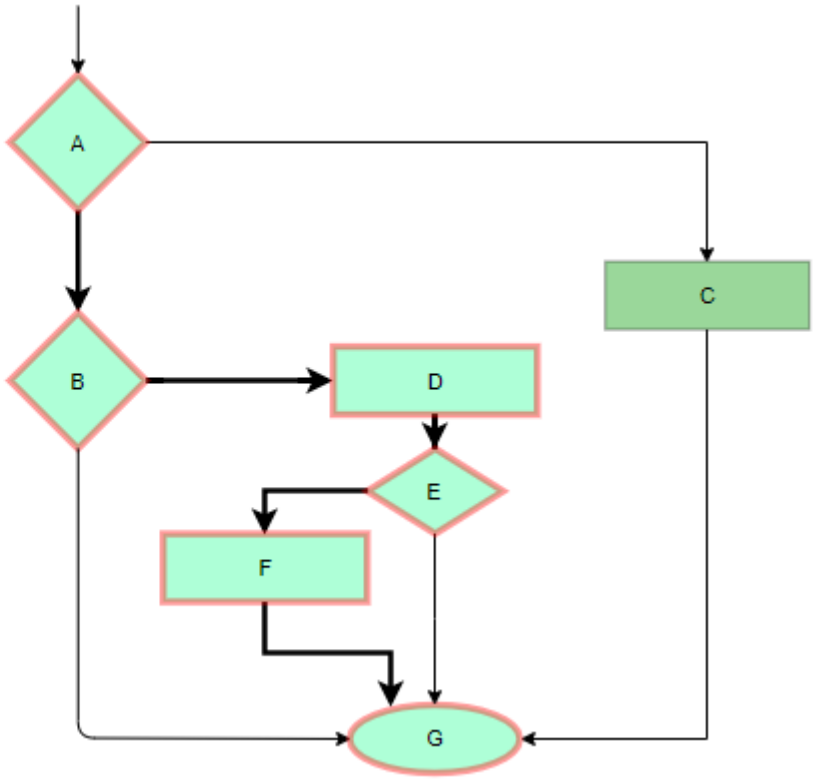
White box testing (Glass- box Testing)

- It is applicable to the following levels of software testing.
  - Unit Testing: For testing paths within a unit.
  - Integration Testing: For testing paths between units.
  - System Testing: For testing paths between subsystems.
- White-box testing strategies
  1. Statement coverage
  2. Branch coverage
  3. Path coverage(Flow graph, Cyclomatic Complexity and Graph Metrics are used to arrive at basis path)

White box testing techniques

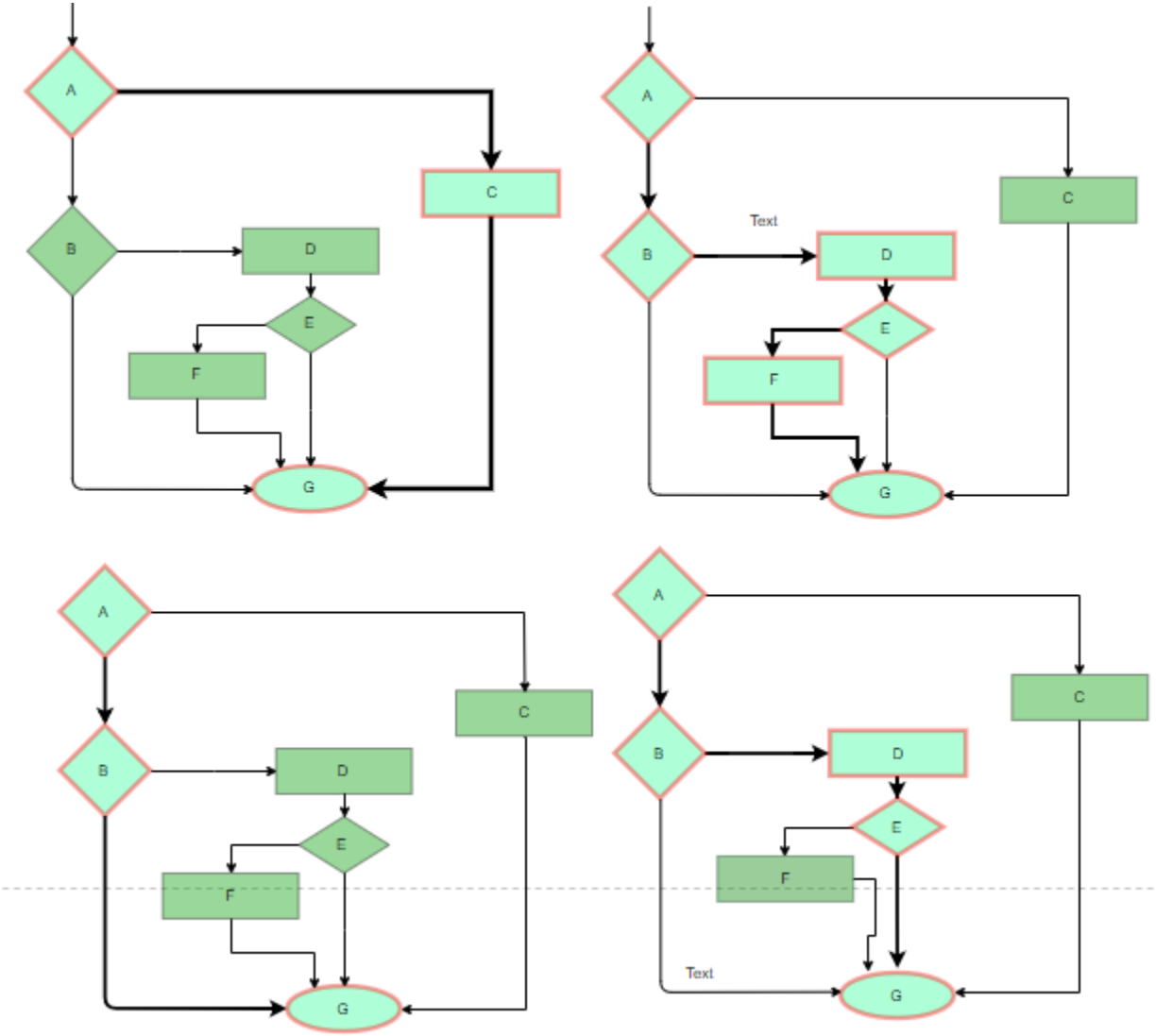
- **Statement coverage:** In this technique, the aim is to traverse all statement at least once. Hence, each line of code is tested.
- In case of a flowchart, every node must be traversed at least once. Since all lines of code are covered, helps in pointing out faulty code.





White box testing techniques

- **Branch Coverage:** In this technique, test cases are designed so that each branch from all decision points are traversed at least once. In a flowchart, all edges must be traversed at least once.



White box testing techniques

- **Condition Coverage:** In this technique, all individual conditions must be covered as shown in the following example:

READ X, Y

IF(X == 0 || Y == 0)

PRINT ‘0’

- In this example, there are 2 conditions: X == 0 and Y ==

  0. Now, test these conditions get TRUE and FALSE as their values. One possible example would be:

- #TC1 – X = 0, Y = 55
- #TC2 – X = 5, Y = 0

White box testing techniques

- **Basis Path Testing:** This technique, flow graph are made from code control flowchart Cyclomatic complexity is calculated which defines the number of independent paths so that the minimal number of test cases can be designed for each independent path.

White box testing techniques - Cyclomatic Complexity

- Cyclomatic Complexity in **Software Testing** is a testing metric used for measuring the **complexity** of a software program.
- It counts the number of decisions in the given program code.
- Cyclomatic complexity of a code section is the quantitative measure of the number of linearly **independent paths** in it.
- Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths
- It is computed using the **Control Flow Graph** of the program.

Importance of Cyclomatic Complexity

- It helps in determining the software quality.
- It is an important indicator of program code's readability, maintainability and portability.
- It helps the developers and testersto determine independent path executions.
- It helps to focus more on the uncovered paths.
- It evaluates the risk associated with the application or program.
- It provides assurance to the developers that all the paths have been tested at least once.

Cyclomatic Complexity

Cyclomatic Complexity	Meaning
1 – 10	• Structured and Well Written Code • High Testability • Less Cost and Effort
10 – 20	• Complex Code • Medium Testability • Medium Cost and Effort
20 – 40	• Very Complex Code • Low Testability • High Cost and Effort
> 40	• Highly Complex Code • Not at all Testable • Very High Cost and Effort

Calculating Cyclomatic Complexity

- Cyclomatic complexity is calculated using the control flow representation of the program code

Method-01:

Cyclomatic Complexity = Total number of closed regions in the control flow graph + 1

Method-02:

Cyclomatic Complexity = E – N + 2

Here-

E = Total number of edges in the control flow graph

N = Total number of nodes in the control flow graph

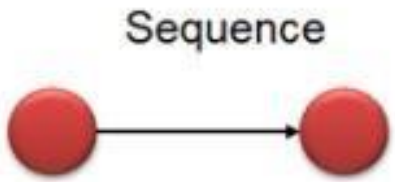
Calculating Cyclomatic Complexity

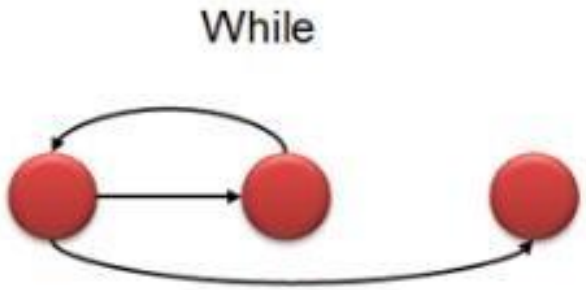
Method-03:

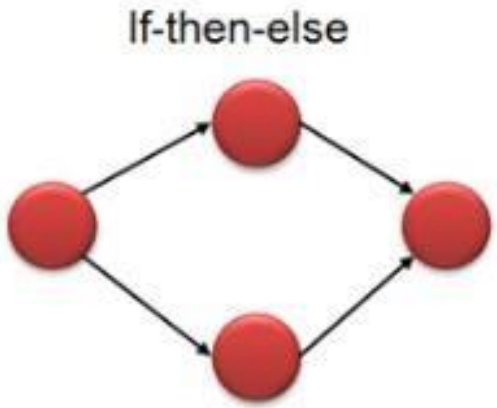
Cyclomatic Complexity = P + 1

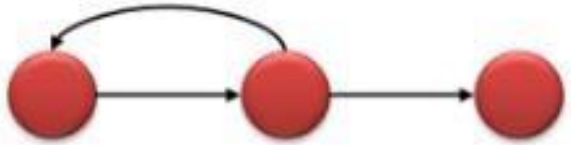
- Here, P = Total number of predicate nodes contained in the control flow graph
  - Predicate nodes are the conditional nodes.
  - They give rise to two branches in the control flow graph.

Control Flow graph notation for a program





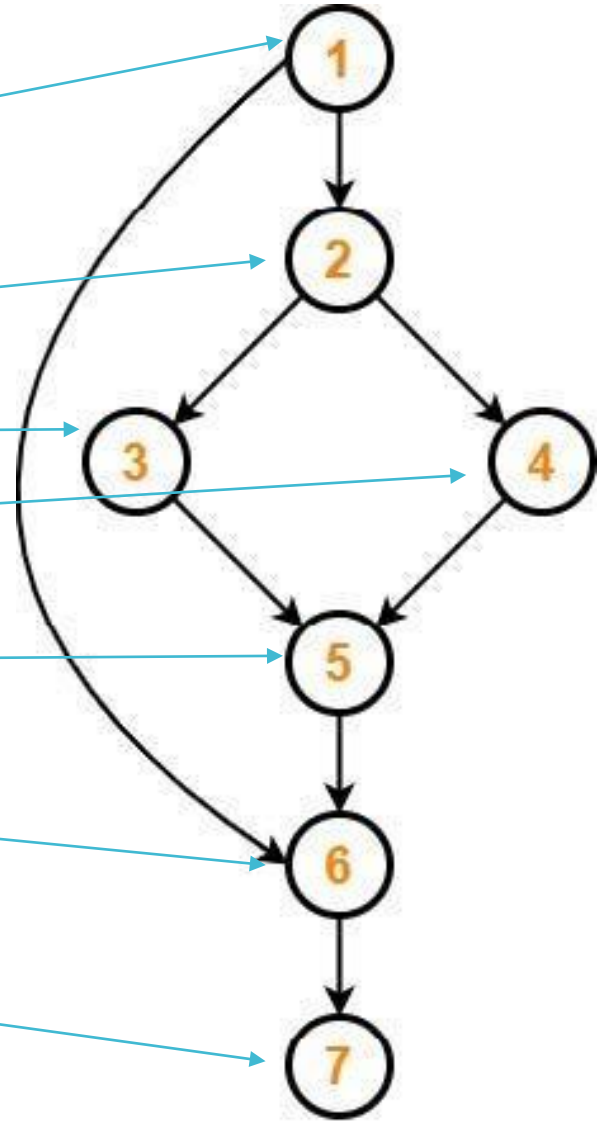




Example

```
IF A =
354THEN IF B > C
        THEN A = B
        ELSE A = C
    END IF
END IF

PRINT A
```



Control Flow Graph

Example

Method 1:

Cyclomatic Complexity = Total number of closed regions in control flow graph + the region outside

= 2 + 1

= 3

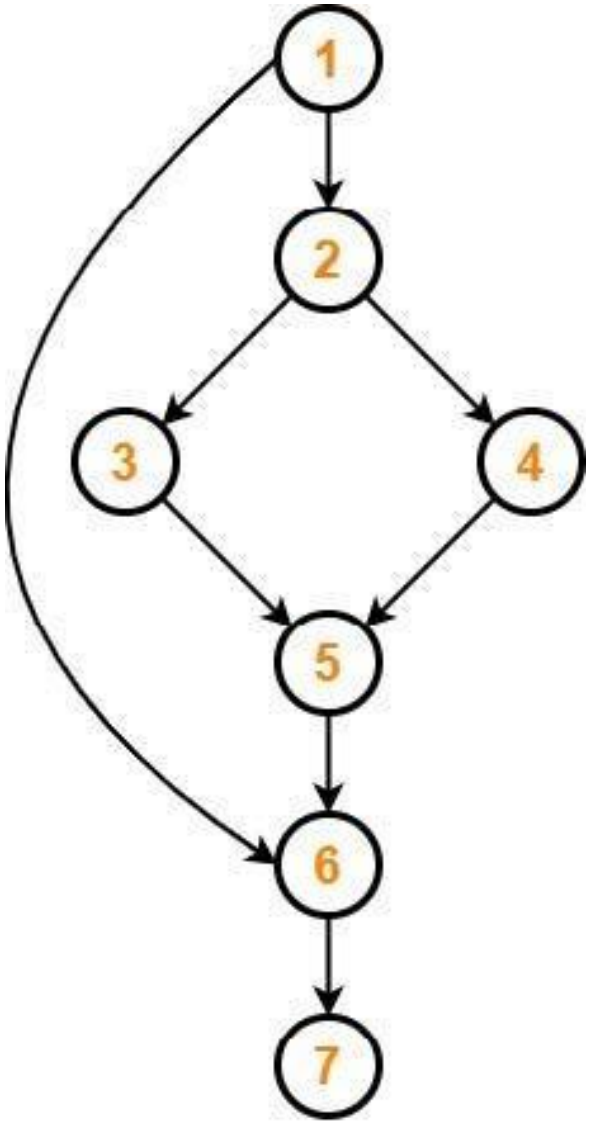
Method 2:

Cyclomatic Complexity

= E – N + 2

= 8 – 7 + 2

= **3**



Control Flow Graph

Example

Method-03:

Cyclomatic Complexity = p + 1

= 2 + 1

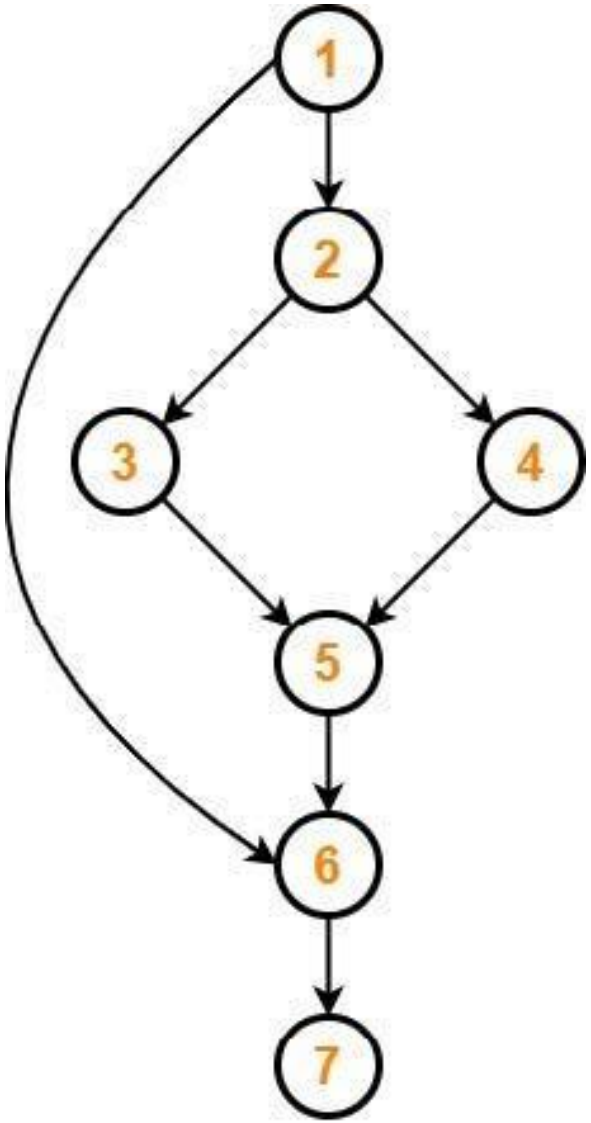
= 3

The Three independent paths are

Path1 : 1 2 3 5 6 7

Path2 : 1 2 4 5 6 7

Path3 : 1 6 7



Control Flow Graph

Advantages of White box testing (Glass- box Testing)

- Testing can be commenced at an earlier stage as one need not wait for the GUI to be available.
- Testing is more complete, with the possibility of covering most paths.

Disadvantages of White box testing (Glass- box Testing)

- Since tests can be very complex, highly skilled resources are required, with complete knowledge of programming and implementation.
- Test script maintenance can be a burden, if implementation changes too frequently.
- Since this method of testing is closely tied with the application being testing, tools to test every kind of implementation/platform may not be readily available.

Testing Object Oriented Applications

Unit Testing in the OO Context

- The concept of the unit changes in object-oriented software.
- In OO software, Encapsulation drives the definition of classes and objects.
  - Means, each class packages data and the operations that manipulate these data.
  - Rather than testing an individual module, here the smallest testable unit is the encapsulated class.
- Unlike unit testing of conventional software, which focuses on the algorithmic detail of a module and the data that flows across the module interface,
- Class testing for OO software is driven by the **operations encapsulated by the class** and **the state behavior of the class.**

Integration Testing in the OO Context

- There are two different strategies for integration testing of OO systems.

1. Thread-based testing

- Integrates the set of classes required to respond to one input or event for the system.
- Each thread is integrated and tested individually.
- Regression testing is applied to ensure that no side effects occur.

2. Use-based testing

- Begins the construction of the system by testing those classes (called independent classes) that use very few of server classes.
- After the independent classes are tested, the next layer of classes, called dependent classes, that use the independent classes are tested.

Validation Testing in an OO Context

- At the validation or system level, the details of class connections disappear.
- Like conventional software validation, the validation of OO focuses on user-visible actions and recognizable outputs from the system.
- To assist in the derivation of validation tests, the tester should draw upon use cases that are part of the requirements model.

Type of Testing Approach

Verification

- refers to the set of activities that ensure that software correctly implements a specific function.
- "Are we building the product right?"

Validation

- refers to the set of activities that ensure that the software that has been built is traceable to customer requirements.
- "Are we building the right product?

Verification and Validation

Verification	Validation
• Are we building the product right?"	• Are we building the right product?"
• Static practice of verifying documents, design, code and program.	• Dynamic mechanism of validating and testing the actual product.
• Does not involve executing the code.	• Always involves executing the code.
• Human based checking of documents and files.	• Computer based execution of program.
• Uses methods like inspections, reviews, walkthroughs etc.	• Uses methods like black box testing, white box testing etc.
• To check whether the software conforms to specifications.	• To check whether software meets the customer expectations and requirements.
• Target is requirements specification, application and software architecture, high level, complete design, and database design etc.	• Target is actual product-a unit, a module, a bent of integrated modules, and effective final product.
• Done by QA team to ensure that the software is as per the	• Carried out with the involvement of testing team

THANKYOU