OXFORD
HIGHER EDUCATION

Software Testing

Naresh Chauhan, Assistant Professor in the Department of Computer Engineering at the YMCA University of Science and Technology, Faridabad

OXFORD
HIGHER EDUCATION

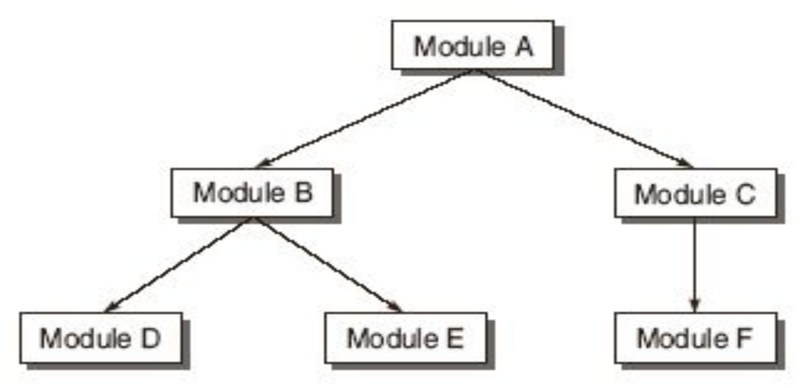
Chapter 7
Validation Activities

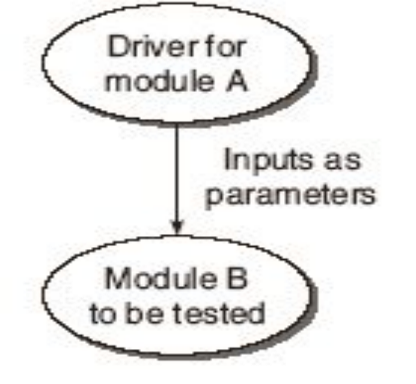
Objectives

- Validation is the next step after verification.
- Validation is performed largely by black box testing techniques.
- Unit validation testing
- Integration Testing and its types
- Function Testing
- System Testing and its types: Recovery testing, Security testing, Stress testing, Performance Testing, Usability testing, Compatibility testing
- Acceptance Testing and its types: Alpha and Beta testing

Unit Validation Testing

- Drivers



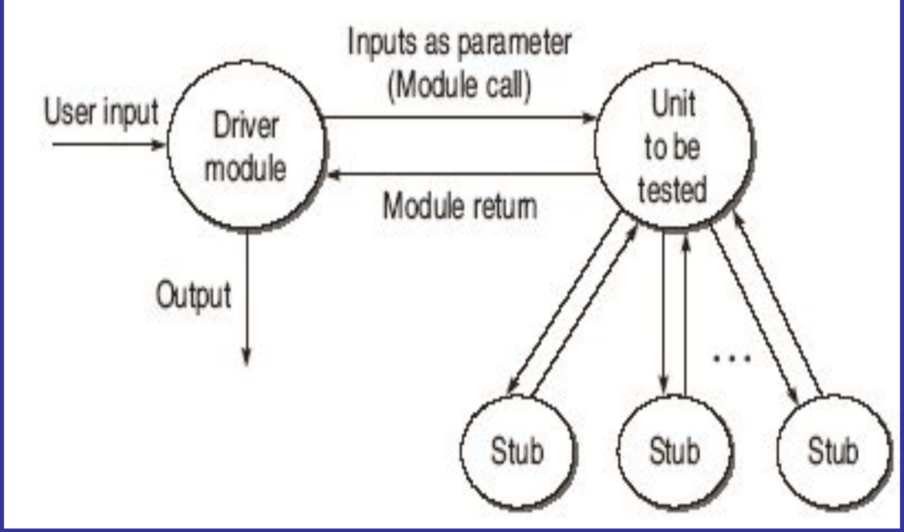


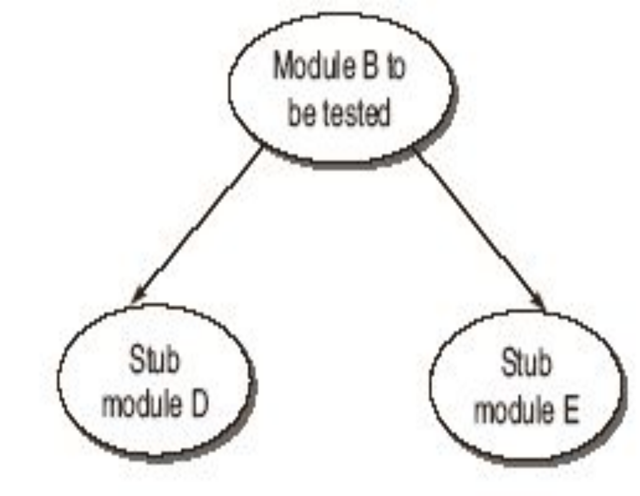
- a test driver is supporting code and data used to provide an environment for testing part of a system in isolation.
- A test driver may take inputs in the following form and call the unit to be tested:
- It may hardcode the inputs as parameters of the calling unit.
- It may take the inputs from the user.
- It may read the inputs from a file.

Unit Validation Testing

- **Stubs**

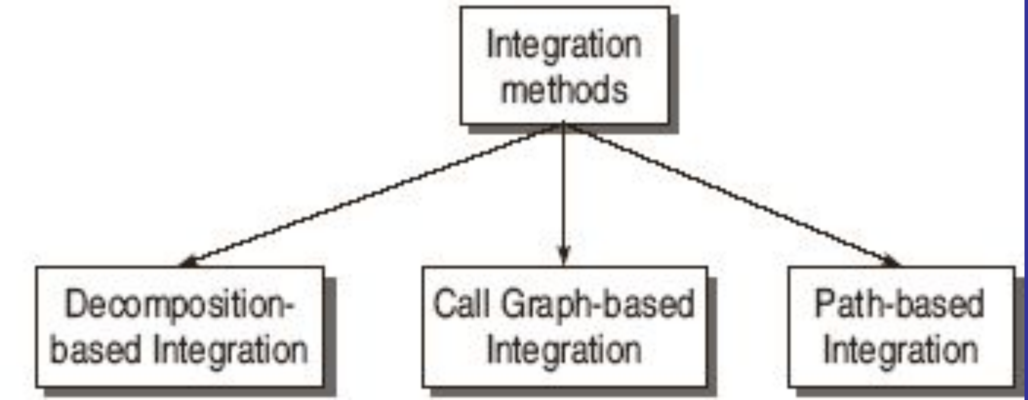
  Stub can be defined as a piece of software that works similar to a unit which is referenced by the Unit being tested, but it is much simpler that the actual unit.



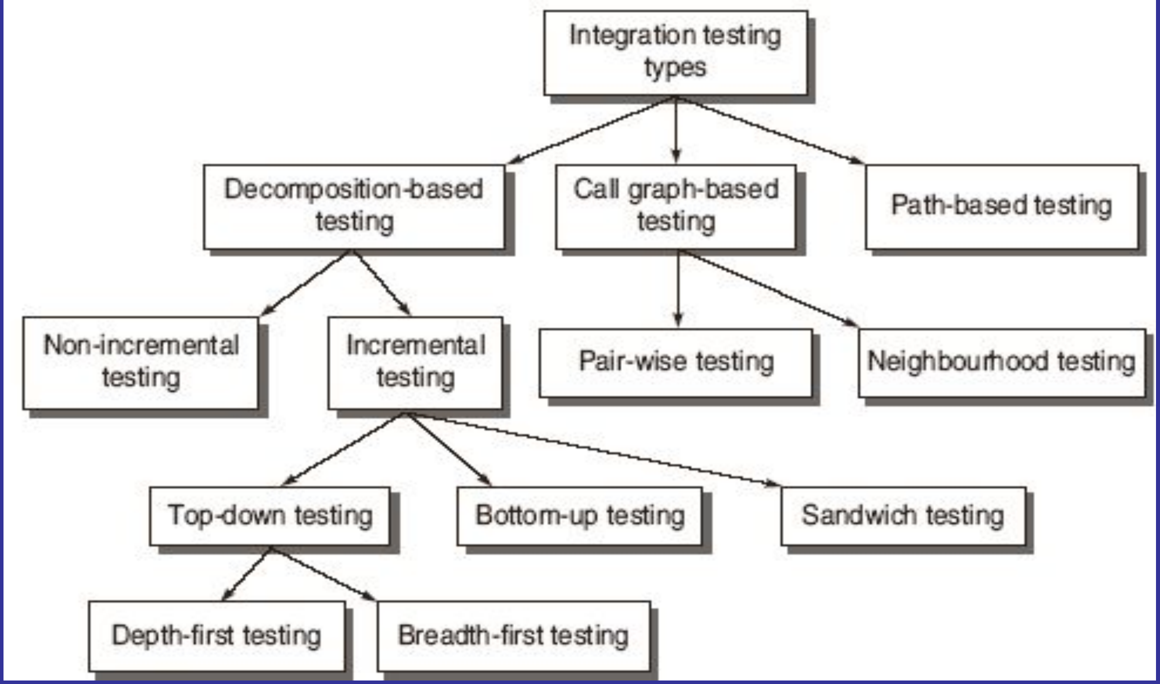


Integration Testing

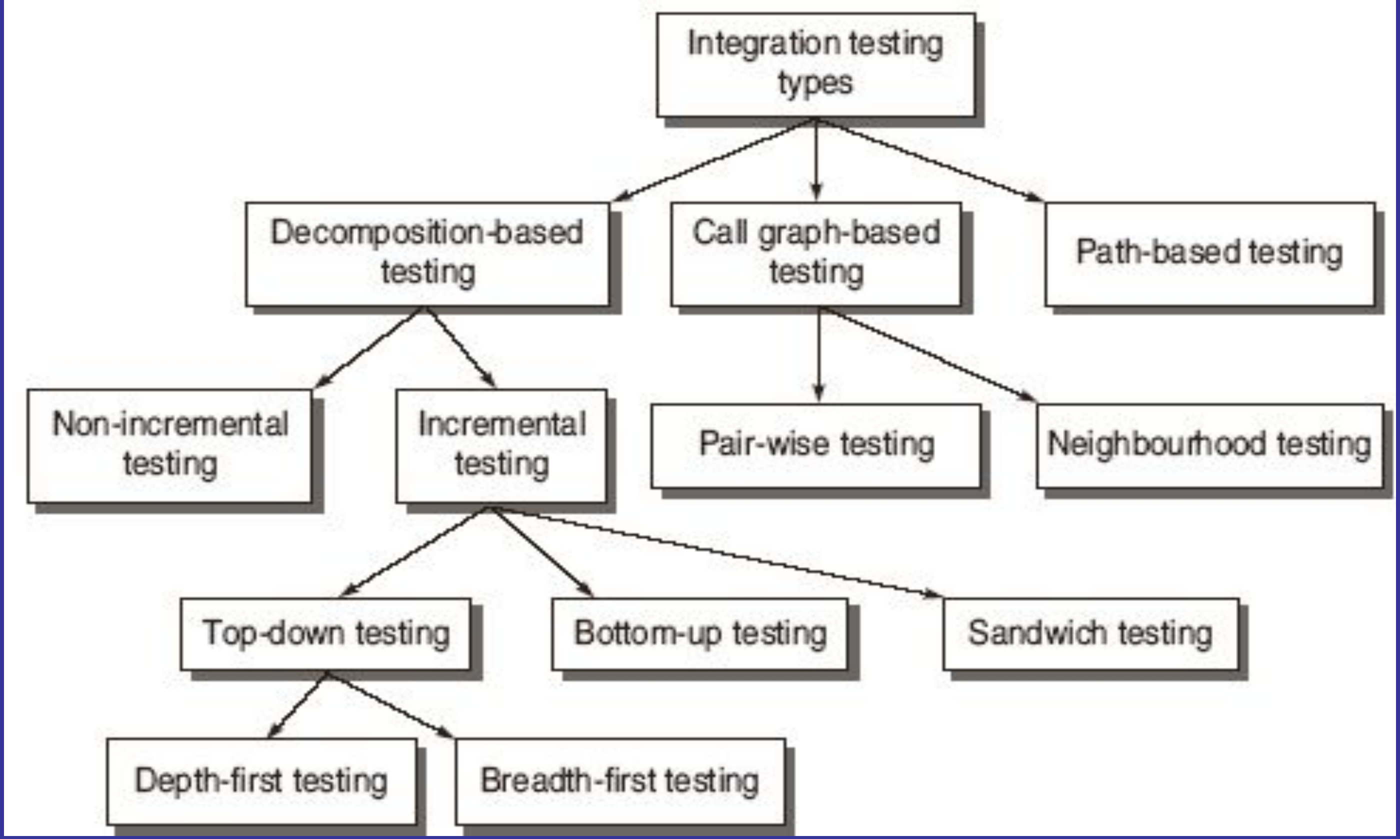
- Integration testing exposes inconsistency between the modules such as improper call or return sequences.
- Data can be lost across an interface.
- One module when combined with another module may not give the desired result.
- Data types and their valid ranges may mismatch between the modules.



Decomposition Based Integration



Integration Testing



Practical Approach for Integration Testing

OXFORD
HIGHER EDUCATION

There is no single strategy adopted for industry practice.

For integrating the modules, one cannot rely on a single strategy. There are situations depending upon the project in hand which will force to integrate the modules by combining top-down and bottom-up techniques.

This combined approach is sometimes known as Sandwich Integration testing.

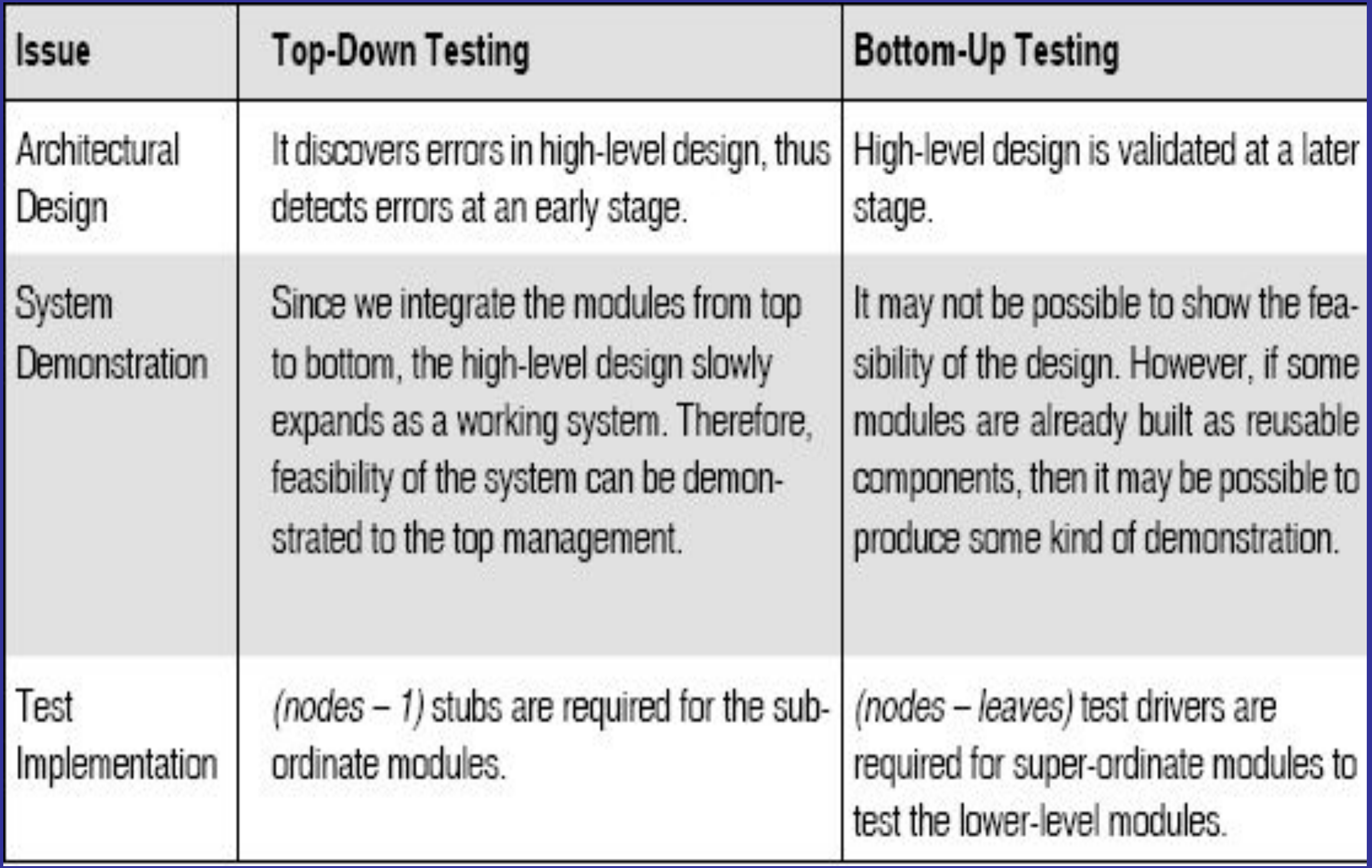
The practical approach for adopting sandwich testing is driven by the following factors:
Priority
Availability

Decomposition based integration testing

- The integration testing effort is computed as number of test sessions. A test session is one set of test cases for a specific configuration. The total test sessions in decomposition based integration is computed as:

  Number of test sessions = nodes – leaves + edges

Incremental Integration Testing

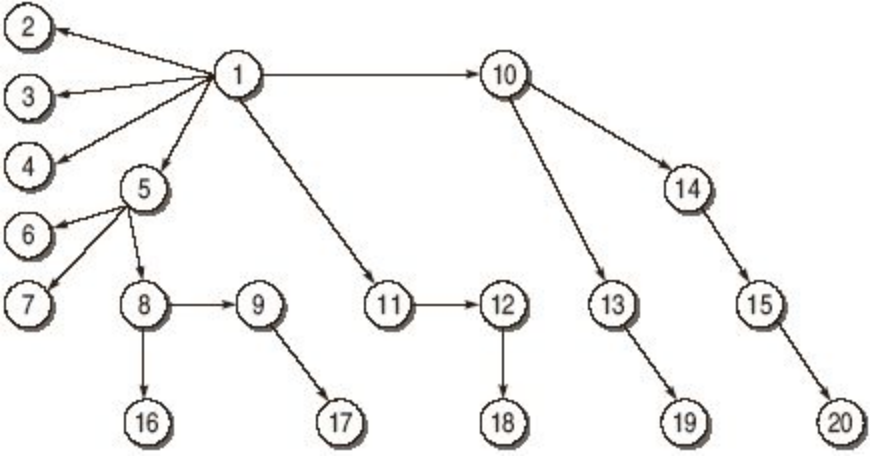


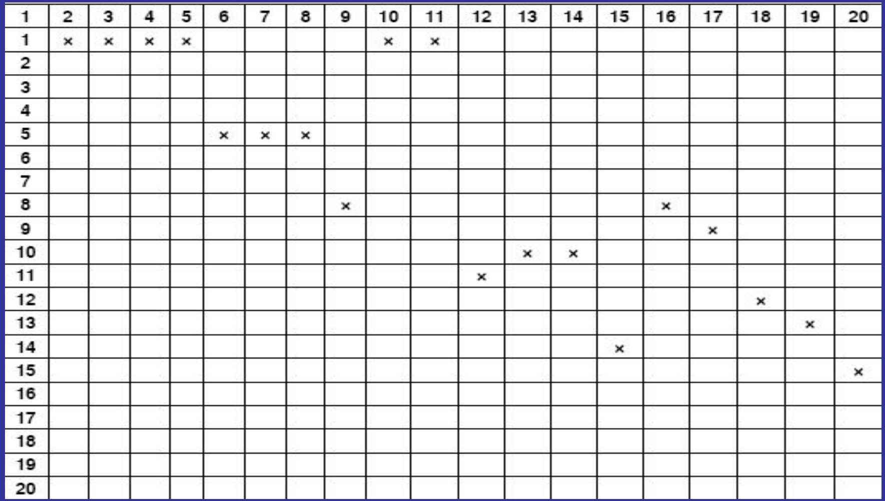
Issue	Top-Down Testing	Bottom-Up Testing
Architectural Design	It discovers errors in high-level design, thus detects errors at an early stage.	High-level design is validated at a later stage.
System Demonstration	Since we integrate the modules from top to bottom, the high-level design slowly expands as a working system. Therefore, feasibility of the system can be demonstrated to the top management.	It may not be possible to show the feasibility of the design. However, if some modules are already built as reusable components, then it may be possible to produce some kind of demonstration.
Test Implementation	*(nodes – 1)* stubs are required for the subordinate modules.	*(nodes – leaves)* test drivers are required for super-ordinate modules to test the lower-level modules.

Call Graph Based Integration

- A call graph is a directed graph wherein nodes are modules or units and a directed edge from one node to another node means one module has called another module. The call graph can be captured

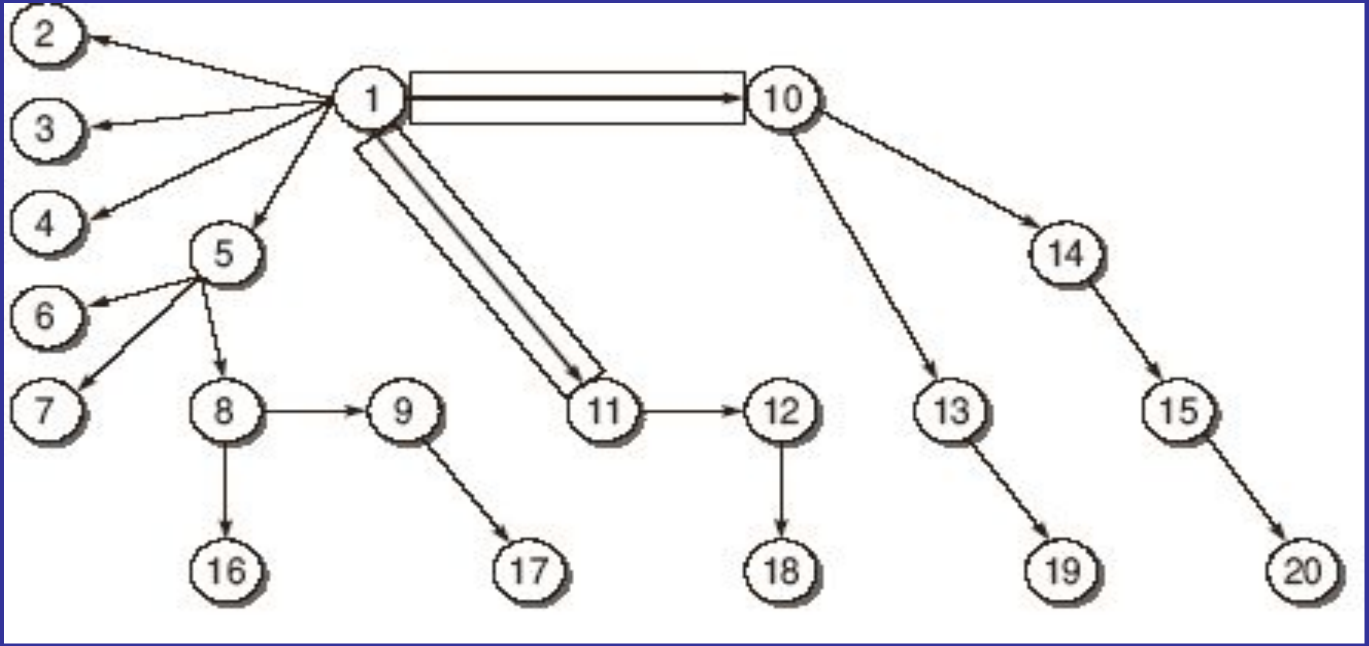
  in a matrix form which is known as adjacency matrix.



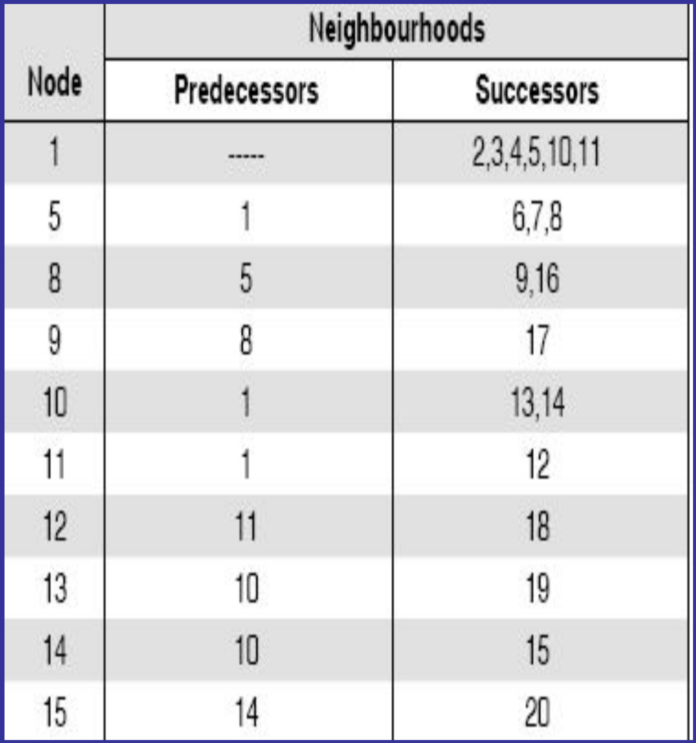


1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	×	×	×	×					×	×									
2																			
3																			
4																			
5					×	×	×												
6																			
7																			
8								×							×				
9																×			
10												×	×						
11											×								
12																	×		
13																		×	
14														×					
15																			×
16																			
17																			
18																			
19																			
20																			

Pair-wise Integration



Neighborhood Integration



Node	Neighbourhoods	
	Predecessors	Successors
1	-----	2,3,4,5,10,11
5	1	6,7,8
8	5	9,16
9	8	17
10	1	13,14
11	1	12
12	11	18
13	10	19
14	10	15
15	14	20

The total test sessions in neighborhood integration can be calculated as:

Neighborhoods = nodes – sink nodes

= 20 - 10

= 10

where Sink Node is an instruction in a module at which execution terminates.

Path Based Integration

- Source Node
- Sink Node
- Module Execution Path (MEP) Message
- MM-Path
- MM-Path Graph

Path Based Integration

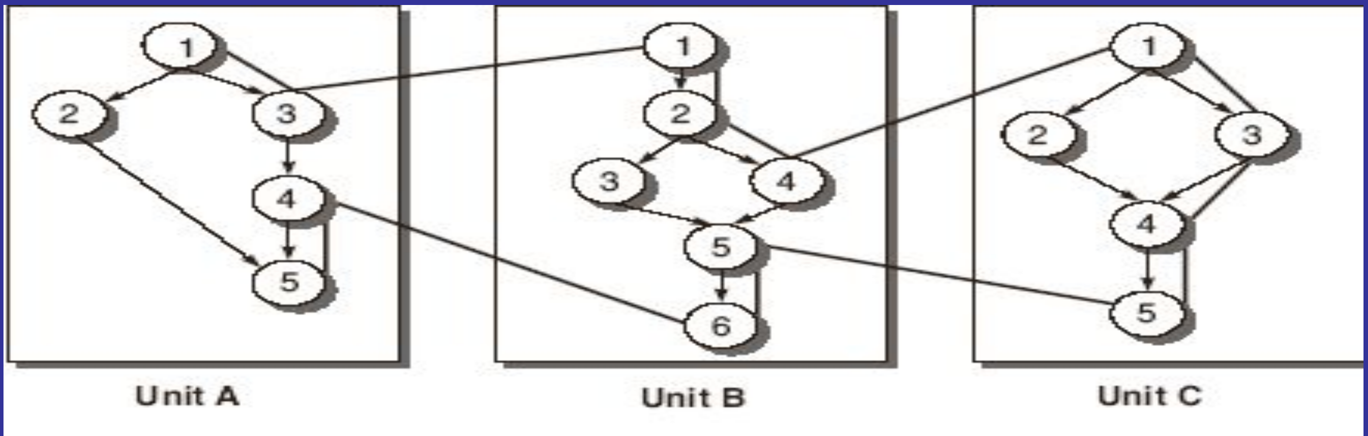
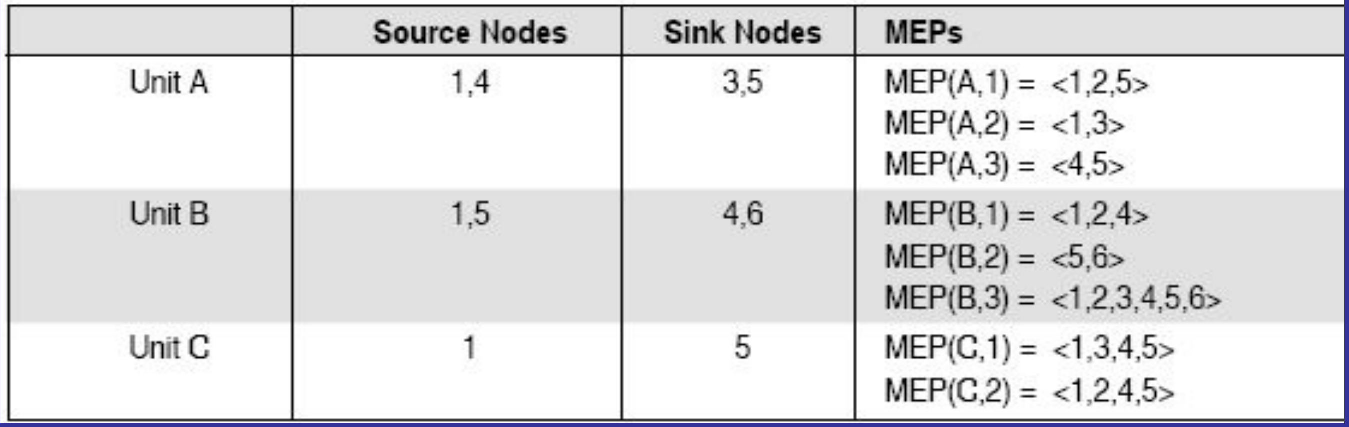


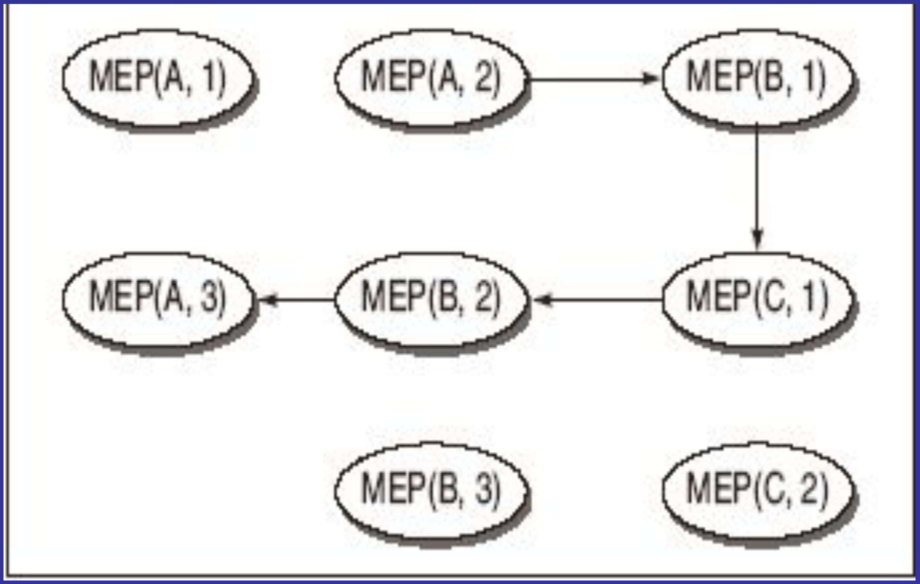
Figure 7.12 MM-path

Table 7.3 MM-path details



	Source Nodes	Sink Nodes	MEPs
Unit A	1,4	3,5	MEP(A,1) = <1,2,5> MEP(A,2) = <1,3> MEP(A,3) = <4,5>
Unit B	1,5	4,6	MEP(B,1) = <1,2,4> MEP(B,2) = <5,6> MEP(B,3) = <1,2,3,4,5,6>
Unit C	1	5	MEP(C,1) = <1,3,4,5> MEP(C,2) = <1,2,4,5>

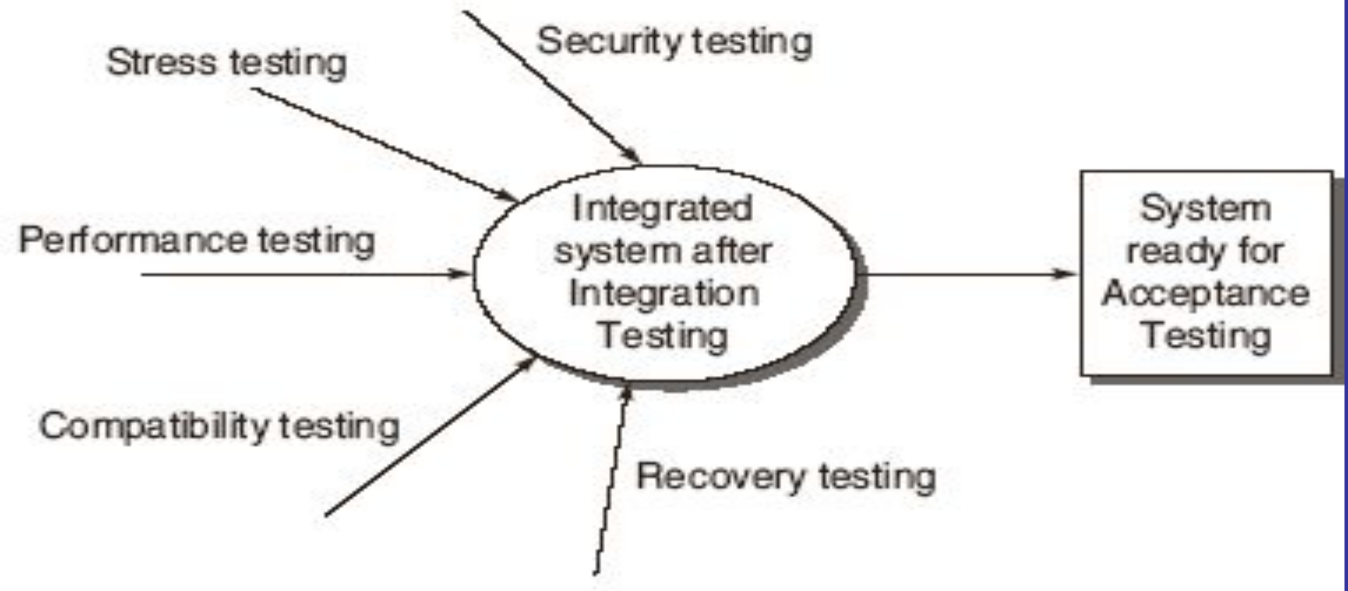
Path Based Integration



Function Testing

- *The process of attempting to detect discrepancies between the functional specifications of a software and its actual behavior.*
- *The function test must determine if each component or business event:*
  - *performs in accordance to the specifications,*
  - *responds correctly to all conditions that may be presented by incoming events / data,*
  - *moves data correctly from one business event to the next (including data stores), and*
  - *business events are initiated in the order required to meet the business objectives of the system.*

System Testing



Recovery Testing

- Recovery is just like the exception handling feature in a programming language.

- Recovery is the ability of a system to restart operations after the integrity of the application has been lost.

- It reverts to a point where the system was functioning correctly and then reprocesses the transactions up until the point of failure Some

Security Testing

- Confidentiality
- Integrity
- Authentication
- Authorization
- Availability
- Non-repudiation

Performance Testing

The performance testing requires that performance requirements must be clearly mentioned in SRS and system test plans.

The main thing is that these requirements must be quantified.

For example, a requirement that the system return a response to a query in a reasonable amount is not an acceptable requirement; the time must be specified in a quantitative way.

- **Load Testing**
- **Stress Testing**

Usability Testing

Ease of Use

Interface steps

Response Time

Help System

Error Messages

Compatibility/Conversion/Configuration Testing

OXFORD
HIGHER EDUCATION

- **Operating systems:** The specifications must state all the targeted end-user operating systems on which the system being developed will be run.

- **Software/ Hardware:** The product may need to operate with certain versions of web browsers, with hardware devices such as printers, or with other software, such as virus scanners or word processors.

- **Conversion Testing**
- **Ranking of possible configurations: Identification of test cases:**

- **Updating the compatibility test cases**

Acceptance Testing

- *Acceptance Testing is the formal testing conducted to determine whether a software system satisfies its acceptance criteria and to enable buyer to determine whether to accept the system or not.*
- Determine whether the software is fit for the user to use.
- Making users confident about product
- Determine whether a software system satisfies its acceptance criteria.
- Enable the buyer to determine whether to accept the system.

- Alpha Testing Beta Testing