Integration Testing
Path Based

Chapter 13

Call graph based integration

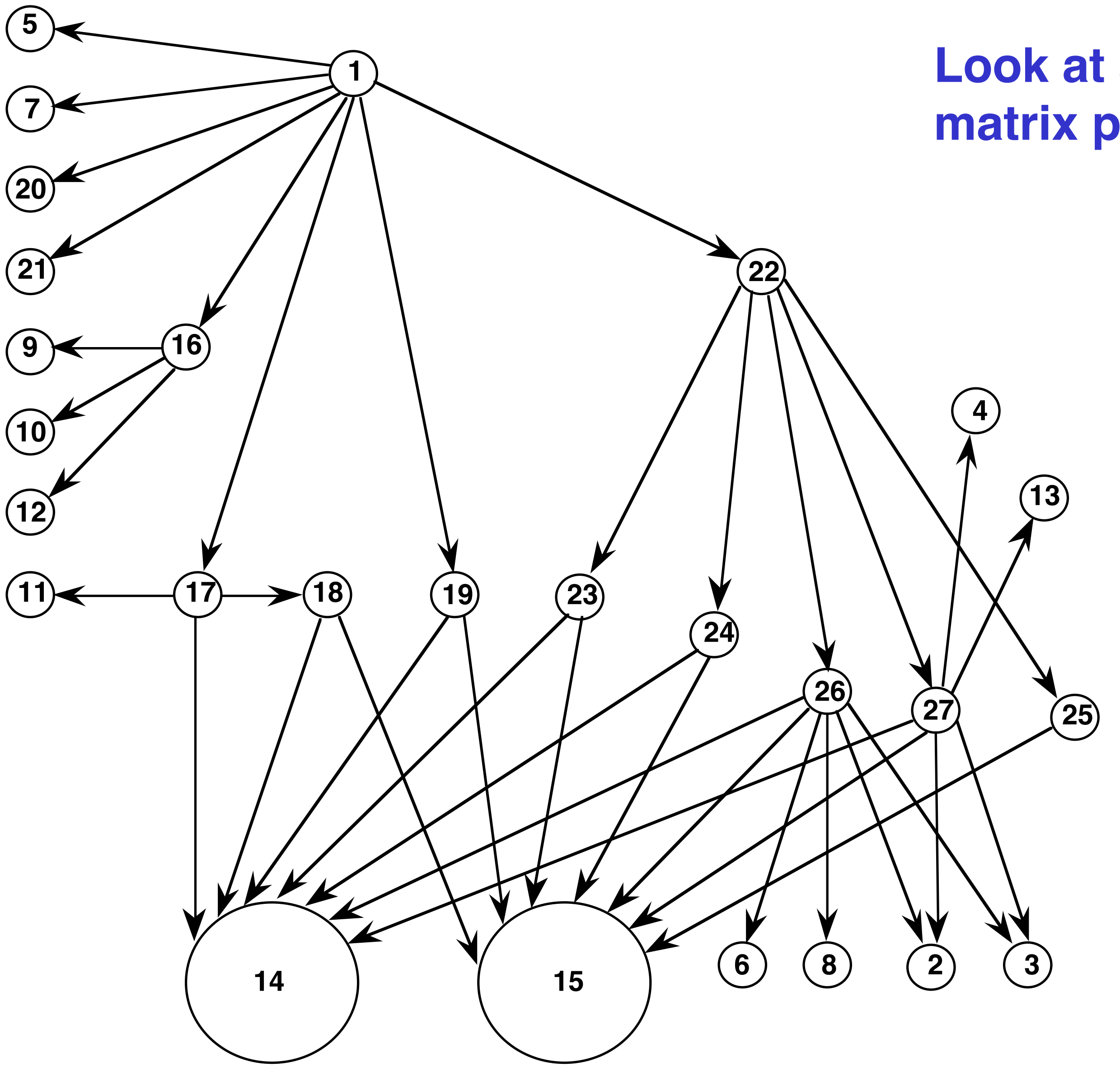
- Use the call graph instead of the decomposition tree

- **What is a call graph?**

Call graph definition

- Is a directed, labeled graph
  - **Vertices are methods**
  - **A directed edge joins calling vertex to the called vertex**
  - **Adjacency matrix is also used**
  - **Does not scale well, although some insights are useful**
    - **Nodes of high degree are critical**

SATM call graph example



Look at adjacency matrix p204

Call graph integration strategies

- What types of integration strategies are used?

Call graph integration strategies – 2

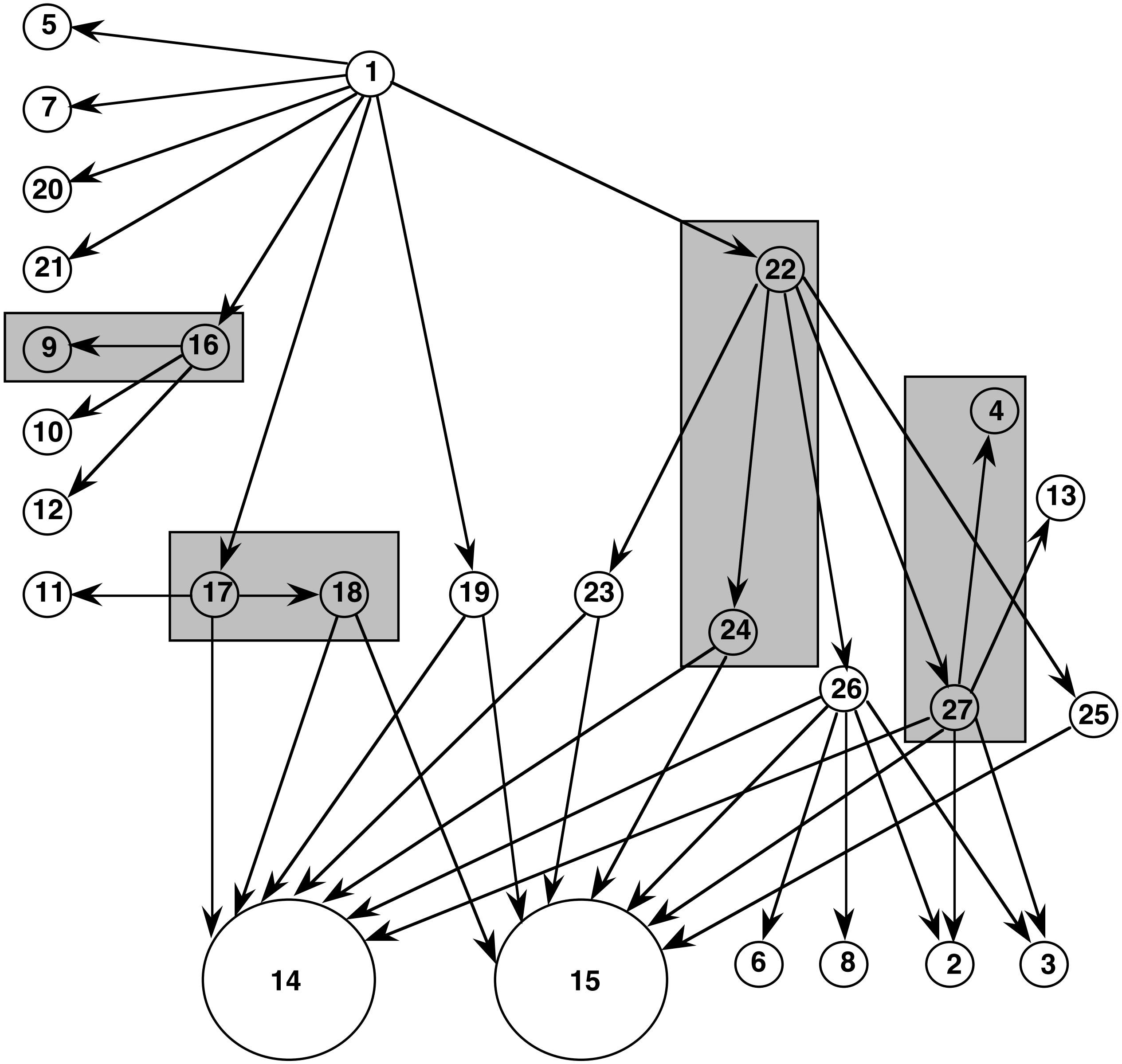
- Pair-wise Integration Testing

- Neighborhood Integration Testing

Pair-wise integration

- **What is pair-wise integration**

Pair-wise integration session example



Pair-wise integration – 2

- The idea behind Pair-Wise integration testing
  - **Eliminate need for developing stubs / drivers**
  - **Use actual code instead of stubs/drivers**

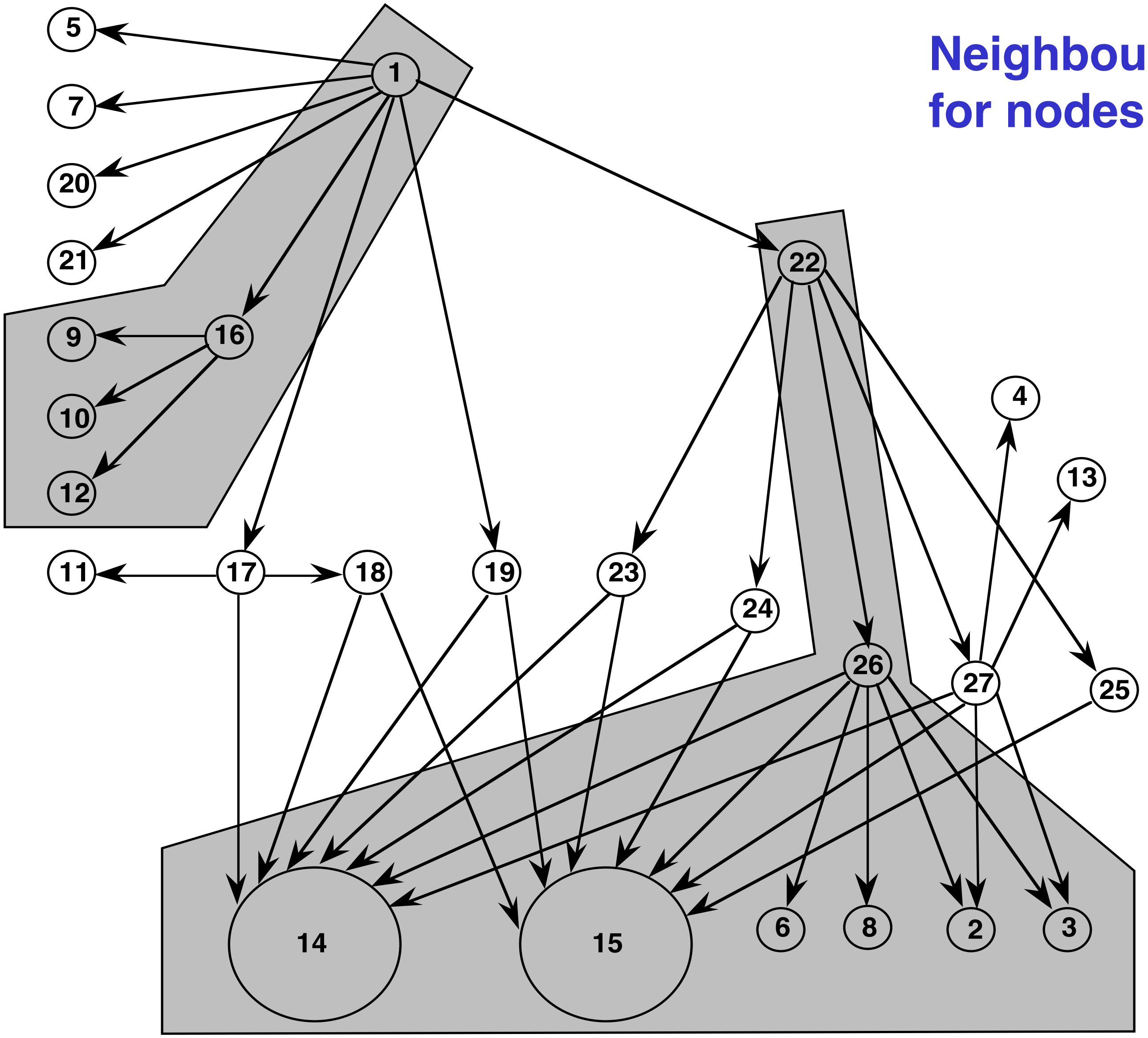
Pair-wise integration – 3

- In order not to deteriorate the process to a big-bang strategy
  - **Restrict a testing session to just a pair of units in the call graph**
  - **Results in one integration test session for each edge in the call graph**

Neighbourhood integration

- **What is neighbourhood integration?**

Neighbourhood integration example



Neighbourhoods for nodes 16 & 26

Neighbourhood integration – 2

- The neighbourhood of a node in a graph
  - **The set of nodes that are one edge away from the given node**

- In a directed graph
  - **All the immediate predecessor nodes and all the immediate successor nodes of a given node**

Neighbourhood integration – 3

- Neighborhood integration testing
  - **Reduces the number of test sessions**
  - **Fault isolation is difficult**

Pros of call-graph integration

- **What are the pros of call-graph integration?**

Pros of call-graph integration – 2

- Reduces the need for drivers and stubs
  - **Relative to functional decomposition integration**

- Neighborhoods can be combined to create "villages"

- Closer to a build sequence
  - **Well suited to devising a sequence of builds with which to implement a system**

Cons of call-graph integration

- What are the cons of call-graph integration?

Cons of call-graph integration – 2

- Suffers from fault isolation problems
  - **Especially for large neighborhoods**

- Redundancy
  - **Nodes can appear in several neighborhoods**

- Assumes that correct behaviour follows from correct units and correct interfaces
  - **Not always the case**

Path-based integration

- What is path-based integration?

- Why use it?

Path-Based Integration – 2

- Motivation
  - **Combine structural and behavioral type of testing for integration testing as we did for unit testing**
- Basic idea
  - **Focus on interactions among system units**
  - **Rather than merely to test interfaces among separately developed and tested units**
- Interface-based testing is structural while interaction-based testing is behavioral

Source node

- **What is it?**

Source node – 2

- A program statement fragment at which program execution begins or resumes.

  - **For example the first “begin” statement in a program.**

  - **Nodes immediately after nodes that transfer control to other units.**

Sink node

- What is a sink node?

Sink node

- A statement fragment at which program execution terminates
  - **The final “end” in a program as well as statements that transfer control to other units**

Module execution path (MEP)

- What is a module execution path?

Module execution path (MEP) – 2

- A sequence of statements within a module that
  - **Begins with a source node**
  - **Ends with a sink node**
  - **With no intervening sink nodes**

Message

- **What is a message?**

Message – 2

- A programming language mechanism by which one unit transfers control to another unit

- Usually interpreted as subroutine / function invocations

- The unit which receives the message always returns control to the message source

MM-path

- What is an MM-path?

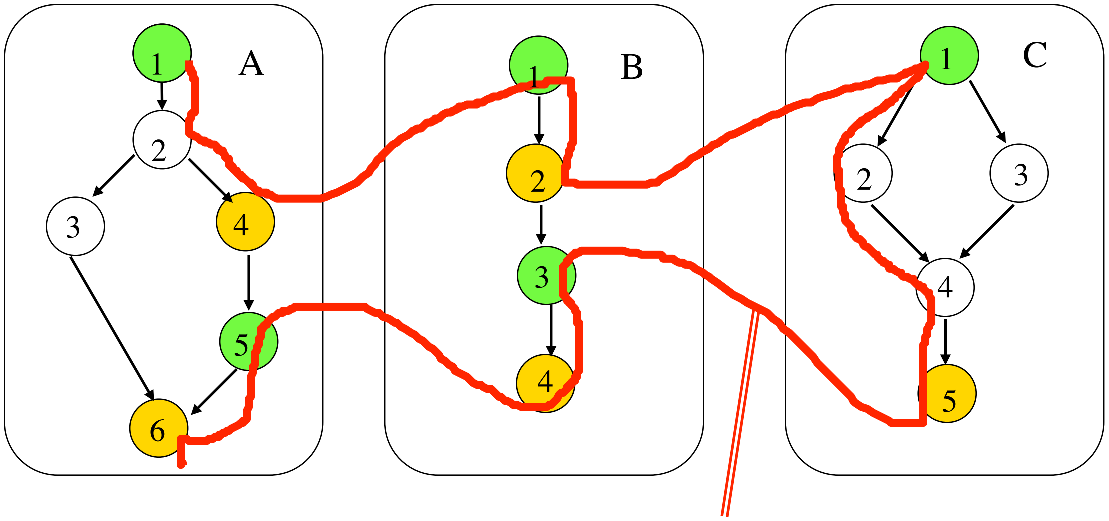
MM-path – 2

- A module to module path
  - **An interleaved sequence of module execution paths and messages**

- Used to describes sequences of module execution paths that include transfers of control among separate units

- MM-paths always represent feasible execution paths, and these paths cross unit boundaries

MM-path example



MM-path

🟢 Source nodes

🟡 Sink nodes

Module Execution Paths

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

MEP(B,1) = <1, 2>
MEP(B,2) = <3, 4>

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

MEPs and DD-paths

- What is the correspondence between MEPs and a DD-paths?

MEPs and DD-paths – 2

- There is no correspondence between MM execution paths and DD-paths

MEPs and slices

- **What is the correspondence between MEPs and slices?**

MEPs and slices – 2

- There is no correspondence but there is an analog
  - **The intersection of a module execution path with a unit is the analog of a slice with respect to the MM-path function**

MM-path graph

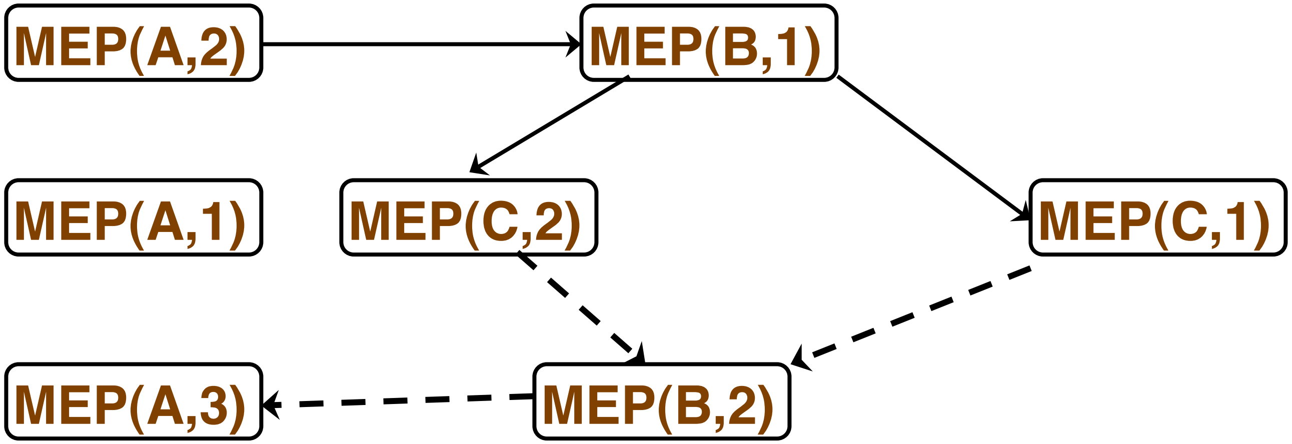
- What is an MM-path graph?

MM-path graph – 2

- Given a set of units their **MM-path graph** is the directed graph in which
  - **Nodes are module execution paths**
  - **Edges correspond to messages and returns from one unit to another**

- The definition is with respect to a set of units
  - **It directly supports composition of units and composition-based integration testing**

MM-path graph example



Solid lines indicate messages (calls)
Dashed lines indicate returns from calls

MM-path guidelines

- How long, or deep, is an MM-path? What determines the end points?

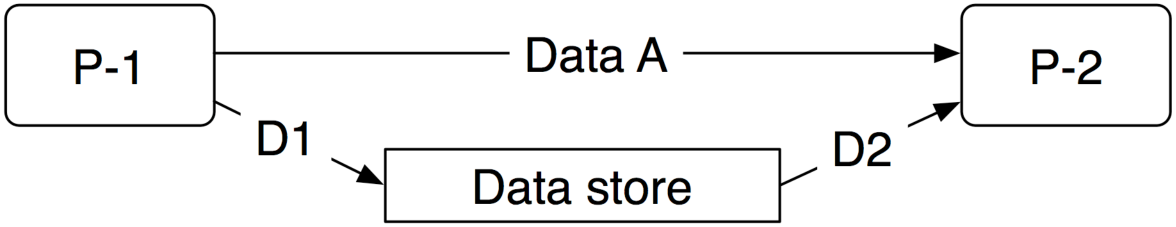
- Quiescence points are natural endpoints for MM-paths
  - **Message quiescence**
  - **Data quiescence**

Message quiescence

- Occurs when a unit that sends no messages is reached
  - **Module C in the example**

Data quiescence

- Occurs when a sequence of processing ends in the creation of stored data that is not immediately used
  - **The causal path Data A has no quiescence**
  - **The non-causal path D1 and D2 is quiescent at the node P-1**



MM-path metric

- What is the minimum number of MM-paths that are sufficient to test a system?

MM-Path metric – 2

- What is the minimum number of MM-paths that are sufficient to test a system?
  - Should cover all source-to-sink paths in the set of units

- What about loops? How should they be treated?

MM-Path metric – 3

- What is the minimum number of MM-paths that are sufficient to test a system?
  - Should cover all source-to-sink paths in the set of units
- What about loops? How should they be treated?
  - Use condensation graphs to get directed acyclic graphs
    - Avoids an excessive number of paths

Pros of path-based integration

- Benefits of hybrid of functional and structural testing
  - **Functional – represent actions with input and output**
  - **Structural – how they are identified**
- Avoids pitfall of structural testing
  - **Unimplemented behaviours cannot be tested**
- Fairly seamless union with system testing

Pros of path-based integration – 2

- Path-based integration is closely coupled with actual system behaviour
  - **Works well with OO testing**

- No need for stub and driver development

Cons of path-based integration

- There is a significant effort involved in identifying MM-paths

MM-path compared to other methods

Strategy	Ability to test interfaces	Ability to test co-functionality	Fault isolation resolution
Functional decomposition	Acceptable, can be deceptive	Limited to pairs of units	Good to faulty unit
Call-graph	Acceptable	Limited to pairs of units	Good to faulty unit
MM-path	Excellent	Complete	Excellent to unit path level