Integration Testing

Prof. Durga Prasad Mohapatra
Professor
Dept.of CSE, NIT Rourkela

Introduction

- In the modular design of a software system where the system is composed of different modules, integration is the activity of combining the modules together when all the modules have been prepared.
- Integration of modules is done according to the design of software specified earlier.
- Integration aims at constructing a working software system.
- But a working software demands full testing and thus, integration testing comes into the picture.

Introduction cont …

- Why do we need integration testing? When all modules have been verified independently, then why is integration testing necessary?

- As discussed earlier, modules are not standalone entities.

- They are a part of a software system which comprises of many interfaces. Even if a single interface is mismatched, many modules may be affected.

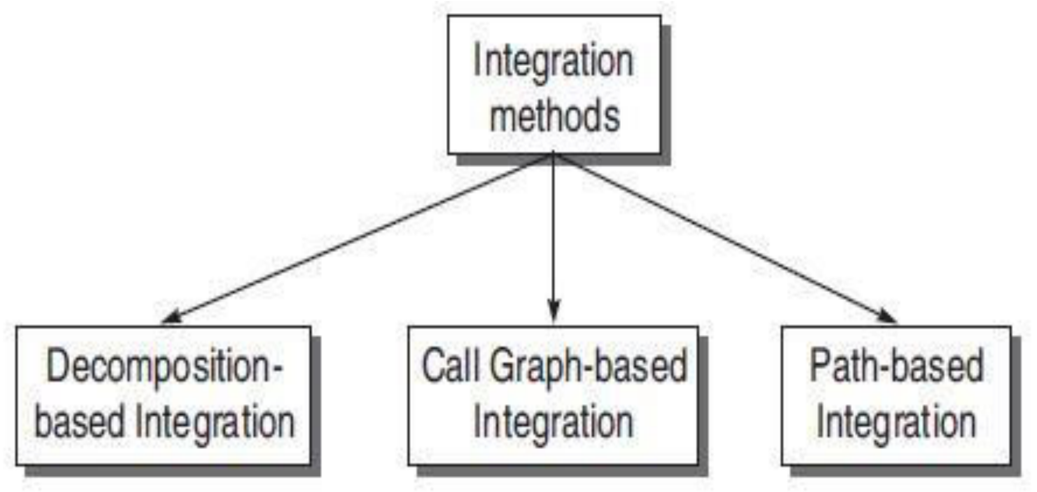
Introduction cont …

Thus, integration testing is necessary for the following reasons:

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

Approaches for integration testing

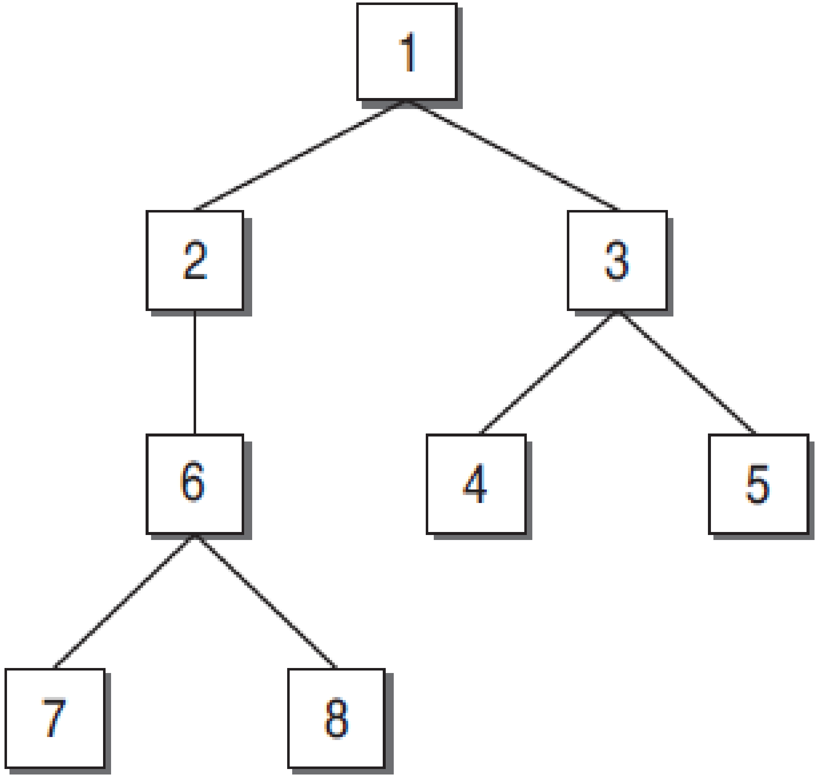
- Thus, integration testing focuses on bugs caused by interfacing between the modules while integrating them.
- There are three approaches for integration testing:



Decomposition-based integration

- Idea is based on decomposition into functional modules
- Functional decomposition is shown as a tree
- Classified into
  - Non-incremental
  - incremental

Integration Testing



- Non incremental Approach
  - Also known as big-bang integration
  - Discarded for the following reasons:
    - Big-bang requires more work
      - With increase in the number of modules, more number of drivers and stubs will be required to test the modules independently.
    - Actual modules are not interfaced directly until the end of the software system.
    - It will be difficult to localize the errors since the exact location of bugs cannot be found easily.

- Incremental Approach
  - It is beneficial for the following reasons:
    - Does not require many drivers and stubs.
    - Interfacing errors are uncovered earlier.
    - It is easy to localize the errors since modules are combined one by one, thus debugging is easier.
    - Incremental testing is a more thorough testing

Types of incremental integration testing

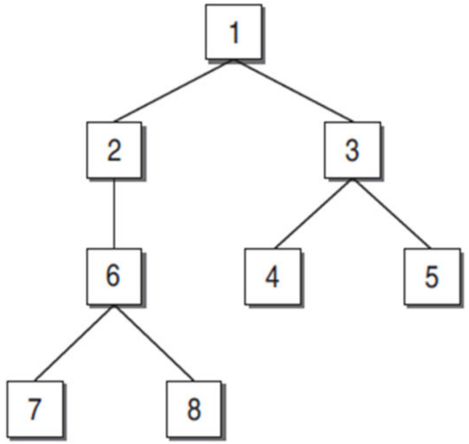
1. Top-down integration
2. Bottom-up integration

Top-down integration Testing

- The strategy in top-down integration is to look at the design hierarchy from top to bottom.
- Start with the high-level modules and move downward through the design hierarchy.
- Modules subordinate to the top module are integrated in the following two ways:
  - Depth first integration
  - Breadth first integration

Depth first integration

- In this type, all modules on a major control path of the design hierarchy are integrated first.
- In the example shown in below figure, modules 1, 2, 6, 7/8 will be integrated first. Next, modules 1, 3, 4/5 will be integrated.



Breadth first integration

- In this type, all modules directly subordinate at each level, moving across the design hierarchy horizontally, are integrated first.
- In the example shown in previous figure, modules 2 and 3 will be integrated first.
- Next, modules 6, 4, and 5 will be integrated. Modules 7 and 8 will be integrated last.

However, in practice, these two sequences of top-down integration cannot be used every time. In general, there is no best sequence, but the following guidelines can be considered:

1. In practice, the availability of modules matters the most. The module which is ready to be integrated, will be integrated and tested first. We should not wait to test it according to depth first or breadth first sequence, but use the availability of modules.

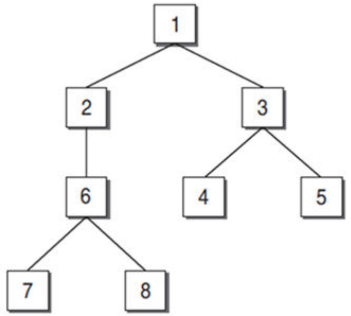
2. If there are critical sections of the software, design the sequence such that these sections will be added and tested as early as possible. A critical section might be a complex module, a module with a new algorithm or a module suspected to be error prone.

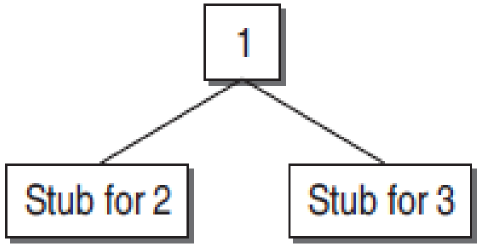
3. Design the sequence such that the I/O modules are added as early as possible, so that all interface errors will be detected earlier.

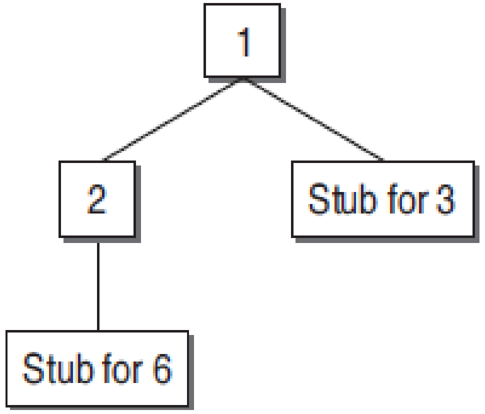
Top-Down Integration Procedure

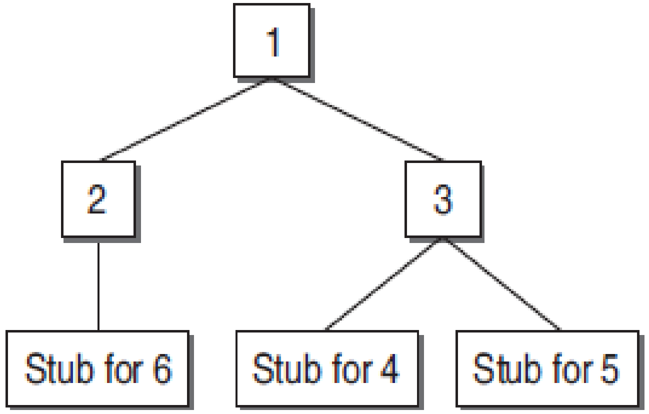
1. Start with the top or initial module in the software. Substitute the stubs for all the subordinate modules of top module. Test the top module.
2. After testing the top module, stubs are replaced one at a time with the actual modules for integration.
3. Perform testing on this recent integrated environment.
4. Regression testing may be conducted to ensure that new errors have not appeared.
5. Repeat steps 2-4 for the whole design hierarchy.

Example

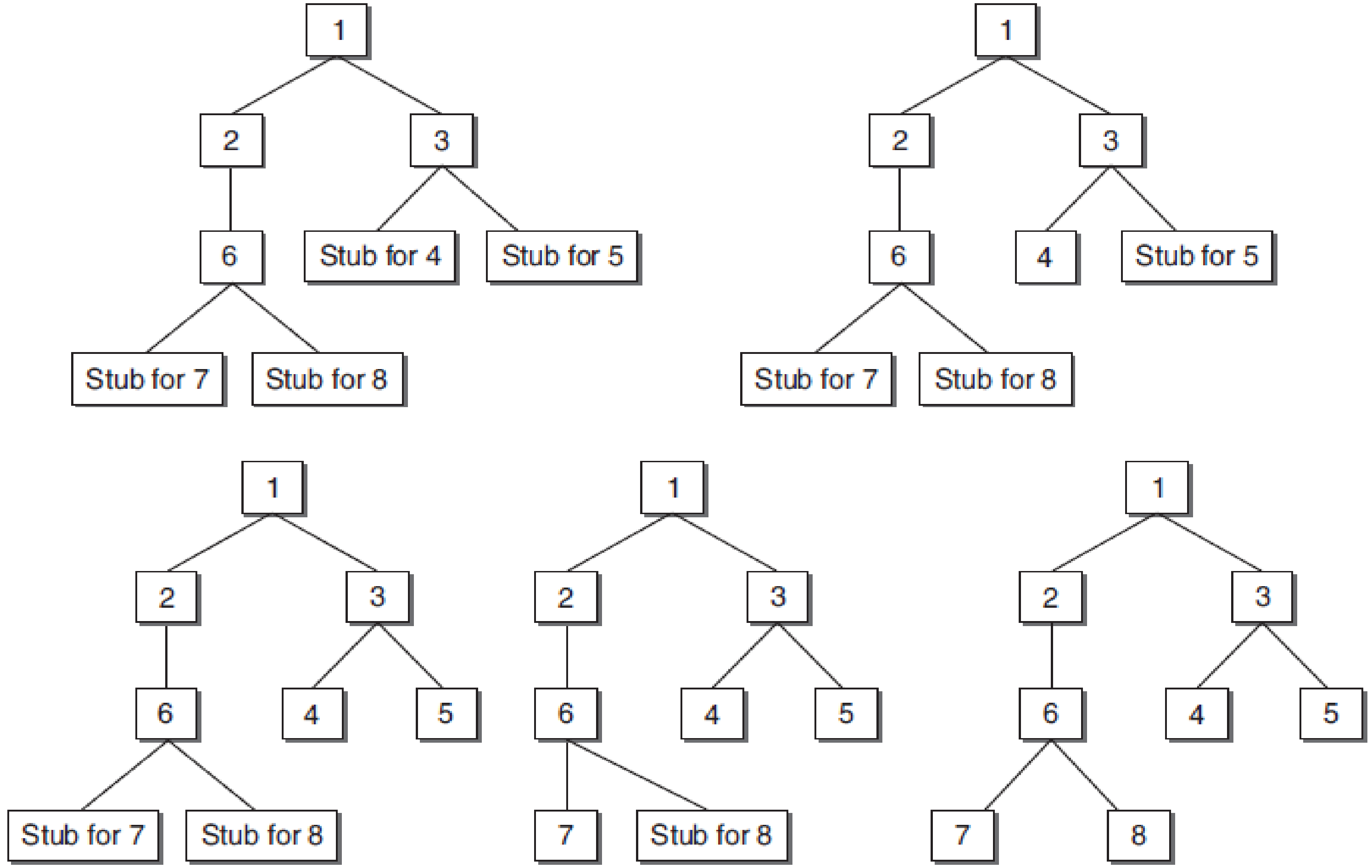

1
2
3
6
4
5
7
8


1
Stub for 2
Stub for 3


1
2
Stub for 3
Stub for 6


1
2
3
Stub for 6
Stub for 4
Stub for 5

Example cont…



Drawbacks of top-down integration

- Stubs must be prepared as required
- Stubs are often more complicated than they appear
- Before the I/O functions are added, the representation of test cases in stubs can be difficult

Bottom-up Integration Testing

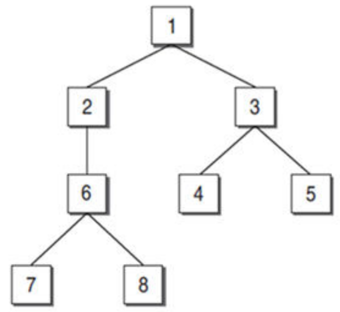
- The bottom-up strategy begins with the terminal or modules at the lowest level in the software structure. After testing these modules, they are integrated and tested moving from bottom to top level.
- Since the processing required for modules subordinate to a given level is always available, stubs are not required in this strategy.

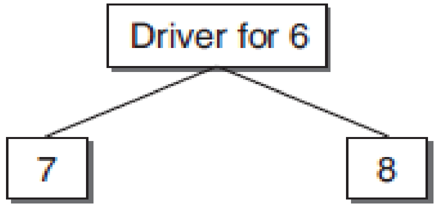
- Bottom-up integration can be considered as the opposite of top-down approach.
- Unlike top-down strategy, this strategy does not require the architectural design of the system to be complete.
- Thus, bottom-up integration can be performed at an early stage in the developmental process.
- It may be used where the system reuses and modifies components from other systems.

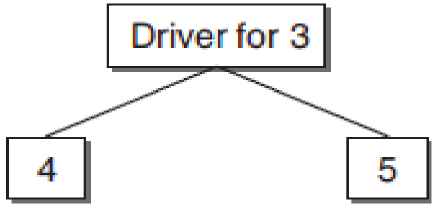
Steps in bottom-up integration

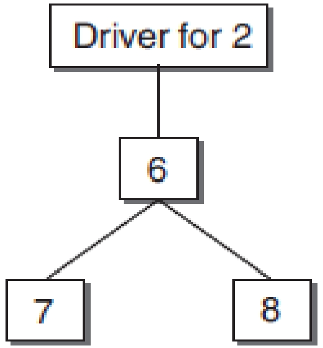
1. Start with the lowest level modules (modules from which no other module is being called), in the design hierarchy.
2. Look for the super-ordinate module which calls the module selected in Step 1. Design the driver module for this super-ordinate module.
3. Test the module selected in Step 1 with the driver designed in Step 2.
4. The next module to be tested is any module whose subordinate modules (modules it calls) have all been tested.
5. Repeat Steps 2 to 4 and move up in the design hierarchy.
6. Whenever, the actual modules are available, replace stubs and drivers with the actual one and test again.

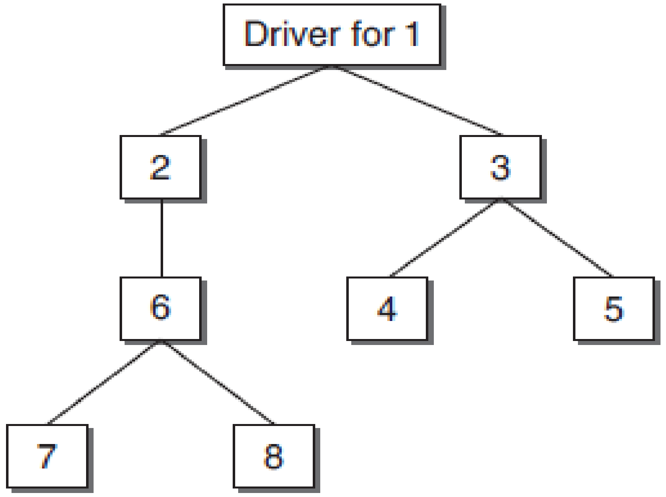
Example

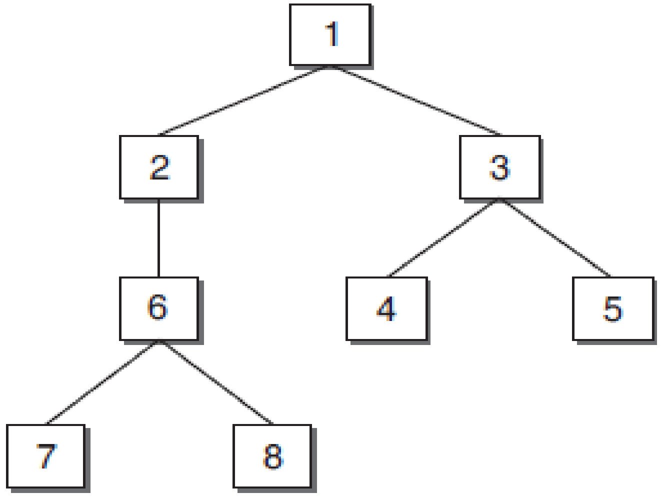

1
2
3
6
4
5
7
8


Driver for 6
7
8


Driver for 3
4
5


Driver for 2
6
7
8


Driver for 1
2
3
6
4
5
7
8


1
2
3
6
4
5
7
8

Comparison between top-down and bottom-up 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.

Practical Approach for Integration Testing

- There is no single strategy adopted for industry practice.
- For integrating the modules, one cannot rely on a single strategy.
- There are situations depending on 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**.

- Selection of an integration testing strategy depends on software characteristics and sometimes project schedules.
- In general, sandwich testing strategy that uses top-down tests for upper levels of the program structure with bottom-up tests for subordinate levels is the best compromise.
- The practical approach for adopting sandwich testing is driven by the following factors:

– Priority
  - First putting together those subsystems with more important requirements.
  - Follow top-down approach if the module has high level of control on its sub-ordinate modules.
  - Modules with more user interfaces should be tested first, as they are more error prone.
  - Module having high cyclomatic complexity should be tested first.
– Availability
  - The module that is ready to be integrated. Will be tested first.

Pros and Cons of Decomposition Tech.

- Debugging is easy in decomposition based integration
- Better for monitoring the progress of integration.
- But, more effort is required as stubs and drivers are needed.

Integration Testing Effort

- The integration testing effort is computed as the number of test sessions.
- A test session is one set of test cases for a specific configuration.
- The total number of test sessions in a decomposition-based integration is computed as:
- Number of test sessions = nodes - leaves + edges

Summary

- Discussed basics of different approaches for integration testing.
- Discussed decomposition-based integration in detail.
  - Big bang integration
  - Top-down integration
    - Breadth First Integration
    - Depth First Integration
  - Bottom-up integration
  - Sandwich integration
- Explained how to compute the integration testing effort .

References

1. Rajib Mall, Fundamentals of Software Engineering, (Chapter – 10), Fifth Edition, PHI Learning Pvt. Ltd., 2018.
2. Naresh Chauhan, Software Testing: Principles and Practices, (Chapter – 7), Second Edition, Oxford University Press, 2016.

Thank You

Integration Testing cont …

Prof. Durga Prasad Mohapatra
Professor
Dept.of CSE, NIT Rourkela

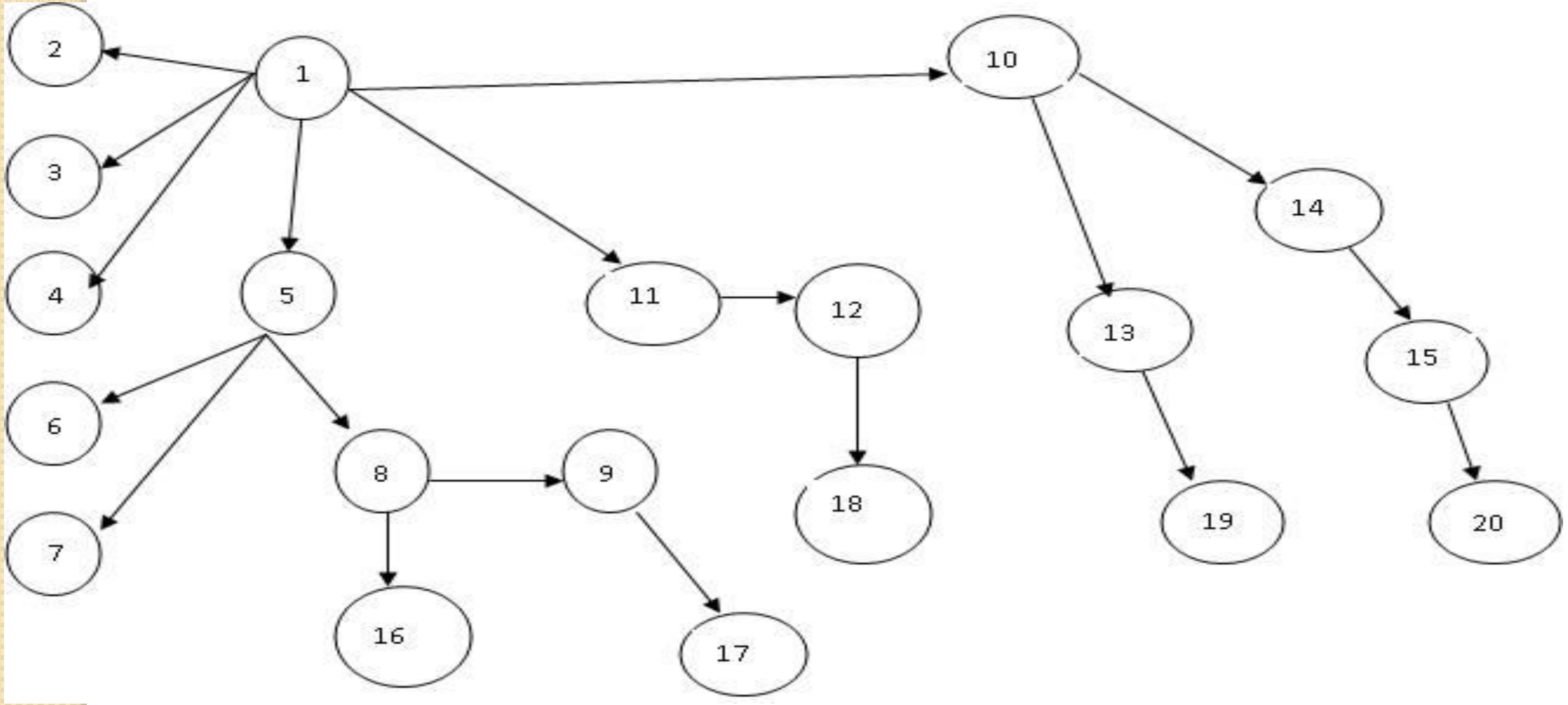
Call Graph-Based Integration

- It is assumed that integration testing detects bugs which are structural.
- However, it is also important to detect some behavioral bugs.
- If we can refine the functional decomposition tree into a form of module calling graph, then we are moving towards behavioral testing at the integration level.
- This can be done with the help of a *call graph* as given by Jorgensen.

Call Graph-Based Integration

- A call graph is a directed graph, wherein the nodes are either modules or units, and a directed edge from one node to another means one module has called another module.
- The call graph can be captured in a matrix form which is known as the adjacency matrix.

Example Call Graph



Adjacency Matrix

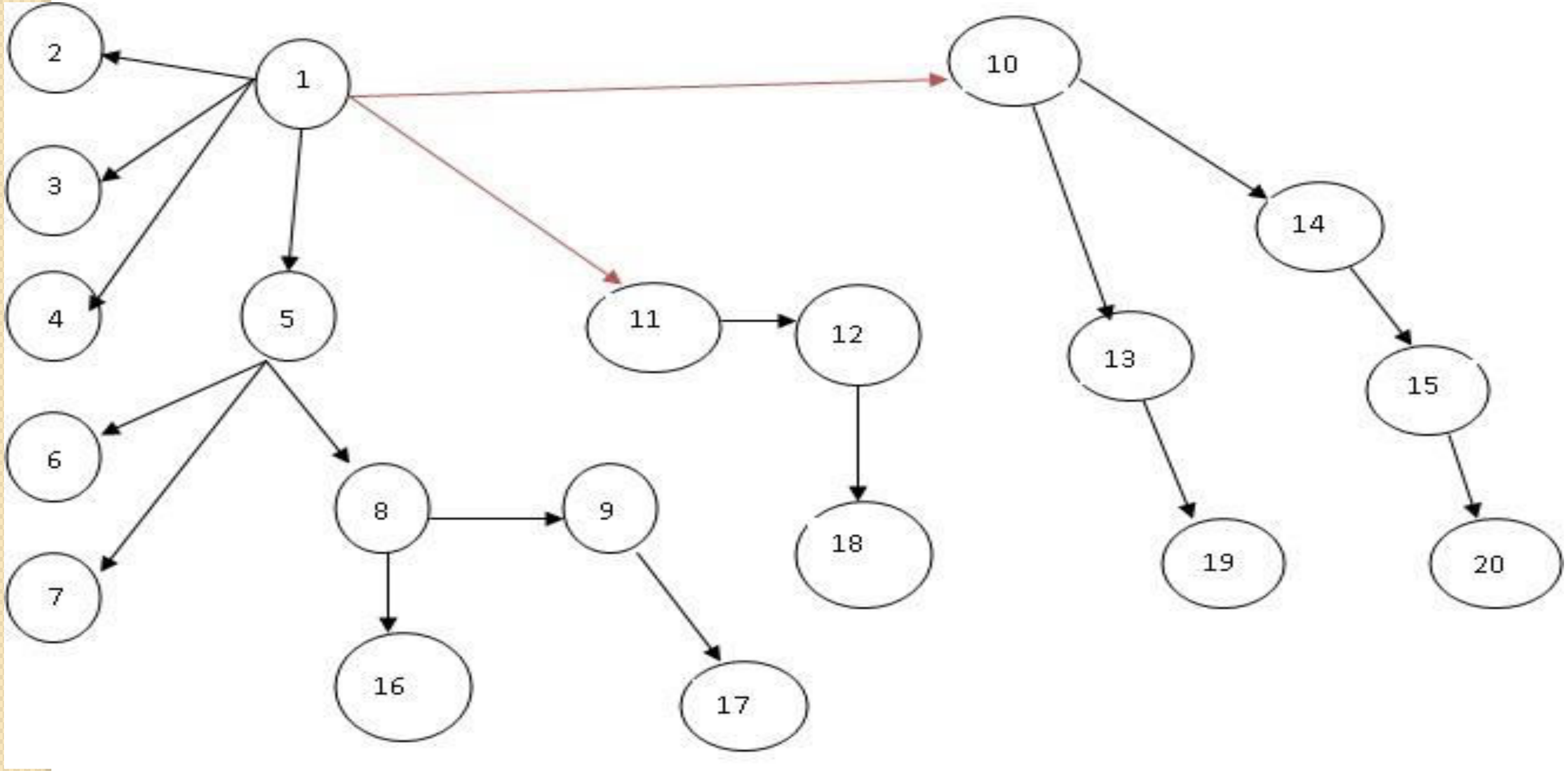
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	x	x	x	x					x	x									
2																			
3																			
4																			
5					x	x	x												
6																			
7																			
8								x							x				
9																x			
10												x	x						
11											x								
12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

•The idea behind using a call graph for integration testing is to avoid the efforts made in developing the stubs and drivers.

•If we know the calling sequence, and if we wait for the called or calling function, if not ready, then call graph based integration can be used.

- There are two types of integration testing based on call graph:
  - Pair-wise Integration
    - If we consider only one pair of calling and called modules, then we can make a set of pairs for all such modules.
    - The resulting set will be the total test sessions which will be equal to the sum of all edges in the call graph.
  - Neighborhood Integration

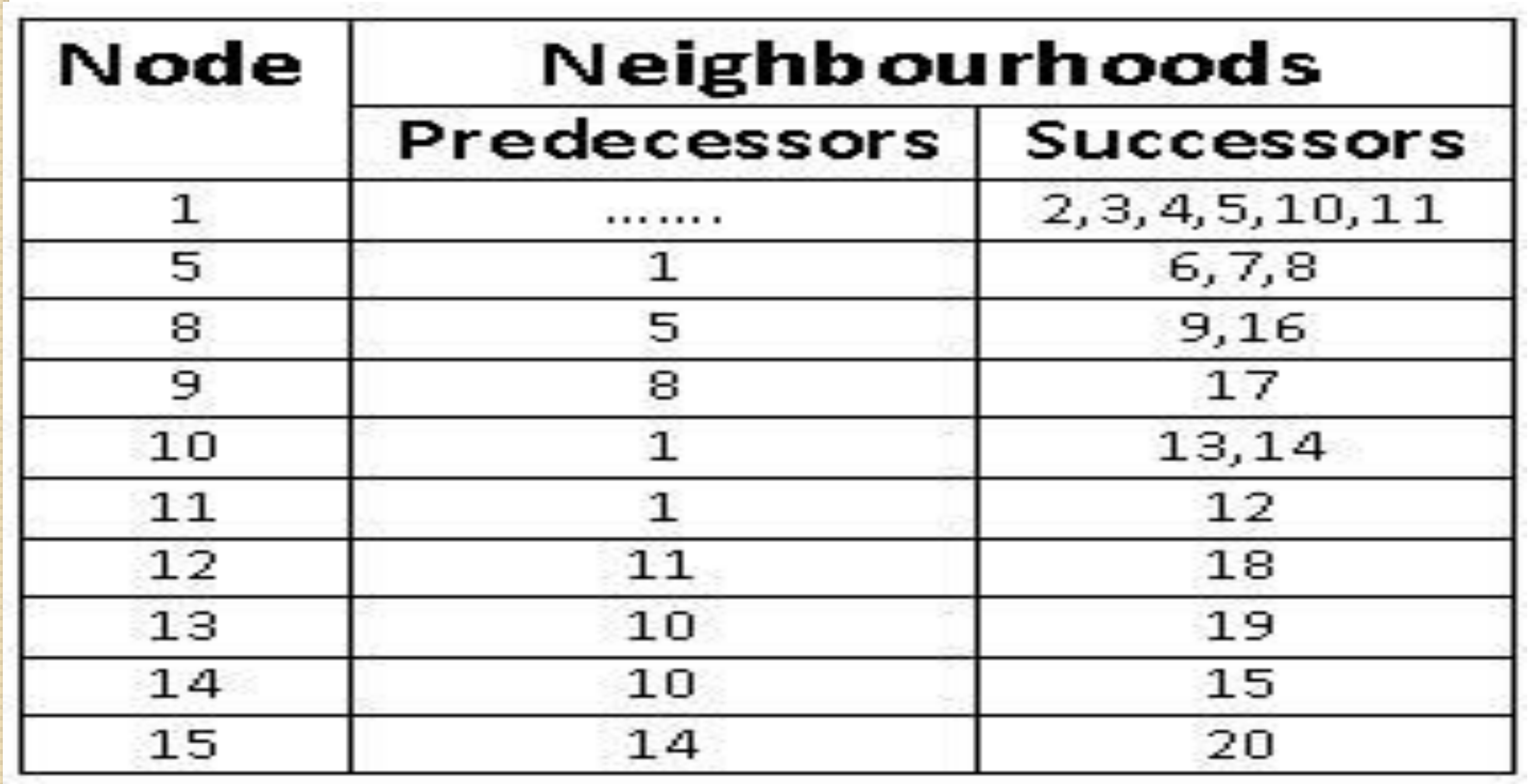
Pair wise Integration



Test sessions = No. of edges = 19

Neighborhood Integration

- There is not much reduction in the number of test sessions in pair-wise integration as compared to the decomposition-based integration.
- If we consider neighborhoods of a node in the call graph, then number of test sessions may reduce.
- Neighborhood of a node is the immediate predecessor as well as the immediate successor of the node.
- So, neighborhood of a node can be defined as the set of nodes that are one edge away from the given node



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 = nodes – sink nodes= **20 – 10 = 10**

sink node is an instruction in a module at which execution terminates.

Path Based Integration

- In call graph, when a module or unit executes, some path of the source instructions is executed.

- It is possible that in the path execution, there may be a call to another unit.

- At that point, the control is passed from the calling unit to the called unit.

Path Based Integration

- This passing of control from one unit to another unit is important for integration testing.
- It can be done with path based integration.
- We need to understand the following definitions for path-based integration:

- –Source node
  - It is an instruction in the module at which the execution starts or resumes.
  - The nodes where the control is being transferred after calling the module are also source nodes.
- –Sink node
  - It is an instruction in the module at which the execution terminates.
  - The nodes from which the control is transferred are also sink nodes.
- –Module execution path (MEP)
  - It is a path consisting of set of executable statements within a module like in a flow graph.

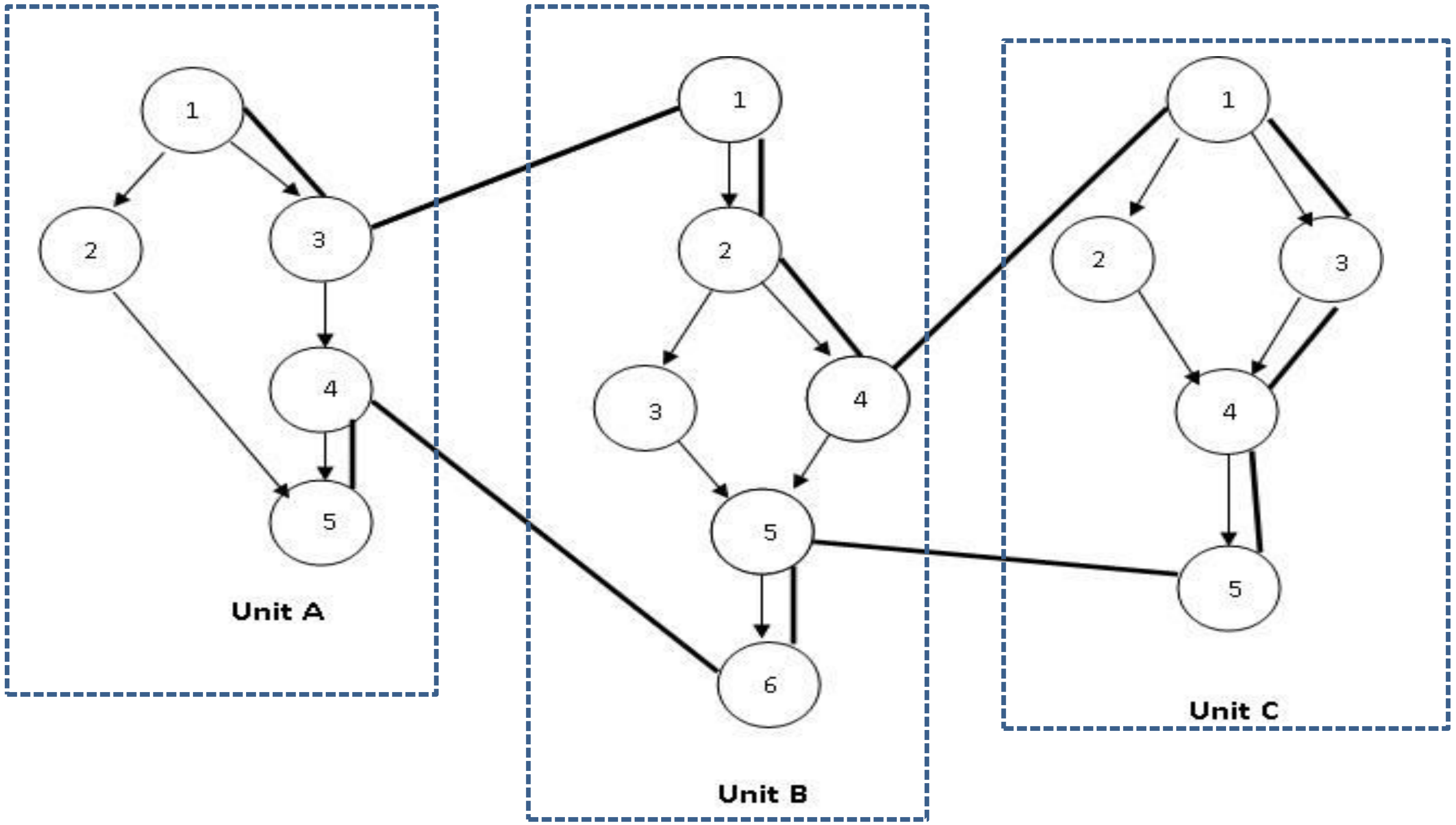
– Message
  - When the control is transferred from one unit to another, then the programming language mechanism used to do this is known as a message.
  - For example, a function call (message from one unit to another unit)

– MM-path
  - It is a path consisting of MEPs and messages.
  - The path shows the sequence of executable statements; it also crosses the boundary of a unit when a message is followed to call another unit.
  - In other words, MM-path is a set of MEPs & transfer of control among different units in the form of messages.

– MM-Path Graph

- It can be defined as an extended flow graph where nodes are MEPs and edges are messages.
- It returns from the last called unit to the first unit where the call was made.
- In this graph (shown in next slide), messages are highlighted with thick lines.

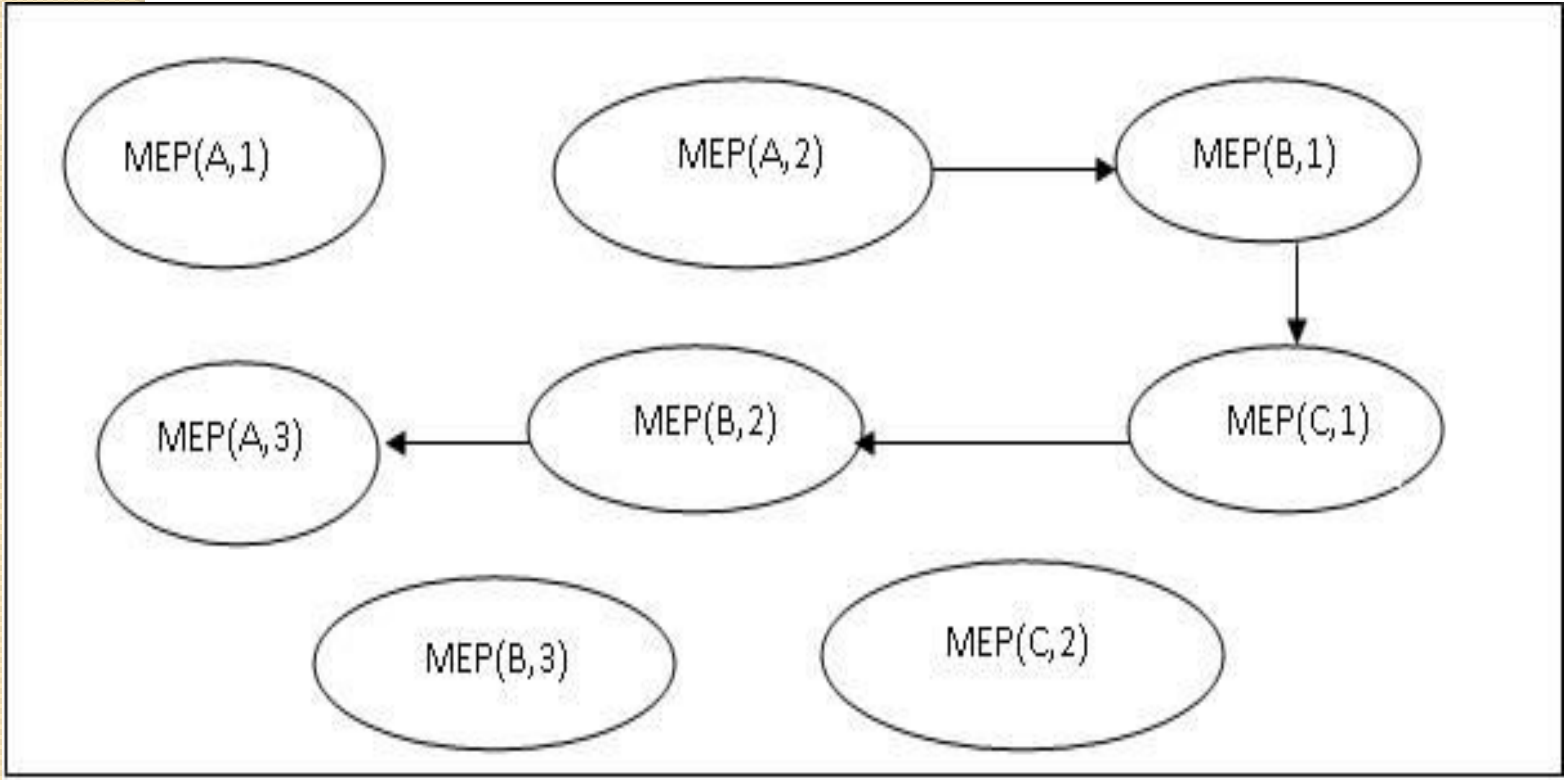
MM-Path



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)=<1,2,3,5,6>
Unit C	1	5	MEP(C,1)=<1,3,4,5> MEP(C,2)=<1,2,4,5>

MM-Path Graph



Function Testing

- When an integrated system is tested, all its specified functions and external interfaces are tested on the software.
- Every functionality of the system specified in the functions is tested according to its external specifications.
- An external specification is a precise description of the software behavior from the viewpoint of the outside world.

- Kit has defined function testing as the process of attempting to detect discrepancies between the functional specifications of a software and its actual behavior.
- The objective of the function test is to measure the quality of the functional components of the system.
- Tests verify that the system behaves correctly from the user/business perspectives and functions according to the requirements, models or any other design paradigm.

- The function test must determine if each component or business event:

  - Performs in accordance to the specifications,
  - Responds correctly to all conditions that may present themselves by incoming events/data,
  - Moves data correctly from one business event to the next, and
  - Is initiated in the order required to meet the business objectives of the system

- Function testing can be performed after unit and integration testing, or whenever the development team thinks that the system has sufficient functionality to execute some tests.
- The test cases are executed such that the execution of the given test case against the software will exercise external functionality of certain parts.
- To keep a record of function testing, a function coverage metric is used.

- Function coverage can be measured with a function coverage matrix.
- It keeps track of those functions that exhibited the greatest number of errors.
- An effective function test cycle must have a defined set of processes and deliverables.
- The primary processes/ deliverables for requirement based function test are as follows:

– Test Planning
  - The test leader with assistance from the test team defines the scope, schedule, and deliverables for the function test cycle.
  - He delivers a test plan and a test schedule- these often undergo several revisions during the testing cycle.

– Partitioning/ functional Decomposition
  - It is the process of breaking down a system into its functional components or functional areas.
  - Another group in the organization takes responsibility for the functional decomposition of the system.
  - If decompositions are deemed insufficient, then testing organization takes up the responsibility of decomposition.

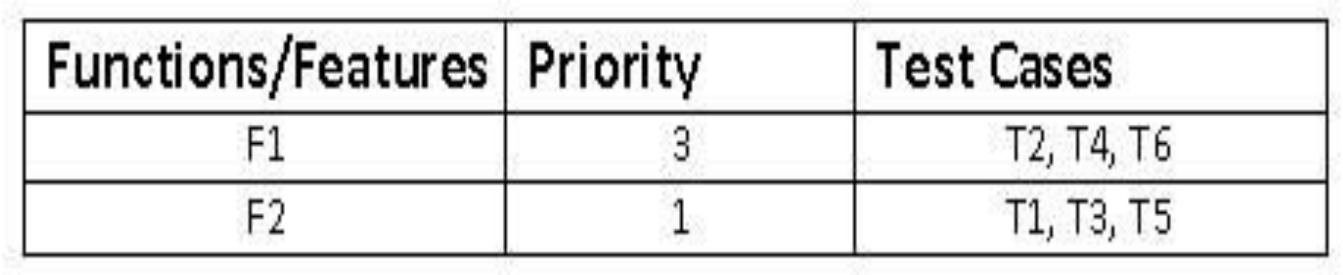
– Requirement definition
  - The testing organization needs specified requirements in the form of proper documents to proceed with the function test.
  - These requirements need to be itemized under an appropriate functional partition.

– Test case design
  - A tester designs and implements a test case to validate that the product performs in accordance with the requirements.
  - These requirements need to be itemized under an appropriate functional partition and mapped to the requirements being tested.

– Traceability matrix formation

- Test cases need to be mapped back to the appropriate requirement.
- A function coverage matrix is prepared.
- This matrix is a table, listing specific functions to be tested, the priority for testing each function, and test cases required to test each function.
- Once all the aspects of the function have been tested by one or more test cases, then the test design activity for that function can be considered complete.

Function coverage matrix



Functions/Features	Priority	Test Cases
F1	3	T2, T4, T6
F2	1	T1, T3, T5

–Test case execution

- In all phases of testing, an appropriate set of test cases need to be executed and the result of those test cases recorded.
- So, the test case to be executed should be defined in the test plan.
- If the current application does not support the testing, then it should be deferred.

Summary

- Discussed call graph-based integration testing in detail.
  - Pair-wise Integration
  - Neighborhood Integration
- Explained path-based integration testing.
- Explained Function Testing in detail.
- Described the primary processes/ deliverables for requirement based function test.

References

1. Rajib Mall, Fundamentals of Software Engineering, (Chapter – 10), Fifth Edition, PHI Learning Pvt. Ltd., 2018.
2. Naresh Chauhan, Software Testing: Principles and Practices, (Chapter – 7), Second Edition, Oxford University Press, 2016.

Thank You