Software Engineering 2UCCE501

Module 5

Module 5 Testing & Maintenance

5.1 Testing Concepts: Purpose of Software Testing, Testing Principles, Goals of Testing, Testing aspects: Requirements, Test Scenarios, Test cases, Test scripts/procedures,

5.2 Strategies for Software Testing, Testing Activities: Planning Verification and Validation, Software Inspections, FTR

5.3 Levels of Testing : unit testing, integration testing, regression testing, product testing, acceptance testing and White-Box Testing

5.4 Black-Box Testing: Test case design criteria, Requirement based Testing, Boundary value analysis, Equivalence Class Partitioning

5.5 Object Oriented Testing: Review of OOA and OOD models, class testing, integration testing, validation testing

5.6 Reverse & Reengineering, types of maintenance

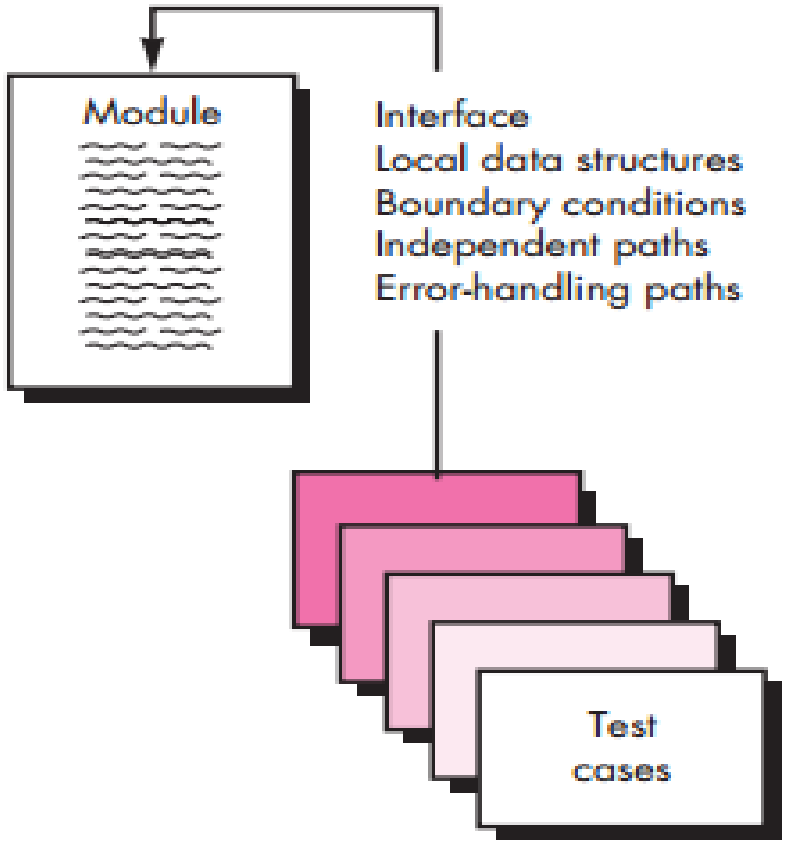
Levels of Testing (Unit Testing)

1. Unit Testing

- **Unit Testing** is a level of software testing where **individual units/ components** of a software are **tested.**
- It is concerned with **functional correctness of the standalone modules.**
- Important control paths are tested to **uncover errors within the boundary** of the module.
- The **relative complexity of tests and the errors those tests uncover** is limited by the constrained **scope** established for unit testing.
- focuses on the **internal processing logic and data structures**
- can be conducted in **parallel for multiple components**

Levels of Testing (Unit Testing)

- Unit Test

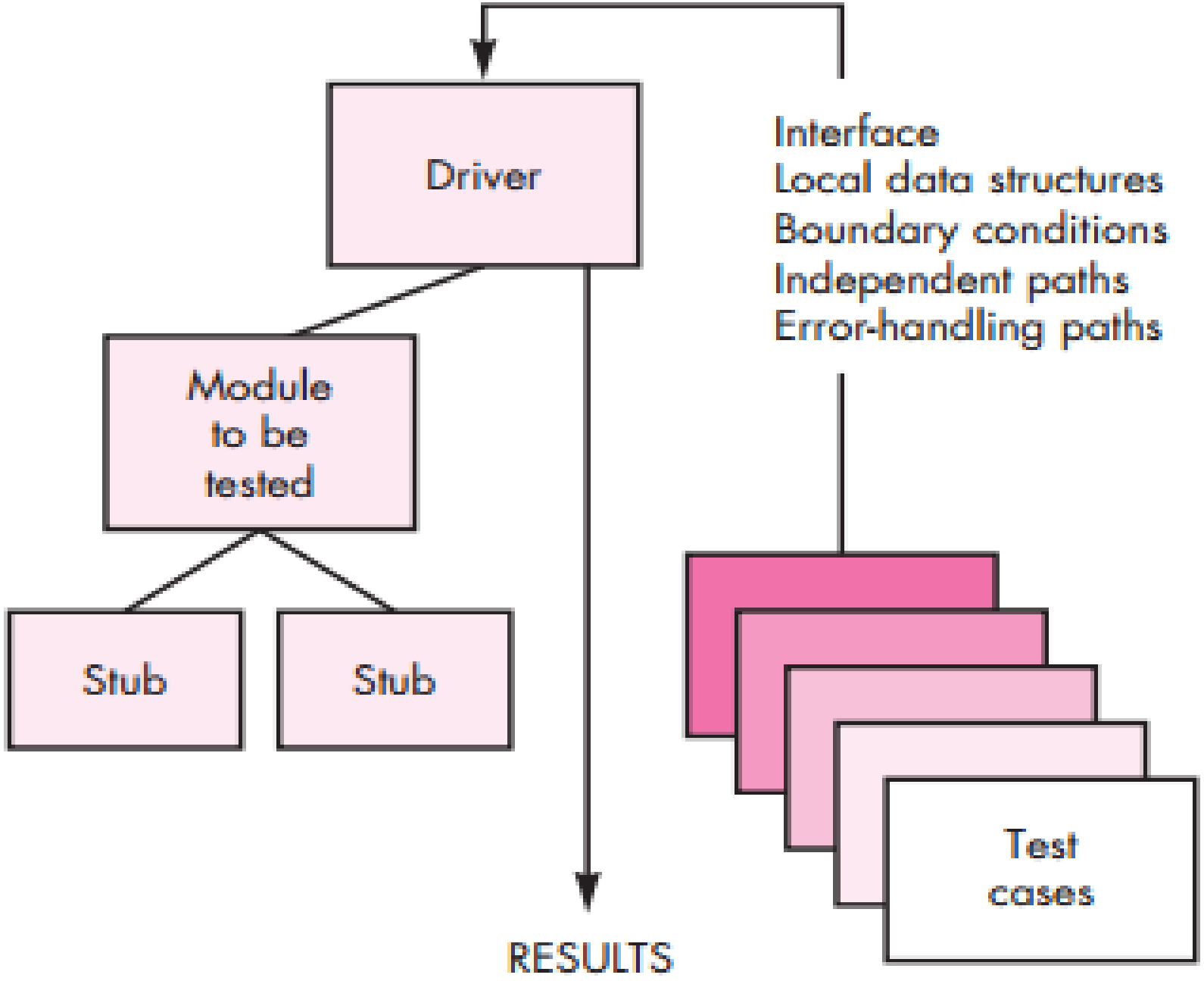


Levels of Testing (Unit Testing)

- The module interface is tested **to ensure that information properly flows into and out** of the program unit under test.
- All **independent paths through control structure exercised** to ensure all statements executed.
- Boundary conditions are tested to limit or restrict processing.
- all error-handling paths are tested.
- **Data flow** across a component interface is **tested before** any other testing.

Levels of Testing (Unit Testing)

- Test cases should be designed to uncover errors due to **improper computation , comparison and data flow.**
- Boundary testing is one of the most important unit testing tasks.
- **Errors often occur** when the **maximum or minimum allowable value** is encountered (boundary testing)
- Test cases that exercise **data structure, control flow, and data values just below, at, and just above maxima and minima** are very likely to uncover errors.


Driver
Interface
Local data structures
Boundary conditions
Independent paths
Error-handling paths
Module to be tested
Stub
Stub
Test cases
RESULTS

Levels of Testing (Unit Testing)

Drivers are considered as the dummy modules that always **simulate the high level modules.**

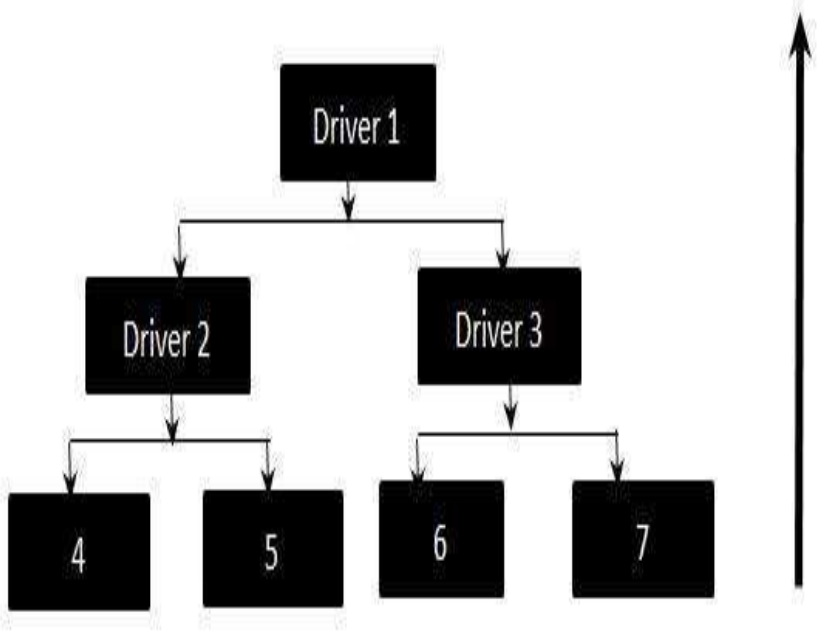
Stubs serve to replace modules that are subordinate (called by) the component to be tested.

A stub or "dummy subprogram" uses the subordinate module's interface, may do minimal data manipulation, prints verification of entry, and returns control to the module undergoing testing

Drivers and stubs are software that must be written but that is not delivered with the final software product.

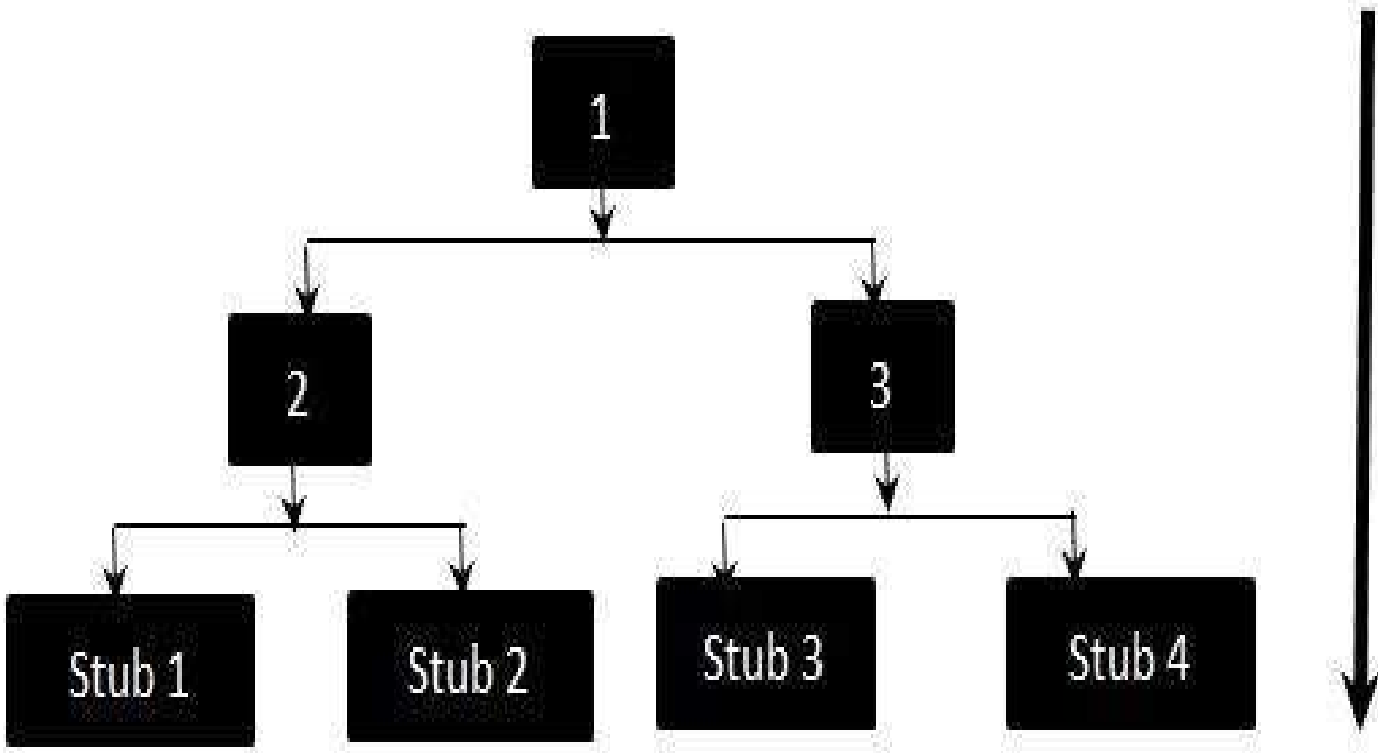
Many a times writing complete drivers/ stubs is not possible hence testing is postponed till integration testing

Driver - Flow Diagram:



Levels of Testing (Unit Testing)

Stubs are considered as the dummy modules that always **simulate the low level modules.**



Levels of Testing (Integration Testing)

2. Integration Testing

- **“If they all work individually, why do you doubt that they’ll work when we put them together?”**
- Why integration testing?
- Data can be lost across an interface
- one component can have an adverse effect on another
- Sub functions, when combined, may not produce the desired major function
- global data structures can present problems
- **“putting them together”—interfacing.**

Levels of Testing (Integration Testing)

- **Integration testing** is a systematic technique for **constructing the software architecture** while at the same time conducting tests to **uncover errors associated with interfacing.**
- take unit-tested components and build a program structure by design.
- **“big bang”** approach to integration is a lazy strategy that might result into failure – endless loop.

Levels of Testing (Integration Testing)

- **Incremental integration** - The program is constructed and tested in **small increments**, **easier to isolate and correct errors**.

- interfaces are more likely to be tested completely

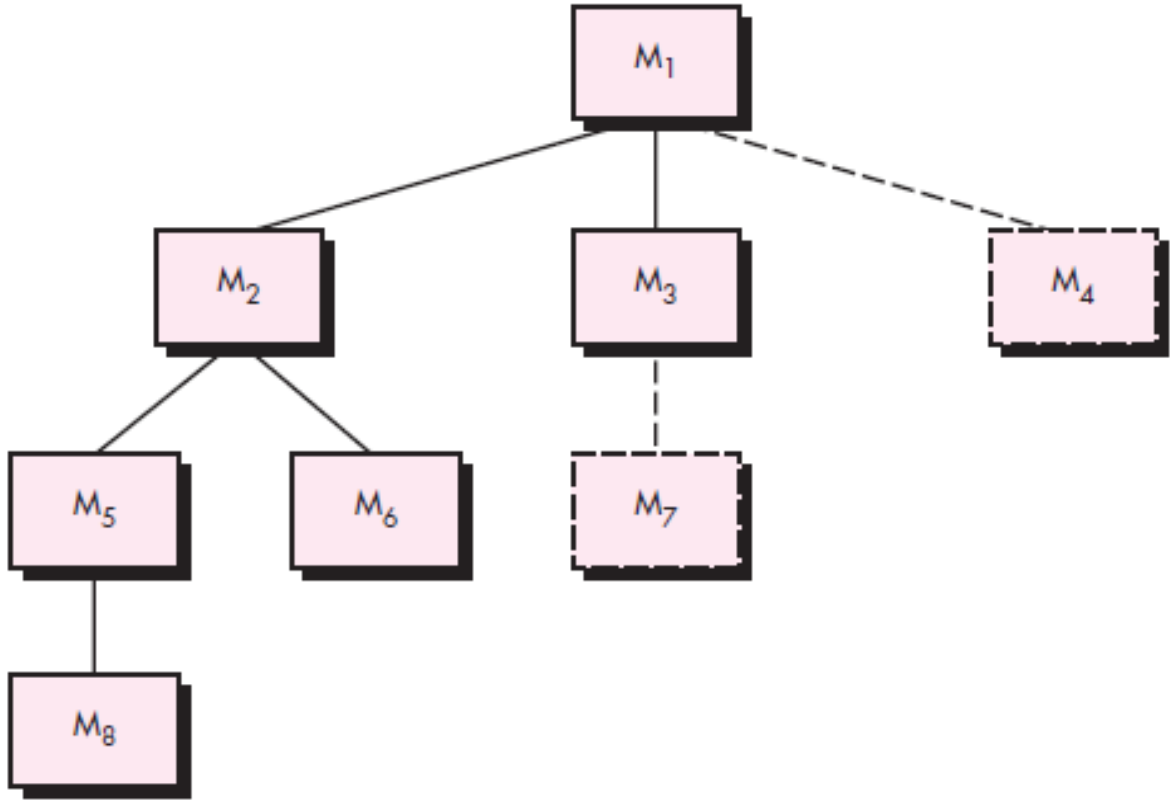
- Incremental Integration strategies:
  1. Top – down integration.
  2. Bottom – up integration.

Levels of Testing (Integration Testing)

- **Top Down** is an approach to Integration Testing where top level units are tested first and lower level units are tested step by step after that.

- **depth-first integration** integrates all components on a major control path of the program structure.

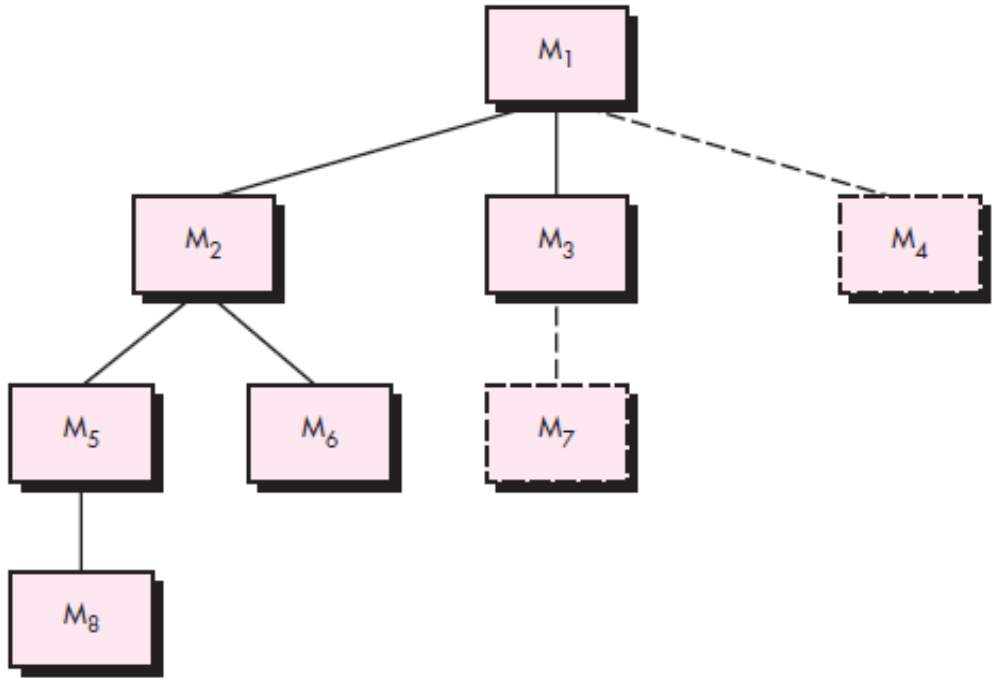
- For example, selecting the left-hand path, components M1, M2 , M5 would be integrated first. Next, M8 or M6 would be integrated.



Levels of Testing (Integration Testing)

- **Breadth-first integration** incorporates all components directly subordinate at each level, moving across the structure horizontally.

- For example: components M2, M3, and M4 would be integrated first. The next control level, M5, M6, and so on, follows.



Levels of Testing (Integration Testing)

- **Steps of Integration:**

1. The **main control module is used as a test driver** and **stubs are substituted** for all components directly subordinate to the main control module.
2. Depending on the **integration approach selected** (i.e., depth or breadth first), **subordinate stubs** are **replaced** one at a time with actual components.
3. **Tests are conducted** as each component is integrated.
4. On completion of each set of tests, **another stub is replaced with the real component.**
5. **Regression testing may be conducted** to ensure that new errors have not been introduced.

Levels of Testing (Integration Testing)

- The top-down integration strategy verifies major control or decision points early in the test process.
- Problem with top down strategy - when processing at low levels in the hierarchy is required to adequately test upper levels.
- Solution ?
- Bottom – up integration

Levels of Testing (Integration Testing)

- **Bottom Up** is an approach to Integration testing where **bottom level units are tested first** and **upper level units step by step after that.**

- Because components are **integrated** from the **bottom up**, need for **stubs is eliminated.**

- **The whole program does not exist until the last module is integrated.**

Levels of Testing (Integration Testing)

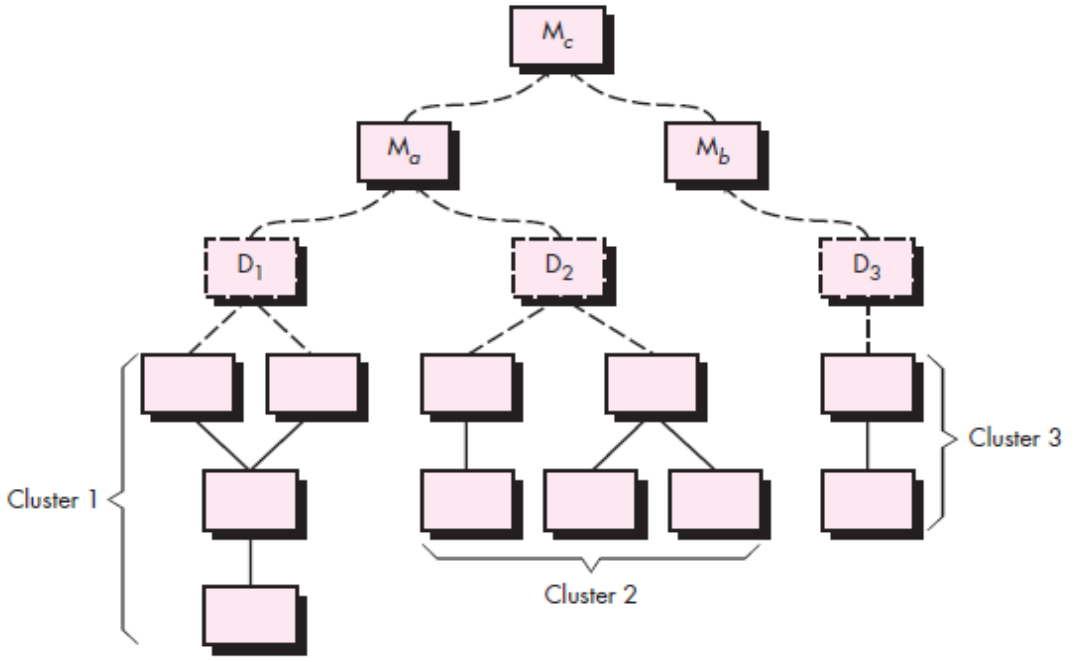
- **Steps of Integration:**

1. Low-level components are combined into clusters (sometimes called *builds*) that perform a specific software sub function.

2. A *driver* (a control program for testing) is written to coordinate test case input and output.

3. The cluster is tested.

4. Drivers are removed and clusters are combined moving upward in the program structure.



Levels of Testing (Regression Testing)

3 Regression Testing

- Regression testing is a type of software testing that **intends to ensure that changes** (enhancements or defect fixes) **to the software have not adversely affected it**.
- **new module is added** as part of integration testing, the **software changes**.
- New data flow paths, new I/O, and new control logic.
- **changes may cause problems** with functions that previously worked flawlessly.

Levels of Testing (Regression Testing)

- ***regression testing*** is the **re-execution of some subset of tests** that have already been conducted to **ensure that changes have not propagated unintended side effects.**
- Discovery of error and then correction changes some aspect of software configuration.
- Regression testing ensures **additional errors are not introduced.**
- Regression testing may be conducted **manually or by automated playback capture tools.**
- Effective Regression testing

Levels of Testing (Regression Testing)

- The regression test suite contains three different classes of test cases:

1. A representative sample of tests that will **exercise all software functions.**

2. **Additional tests** that focus on **software functions** that are **likely** to be **affected by the change.**

3. Tests that focus on the **software components** that have been **changed**.

Levels of Testing (Acceptance Testing)

4 Acceptance Testing

- **Acceptance testing**, a **testing** technique performed to determine whether or not the **software** system has **met** the **requirement specifications.**

- The main purpose of this test is to evaluate the **system's business requirements and verify delivery to end users.**

- Usually, Black Box Testing method is used in Acceptance Testing.

Levels of Testing (Acceptance Testing)

- ***benchmark test***, the client prepares a set of test cases that represent typical conditions under which the system should operate.

- ***competitor testing*****,** the new system is tested against an existing system or competitor product.

- ***shadow testing***, a form of comparison testing, the new and the legacy systems are run in parallel and their outputs are compared.

Levels of Testing (Acceptance Testing)

- After acceptance testing, **the client reports to the project manager which requirements are not satisfied.**
- Acceptance testing also gives the opportunity for a **dialog** between the developers and client.
- If **requirements must be changed** form the basis for **another iteration** of the software life-cycle process.
- If the customer is satisfied, the system is accepted.

Levels of Testing (White Box Testing)

- White Box Testing
- also known as Clear Box Testing, Open Box Testing, Glass Box Testing, Transparent Box Testing, Code-Based Testing or Structural Testing is a software testing method in which the internal structure/ design/ implementation of the item being tested is known to the tester.

Levels of Testing (White Box Testing)

- Using white-box testing methods, you can derive test cases that

(1) all independent paths within a module have been exercised

(2) exercise **all logical decisions** on their true and false sides

(3) Execute **all loops at their boundaries** and within their **operational bounds**

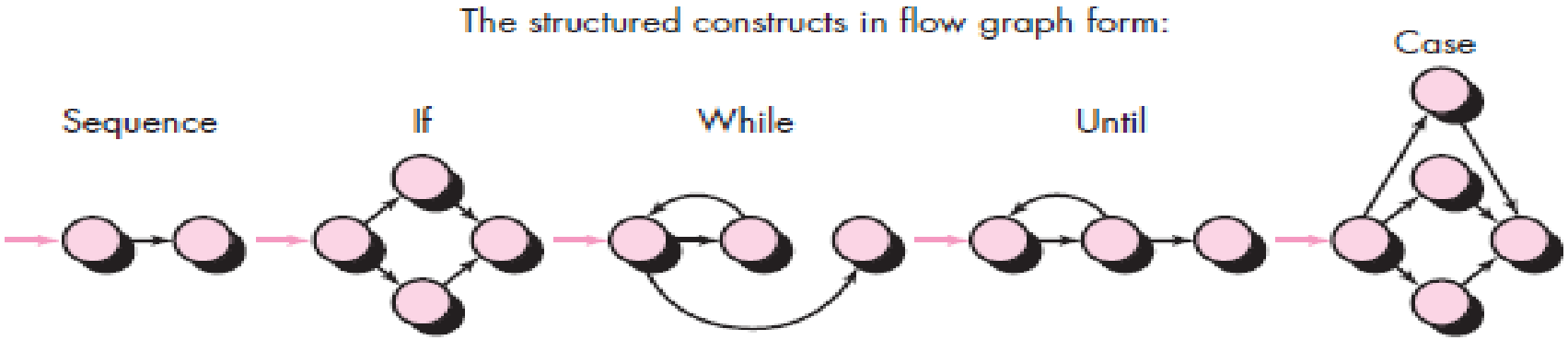
(4) Exercise **internal data structures** to ensure their validity.

Levels of Testing (White Box Testing)

- **Basis path testing** is a white-box testing technique.
- The basis path method – complexities , execution paths(data flow), independent paths and procedural design.
- Test cases derived guaranteed to execute every statement in the program

Levels of Testing (White Box Testing)

- **Flow Graph Notation**



- each circle, called a ***flow graph node,*** represents one or more procedural statements.

- ***edges* or *links,*** represent flow of control and are analogous to flowchart arrows.

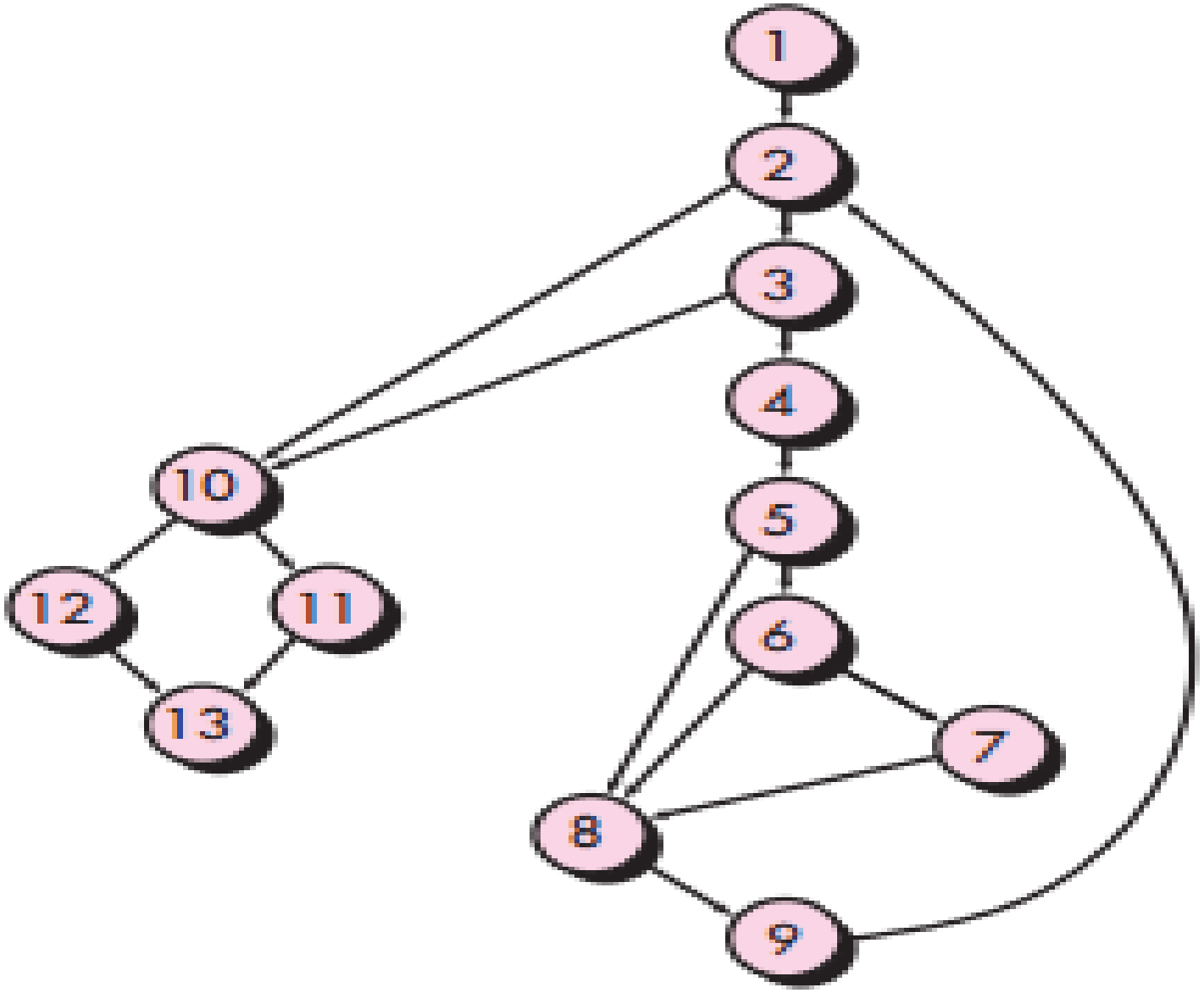
Levels of Testing (White Box Testing)

- Areas bounded by edges and nodes are called ***regions.***
- Each node that contains a condition is called a ***predicate node.***
- An ***independent path*** is any path through the program that introduces at least **one new set of processing statements or a new condition**.

Levels of Testing (White Box Testing)

- ***Cyclomatic complexity*** is a software metric that provides a quantitative measure of the logical complexity of a program.

- Cyclomatic complexity **defines number of independent paths** which can be further used in development of test cases.


1
2
3
4
5
6
7
8
9
10
11
12
13

Levels of Testing (White Box Testing)

- **Complexity is computed in one of three ways:**

1. The number of regions of the flow graph corresponds to the cyclomatic complexity.

2. Cyclomatic complexity $V(G)$ for a flow graph G is defined as

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

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

3. Cyclomatic complexity $V(G)$ for a flow graph G is also defined as

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

where P is the number of predicate nodes contained in the flow graph G.

Levels of Testing (White Box Testing)

white-box test design technique: Procedure to derive and/or **select test cases** based on an **analysis of the internal structure of a component or system.**

- **4 steps for Deriving Test Cases**

1. **Using the design or code as a foundation, draw a corresponding flow graph.**

2. **Determine the cyclomatic complexity of the resultant flow graph.**

3. **Determine a set of linearly independent paths.**

   The value of *V*(*G*) provides the upper bound on the number of linearly independent paths.

Levels of Testing (White Box Testing)

4. Prepare test cases that will force execution of each path in the set.

- Data should be chosen so that conditions at the predicate nodes are appropriately set as each path is tested.
- Each test case is executed and compared to expected results.
- be sure that all statements in the program have been executed at least once.

Levels of Testing (White Box Testing)

- A data structure, called a ***graph matrix,*** can be quite useful for developing a **software tool that assists in basis path testing.**