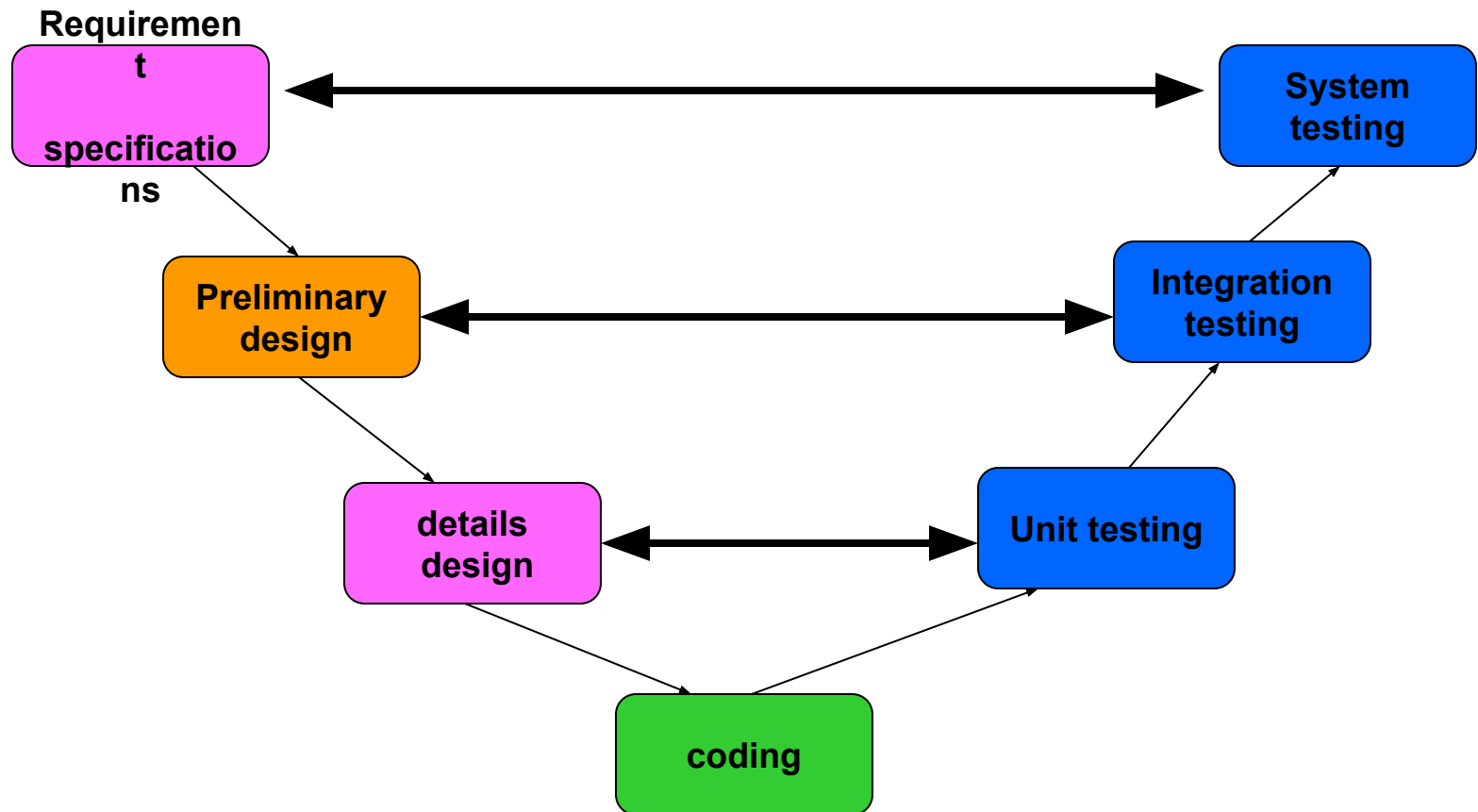




Integration testing

Testing Types: Integration Testing

The waterfall life cycle





Goals/Purpose of Integration Testing

- Presumes previously tested units
- Not system testing
(at level of system inputs and outputs)
- tests functionality "between" unit and system levels
- basis for test case identification?
- Emphasis shifts from “how to test” to “what to test”



Approaches to Integration Testing

1. Based on Functional Decomposition

Top-Down
Bottom-Up
Sandwich
Big Bang
Incremental

2. Based on Call Graph

Pair-wise
Neighborhood

3. Based on Paths

MM-Paths
Atomic System Functions

Steps in Integration-Testing

1. Based on the integration strategy, *select a component* to be tested. Unit test all the classes in the component.
 2. Put selected component together; do any *preliminary fix-up* necessary to make the integration test operational (drivers, stubs)
 3. Do *functional testing*: Define test cases that exercise all uses cases with the selected component
 4. Do *structural testing*: Define test cases that exercise the selected component
 5. Execute *performance tests*
 6. *Keep records* of the test cases and testing activities.
 7. Repeat steps 1 to 7 until the full system is tested.
- The primary *goal of integration testing is to identify errors* in the (current) component configuration.

Which Integration Strategy should you use?

■ Factors to consider

- Amount of test harness (stubs & drivers)
- Location of critical parts in the system
- Availability of hardware
- Availability of components
- Scheduling concerns

■ Bottom up approach

- good for object oriented design methodologies
- Test driver interfaces must match component interfaces
- ...

- ...Top-level components are usually important and cannot be neglected up to the end of testing
- Detection of design errors postponed until end of testing

■ Top down approach

- Test cases can be defined in terms of functions examined
- Need to maintain correctness of test stubs
- Writing stubs can be difficult



Running example: Simple ATM system

- An ATM:
 - Provides 15 screens for interactions
 - includes 3 function buttons

Screen 1

Welcome.

Please Insert your
ATM card for service

Screen 2

Enter your Personal
Identification Number

— — — — —
Press Cancel if Error

Screen 3

Your Personal
Identification Number
is incorrect. Please
try again.

Screen 4

Invalid identification.
Your card will be
retained. Please call
the bank.

Screen 5

Select transaction type:
balance
deposit
withdrawal
Press Cancel if Error

Screen 6
Select account type:

checking
savings

Press Cancel if Error

Screen 7

Enter amount.
Withdrawals must be
in increments of \$10

— — — — . — —
Press Cancel if Error

Screen 8

Insufficient funds.
Please enter a new
amount.

— — — — . — —
Press Cancel if Error

Screen 9

Machine cannot
dispense that amount.

Please try again.

Screen 10

Temporarily unable to
process withdrawals.
Another transaction?
yes
no

Screen 11

Your balance is being
updated. Please take
cash from dispenser.

Screen 12

Temporarily unable to
process deposits.
Another transaction?
yes
no

Screen 13

Please put envelope into
deposit slot. Your
balance will be updated
Press Cancel if Error.

Screen 14

Your new balance is
printed on your receipt.
Another transaction?
yes
no

Screen 15

Please take your
receipt and ATM
card. Thank you.

WELCOME
to the

Simple
Automatic Teller
Machine

Please Insert your
card for service

Receipts

ID Card

B1

B2

B3

1

2

3

4

5

6

7

8

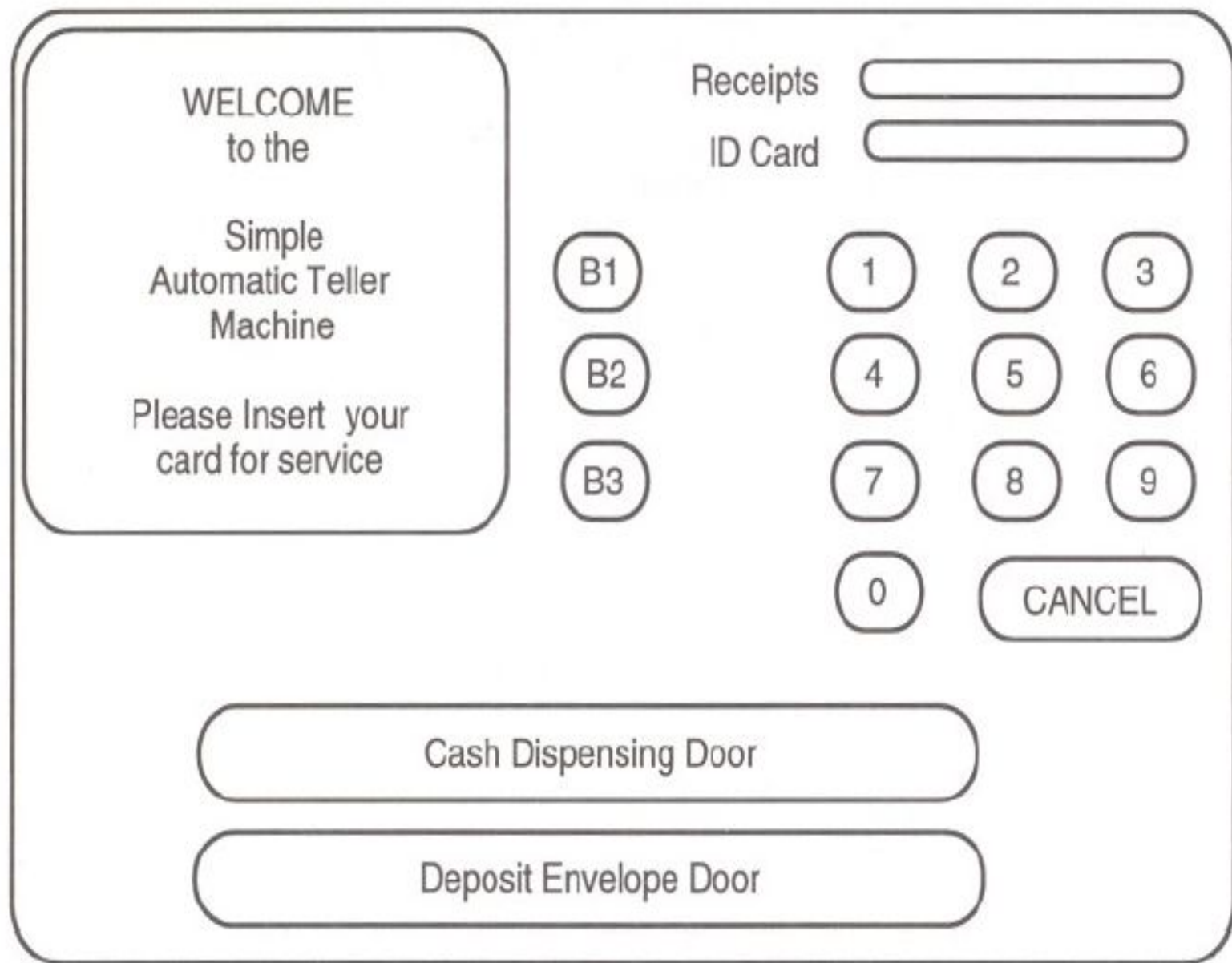
9

0

CANCEL

Cash Dispensing Door

Deposit Envelope Door



The diagram illustrates the layout of a Simple Automatic Teller Machine (SATM) terminal. It features a main display area on the left with a welcome message and instructions. To the right of the display are two horizontal slots for 'Receipts' and 'ID Card'. Below these are three circular buttons labeled 'B1', 'B2', and 'B3'. Further right is a numeric keypad with buttons for digits 1 through 9, 0, and a 'CANCEL' button. At the bottom of the terminal are two large, rounded rectangular buttons labeled 'Cash Dispensing Door' and 'Deposit Envelope Door'.

Figure 12.8 The SATM Terminal

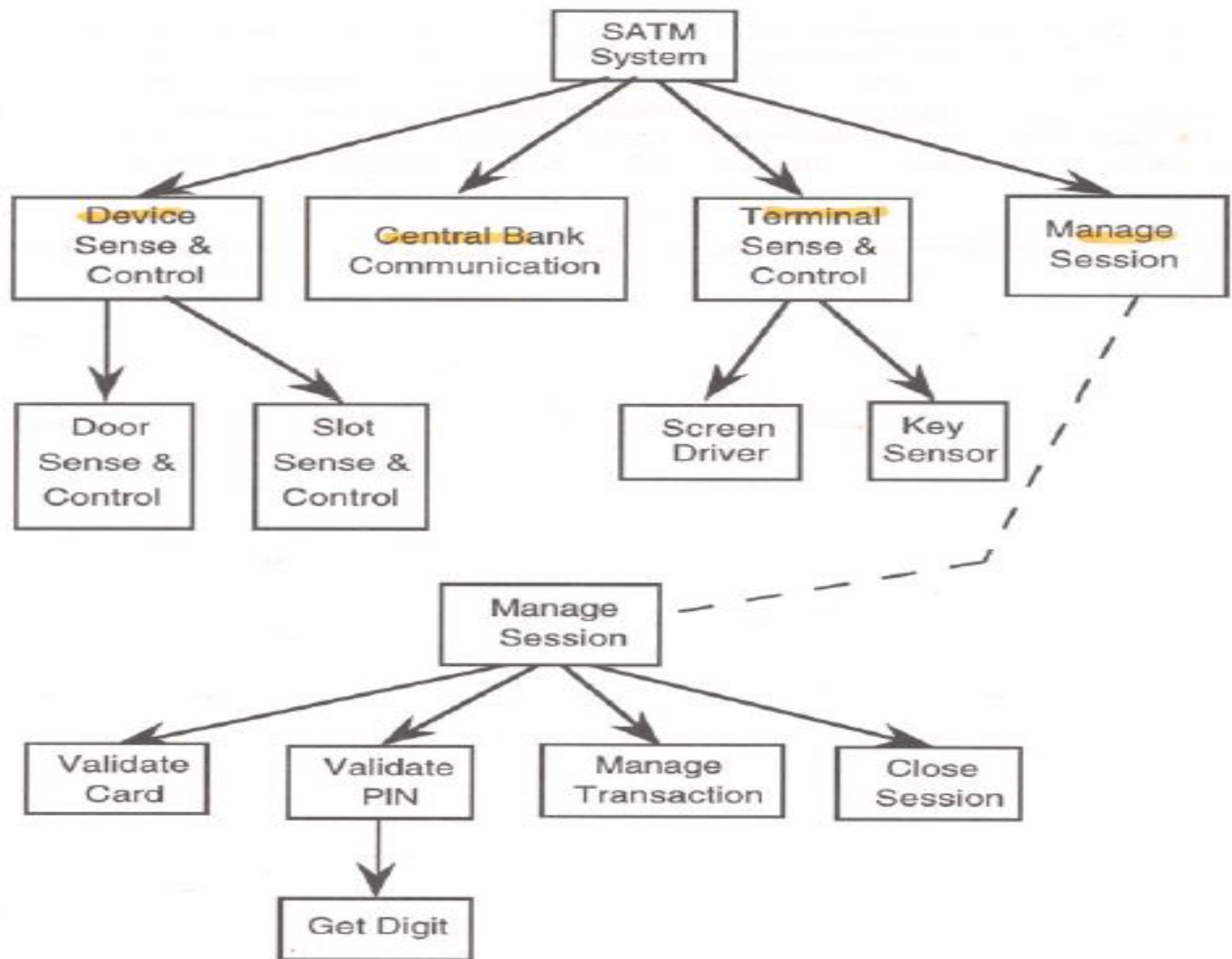


Figure 12.14 A Decomposition Tree for the SATM System

Functional Decomposition of the SATM System

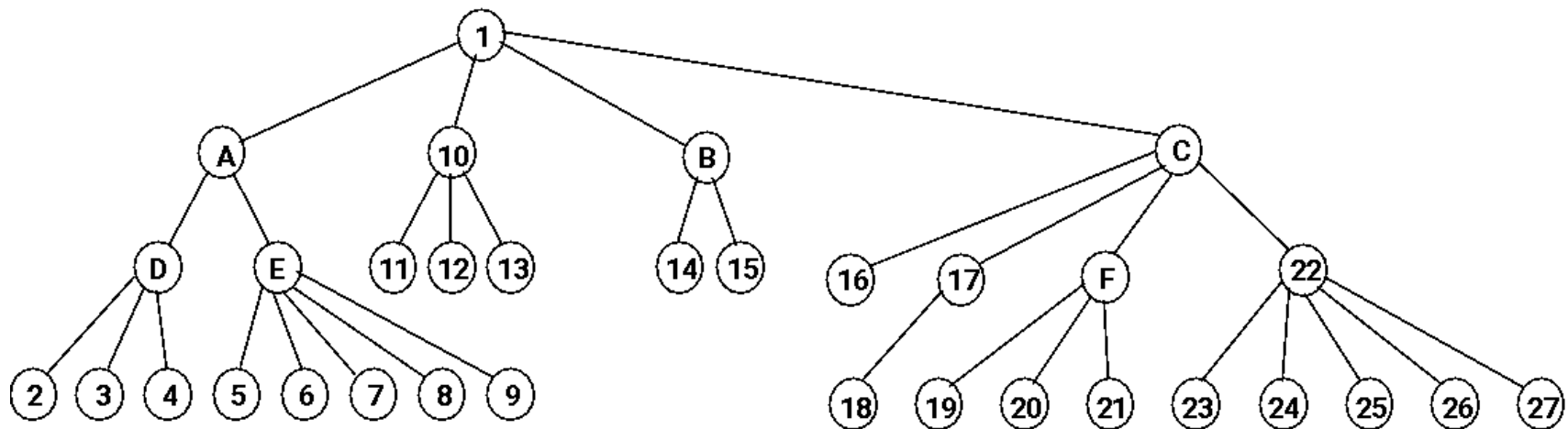


Table 1: SATM Units and Abbreviated Names

Unit	Level	Unit Name
1	1	SATM System
A	1.1	Device Sense & Control
D	1.1.1	Door Sense & Control
2	1.1.1.1	Get Door Status
3	1.1.1.2	Control Door
4	1.1.1.3	Dispense Cash
E	1.1.2	Slot Sense & Control
5	1.1.2.1	WatchCardSlot
6	1.1.2.2	Get Deposit Slot Status
7	1.1.2.3	Control Card Roller
8	1.1.2.3	Control Envelope Roller
9	1.1.2.5	Read Card Strip
10	1.2	Central Bank Comm.
11	1.2.1	Get PIN for PAN
12	1.2.2	Get Account Status
13	1.2.3	Post Daily Transactions

Unit	Level	Unit Name
B	1.3	Terminal Sense & Control
14	1.3.1	Screen Driver
15	1.3.2	Key Sensor
C	1.4	Manage Session
16	1.4.1	Validate Card
17	1.4.2	Validate PIN
18	1.4.2.1	GetPIN
F	1.4.3	Close Session
19	1.4.3.1	New Transaction Request
20	1.4.3.2	Print Receipt
21	1.4.3.3	Post Transaction Local
22	1.4.4	Manage Transaction
23	1.4.4.1	Get Transaction Type
24	1.4.4.2	Get Account Type
25	1.4.4.3	Report Balance
26	1.4.4.4	Process Deposit
27	1.4.4.5	Process Withdrawal



Decomposition based strategies

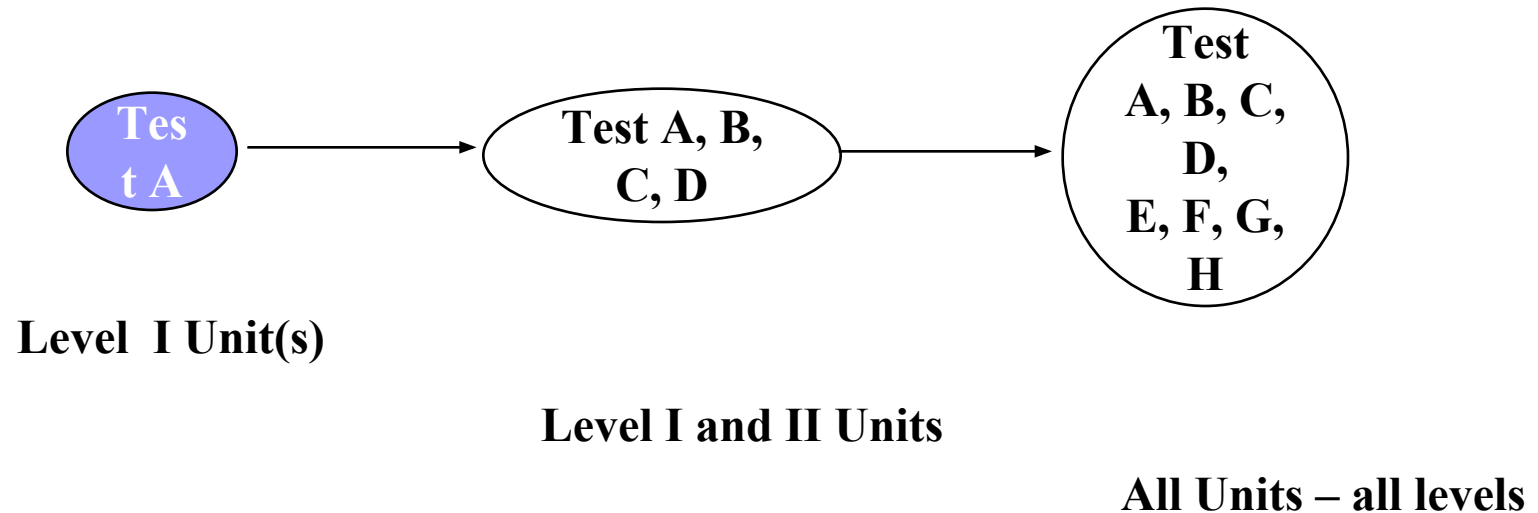
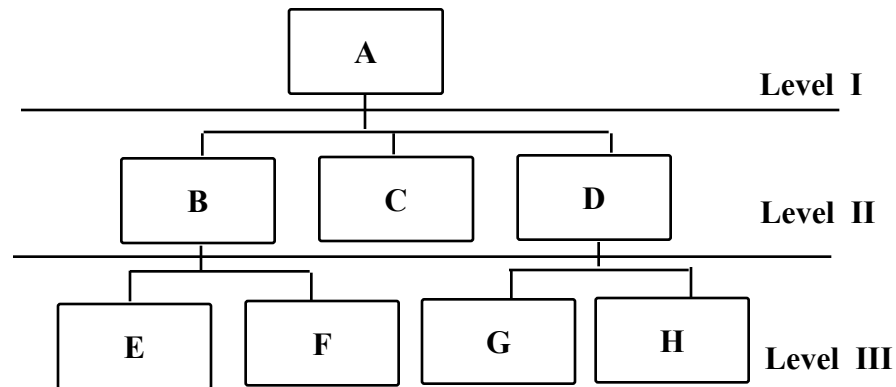
- Decomposition based:

- ☐ Top/down
- ☐ Bottom up
- ☐ Sandwich
- ☐ Big bang

Top-Down Integration Testing Strategy

- Top-down integration strategy focuses on testing the top layer or the controlling subsystem first (i.e. the *main*, or the root of the call tree)
- The general process in top-down integration strategy is to gradually add more subsystems that are referenced/required by the already tested subsystems when testing the application
- Do this until all subsystems are incorporated into the test
- Special program is needed to do the testing, *Test stub*:
 - A program or a method that simulates the input-output functionality of a missing subsystem by answering to the decomposition sequence of the calling subsystem and returning back simulated or “canned” data.

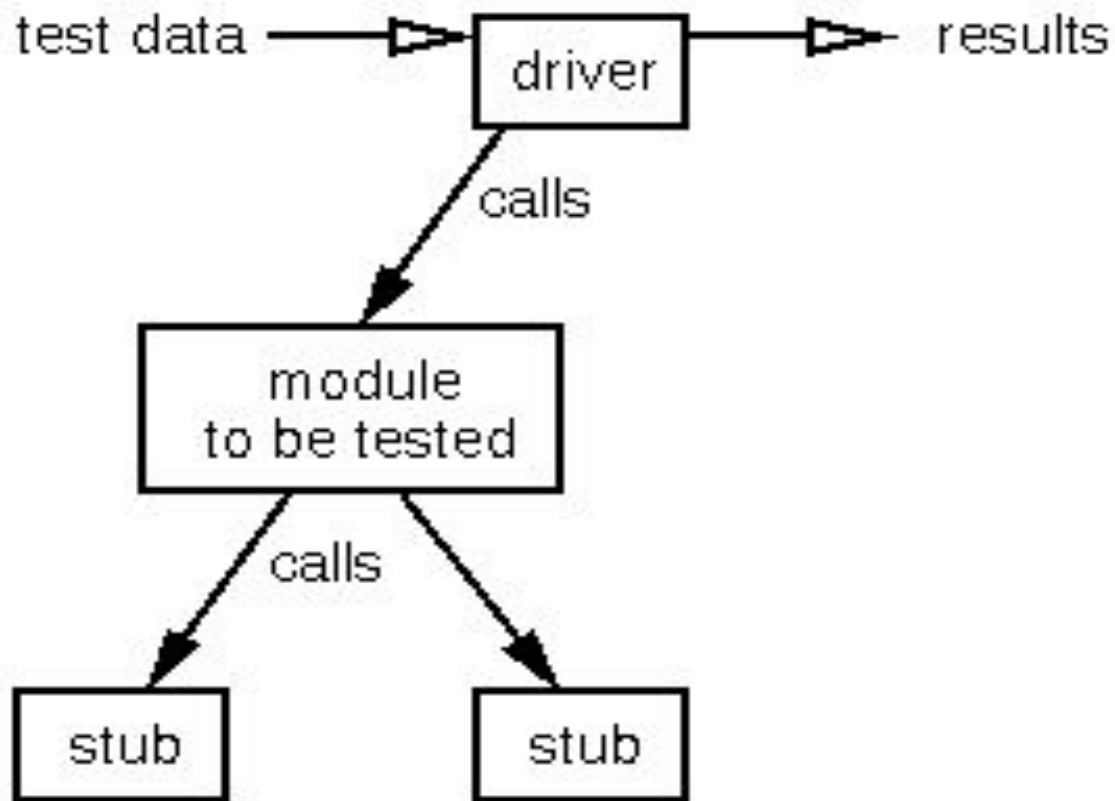
Top-down Integration Testing Strategy



Advantages and Disadvantages of Top-Down Integration Testing

- Test cases can be defined in terms of the functionality of the system (functional requirements). Structural techniques can also be used for the units in the top levels
- Writing stubs can be difficult especially when parameter passing is complex. Stubs must allow all possible conditions to be tested
- Possibly a very large number of stubs may be required, especially if the lowest level of the system contains many functional units
- One solution to avoid too many stubs: *Modified top-down testing strategy (Bruege)*
 - Test each layer of the system decomposition individually before merging the layers
 - Disadvantage of modified top-down testing: Both, stubs and drivers are needed

Deviation: Stubs and Drivers

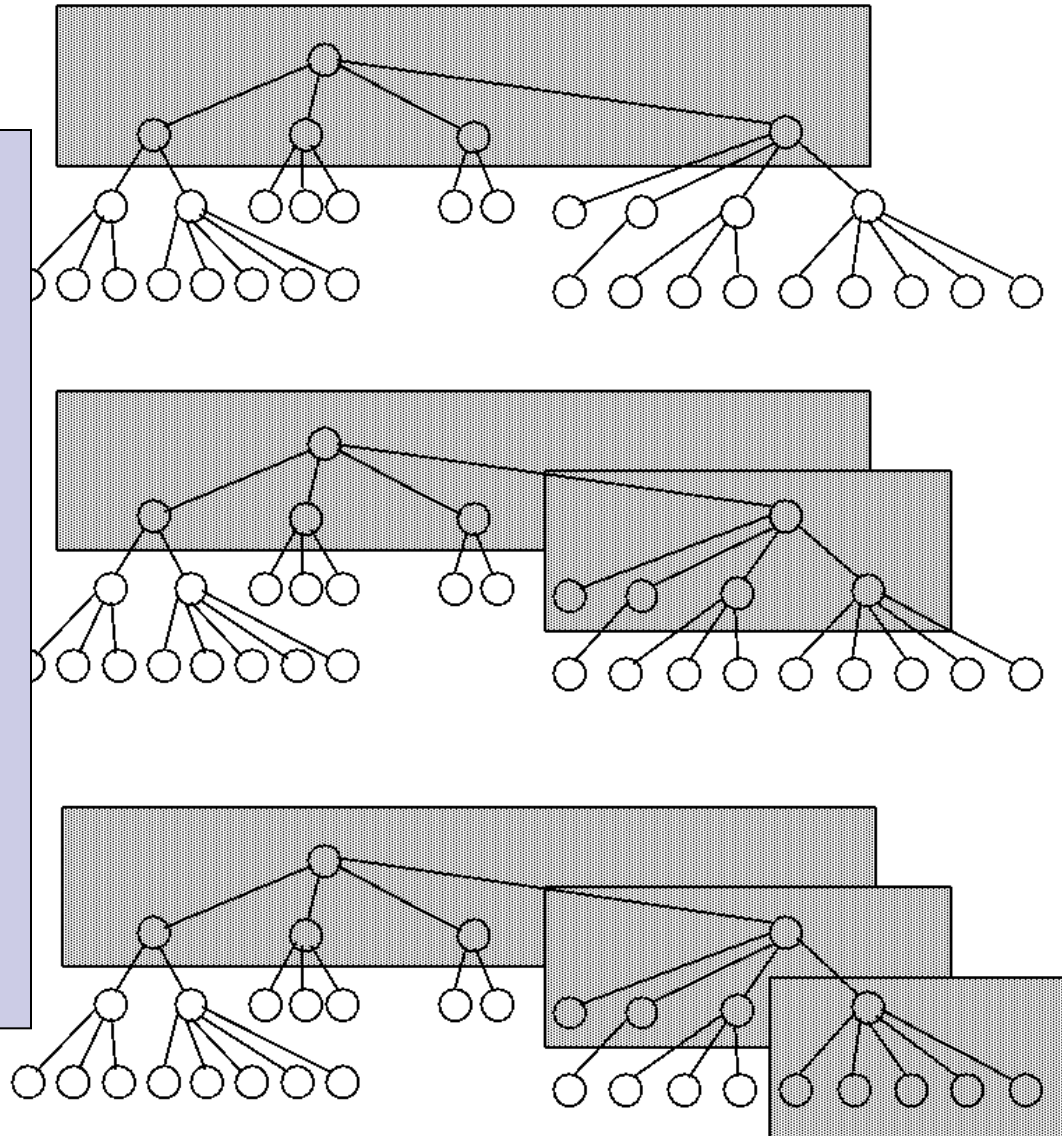


Top-Down Integration

Top Subtree (Sessions 1-4)

```
Procedure GetPINforPAN  
(PAN, ExpectedPIN) STUB  
IF PAN = '1123' THEN PIN :=  
'8876';  
IF PAN = '1234' THEN PIN :=  
'8765';  
IF PAN = '8746' THEN PIN :=  
'1253';  
End,
```

```
Procedure KeySensor  
(KeyHit) STUB  
data: KeyStrokes STACK OF '  
8', '8', '7', 'cancel '  
KeyHit = POP (KeyStrokes)  
End,
```



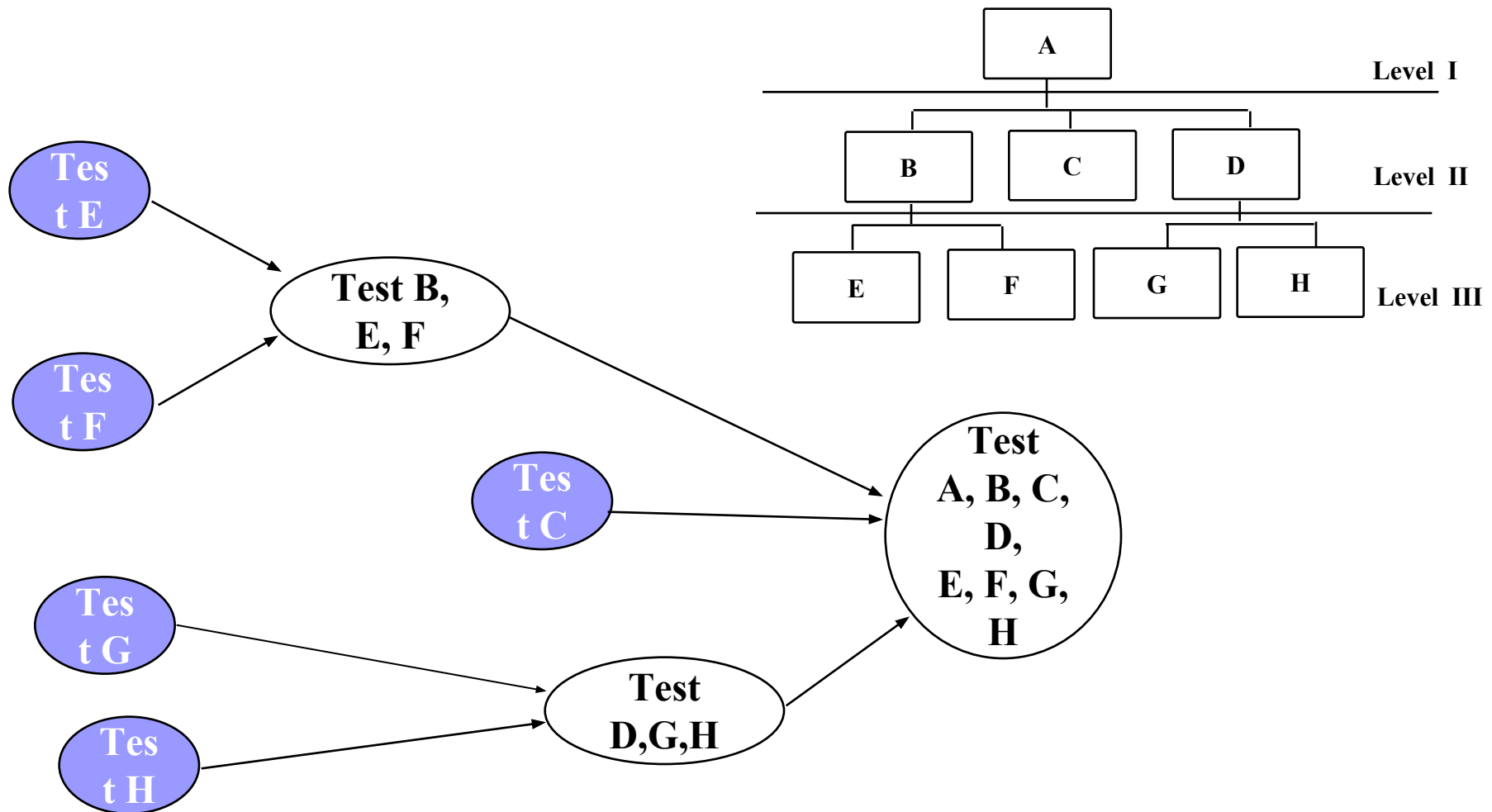
Top-down: Complications

- The most common complication occurs when processing at low level hierarchy demands adequate testing of upper level
- To overcome:
 - Either, delay many tests until stubs are replaced with actual modules (BAD)
 - Or, develop stubs that perform limited functions that simulate the actual module (GOOD)
 - Or, Integrate the software using bottom up approach

Bottom-Up Integration Testing Strategy

- Bottom-Up integration strategy focuses on testing the units at the lowest levels first (i.e. the units at the leafs of the decomposition tree)
- Integrate individual components in levels until the complete system is created
- The general process in bottom-up integration strategy is to gradually include the subsystems that reference/require the previously tested subsystems
- This is done repeatedly until all subsystems are included in the testing
- Special program called *Test Driver* is needed to do the testing,
 - The Test Driver is a “fake” routine that requires a subsystem and passes a test case to it

Example Bottom-Up Strategy





Advantages and Disadvantages of Bottom-Up Integration Testing

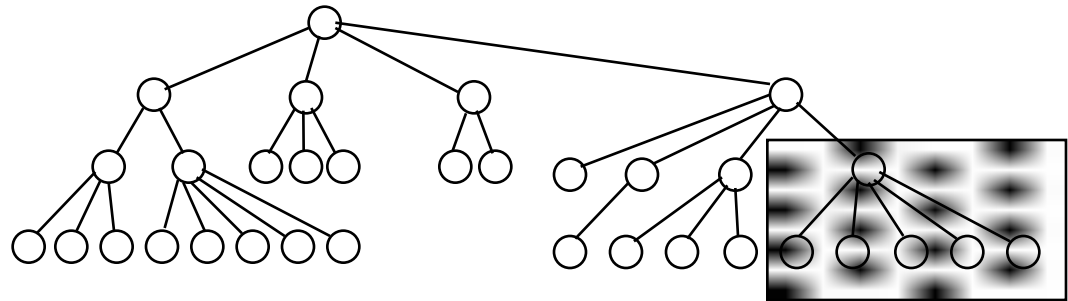
- Not optimal strategy for functionally decomposed systems:
 - Tests the most important subsystem (UI) last
- Useful for integrating the following systems
 - Object-oriented systems
 - Real-time systems
 - Systems with strict performance requirements

Bottom-up approach

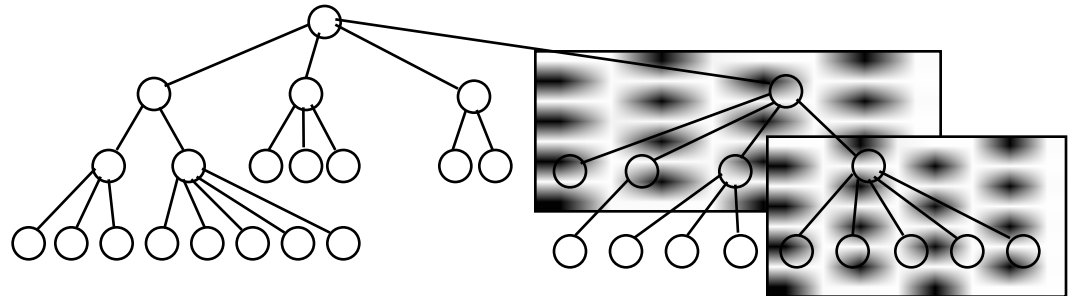
- Starts with construction and testing with atomic modules
 - No need for stub
 - Low level components are combined into cluster (or **builds**) to perform a specific sub-function
 - A driver (a control program for testing) is written to coordinate test cases input/output
 - Cluster is tested
 - Drivers are removed and clusters are combined moving upward in the program structure

Bottom-Up Integration

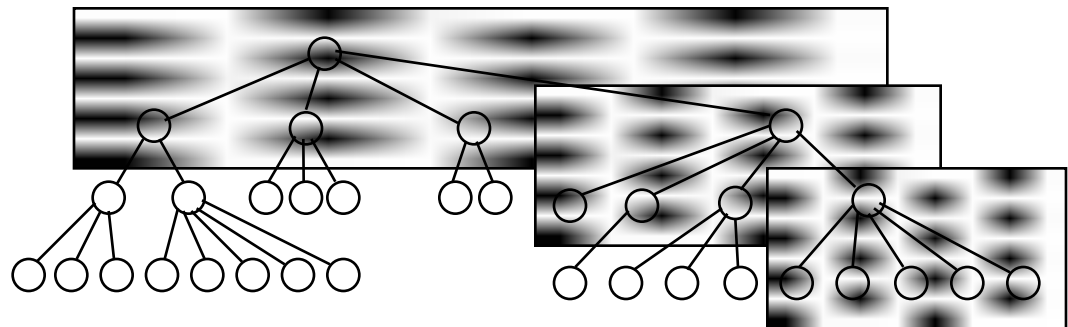
Bottom Level Subtree
(Sessions 13-17)



Second Level Subtree
(Sessions 25-28)



Top Subtree
(Sessions 29-32)



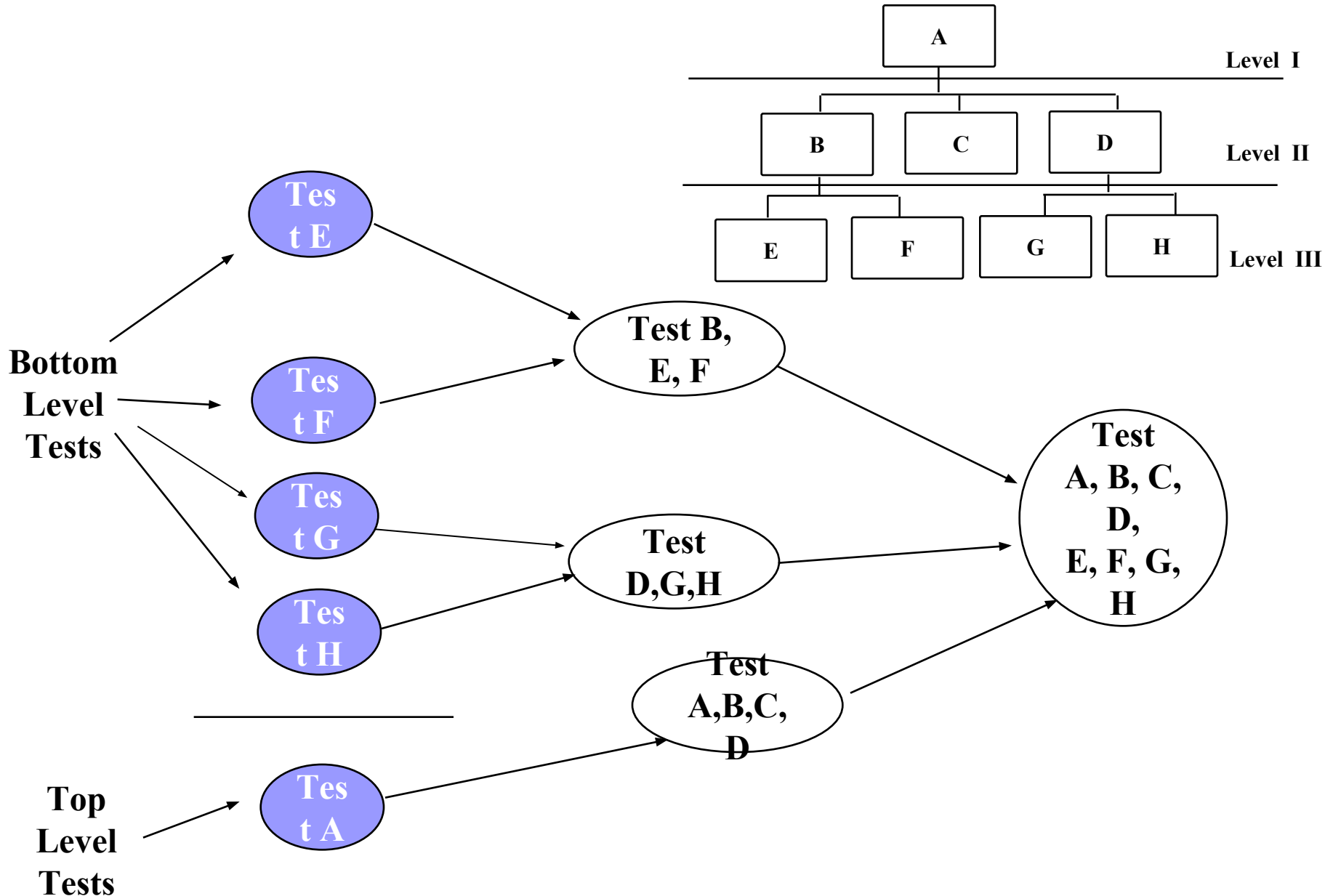
Top-down Vs. Bottom-up

- Architectural validation
 - Top-down integration testing is better at discovering errors in the system architecture
- System demonstration
 - Top-down integration testing allows a limited demonstration at an early stage in the development
- Test implementation
 - Often easier with bottom-up integration testing

Sandwich Testing Strategy

- Combines top-down strategy with bottom-up strategy
- *The system is viewed as having three layers*
 - A target layer in the middle
 - A layer above the target
 - A layer below the target
 - Testing converges at the target layer
- How do you select the target layer if there are more than 3 layers?
 - Heuristic: Try to minimize the number of stubs and drivers

Sandwich Testing Strategy

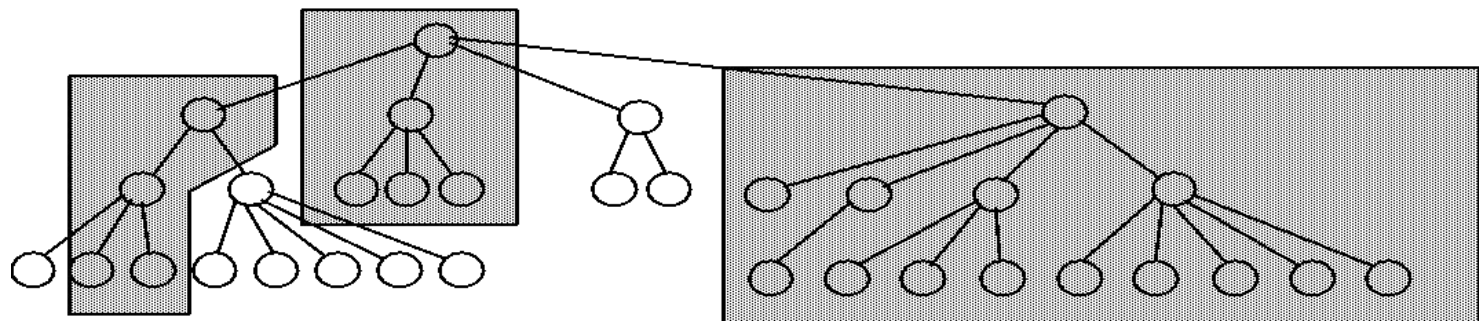
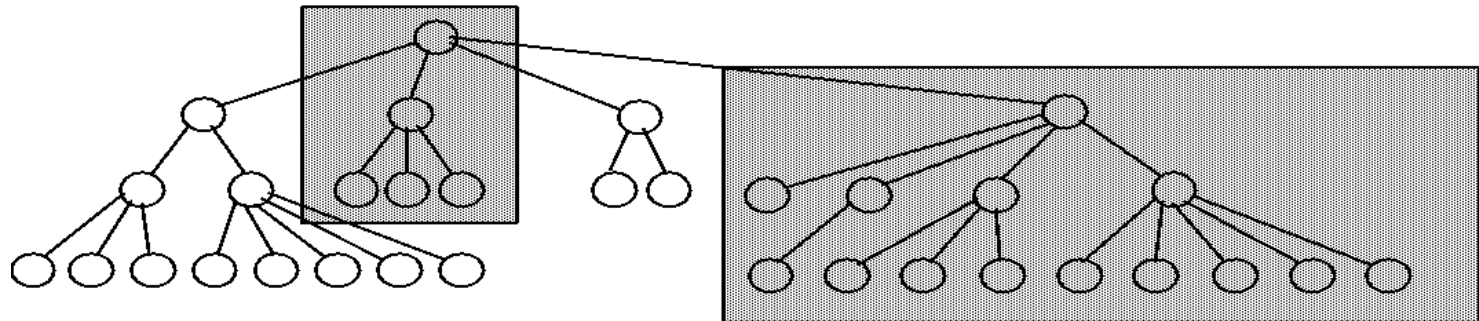
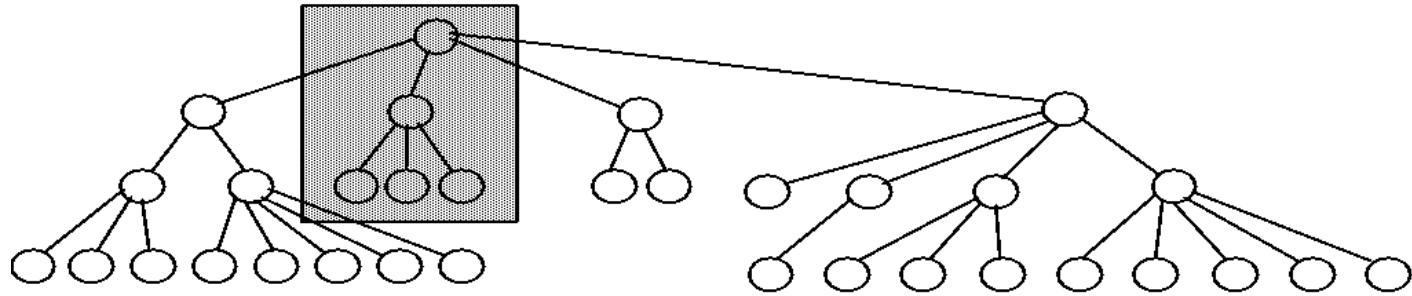




Advantages and Disadvantages of Sandwich Integration Testing

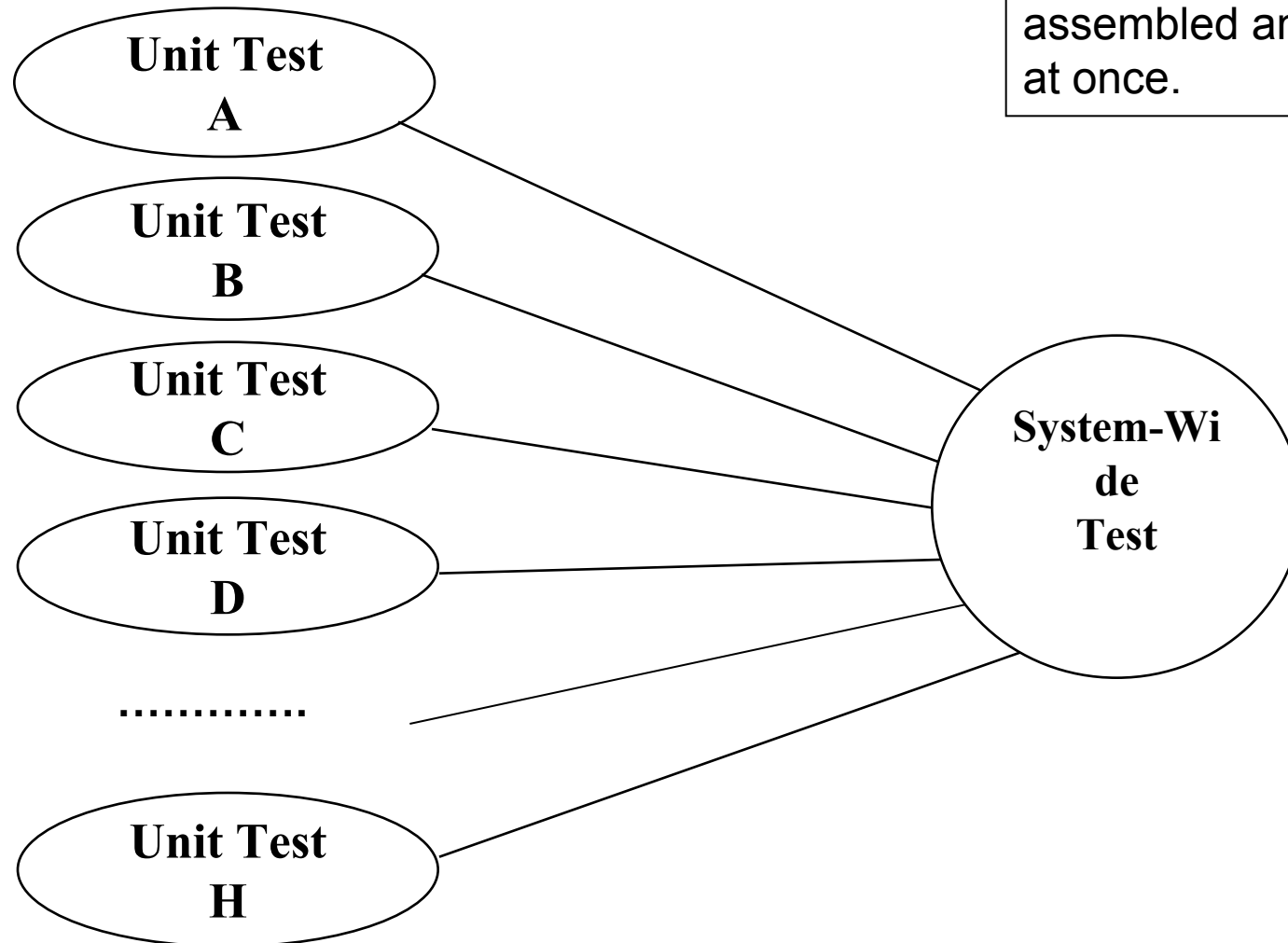
- Top and Bottom Layer Tests can be done in parallel
- Does not test the individual subsystems thoroughly before integration
- Solution: Modified sandwich testing strategy (Bruege)

Sandwich Integration



Big-Bang Integration Testing

All components are assembled and tested at once.



Test Sessions

- A test session refers to **one set of tests** for a specific configuration of actual code and stubs
- The number of integration test sessions using a decomposition tree can be computed
 - $\text{Sessions} = \text{nodes} - \text{leaves} + \text{edges}$

Decomposition based testing: 2

- For SATM system
 - 42 integration testing session (i.e., 42 separate sets of integration test cases)
 - top/down
 - (Nodes-1) stubs are needed
 - 32 stub in SATM
 - bottom/up
 - (Nodes-leaves) of drivers are needed
 - 10 drivers in SATM

Decomposition based strategies: Pros and Cons

- Intuitively clear and understandable
 - In case of faults, most recently added units are suspected ones
- Can be tracked against decomposition tree
- Suggests breadth-first or depth-first traversals
- Units are merged using the decomposition tree
 - Correct behavior follows from individually correct units and interfaces
- Stubs/Drives are major development Overhead



Call Graph Based Integration

- The basic idea is to use the call graph instead of the decomposition tree
- The call graph is a directed, labeled graph
- Two types of call graph based integration testing
 - Pair-wise Integration Testing
 - Neighborhood Integration Testing



Pair-Wise Integration Testing

- The idea behind Pair-Wise integration testing is to eliminate the need for developing stubs/drivers
- The objective is to use actual code instead of stubs/drivers
- In order not to deteriorate the process to a big-bang strategy, we restrict a testing session to just a pair of units in the call graph
- The result is that we have one integration test session for each edge in the call graph

Neighborhood Integration Testing

- We define the neighborhood of a node in a graph to be the set of nodes that are one edge away from the given node
- In a directed graph means all the immediate predecessor nodes and all the immediate successor nodes of a given node
- The number of neighborhoods for a given graph can be computed as:

$$\text{InteriorNodes} = \text{nodes} - (\text{SourceNodes} + \text{SinkNodes})$$

$$\text{Neighborhoods} = \text{InteriorNodes} + \text{SourceNodes}$$

Or

$$\text{Neighborhoods} = \text{nodes} - \text{SinkNodes}$$

- Neighborhood Integration Testing reduces the number of test sessions



Advantages and Disadvantages of Call-Graph Integration Testing

- Call graph based integration techniques move towards a behavioral basis
- Aim to eliminate / reduce the need for drivers/stubs
- Closer to a build sequence
- Neighborhoods can be combined to create “villages”
- Suffer from the fault isolation problem especially for large neighborhoods
- Nodes can appear in several neighborhoods

Call graph based integration testing

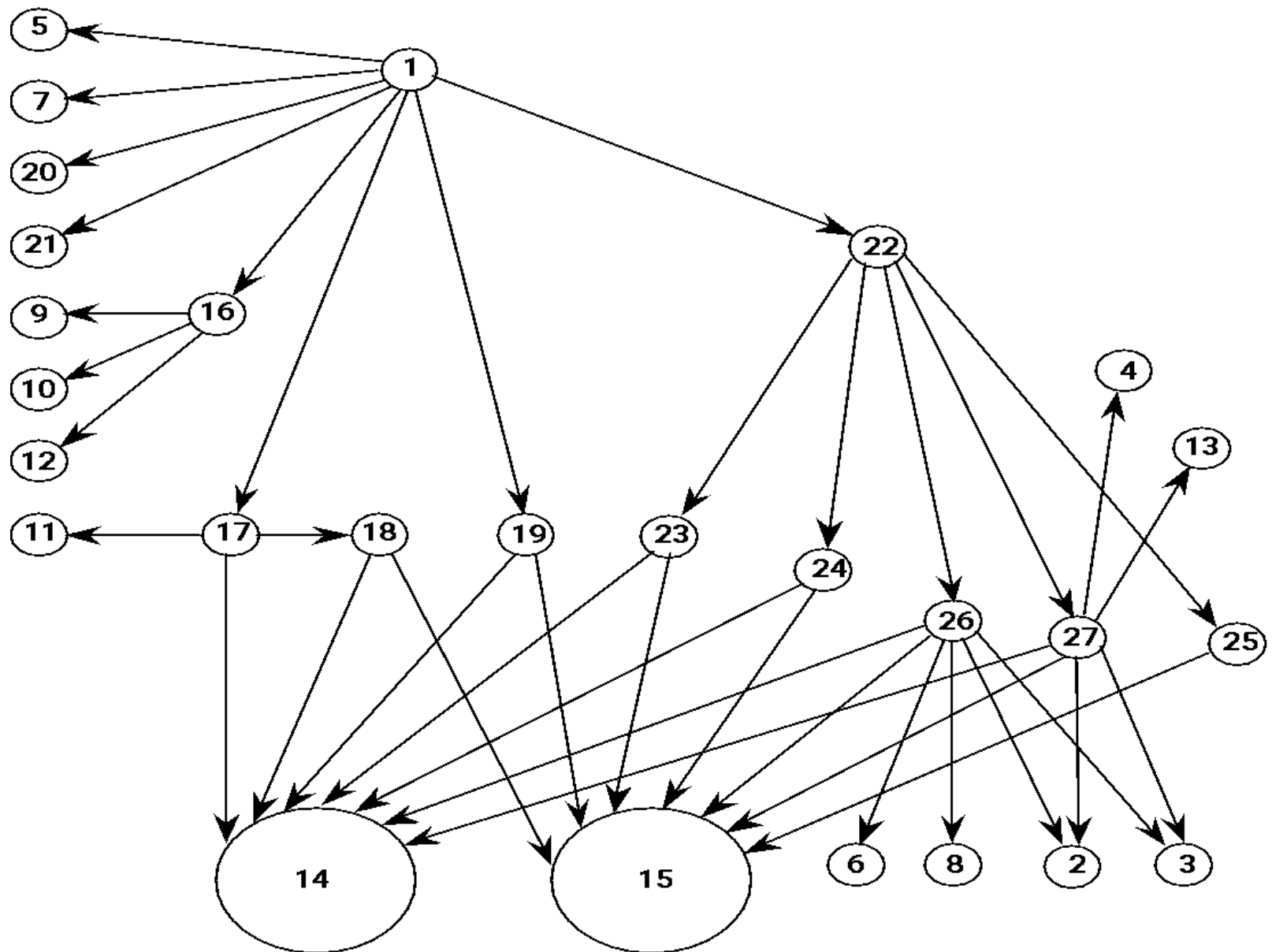
- Call graph
 - A directed graph
 - Nodes corresponds to unit
 - Edges corresponds to the call
 - E.g.
 - $A \rightarrow B$ (i.e., A is calling B)
- Attempts to overcome the decomposition problem (structural)
- Moves toward behavioral testing



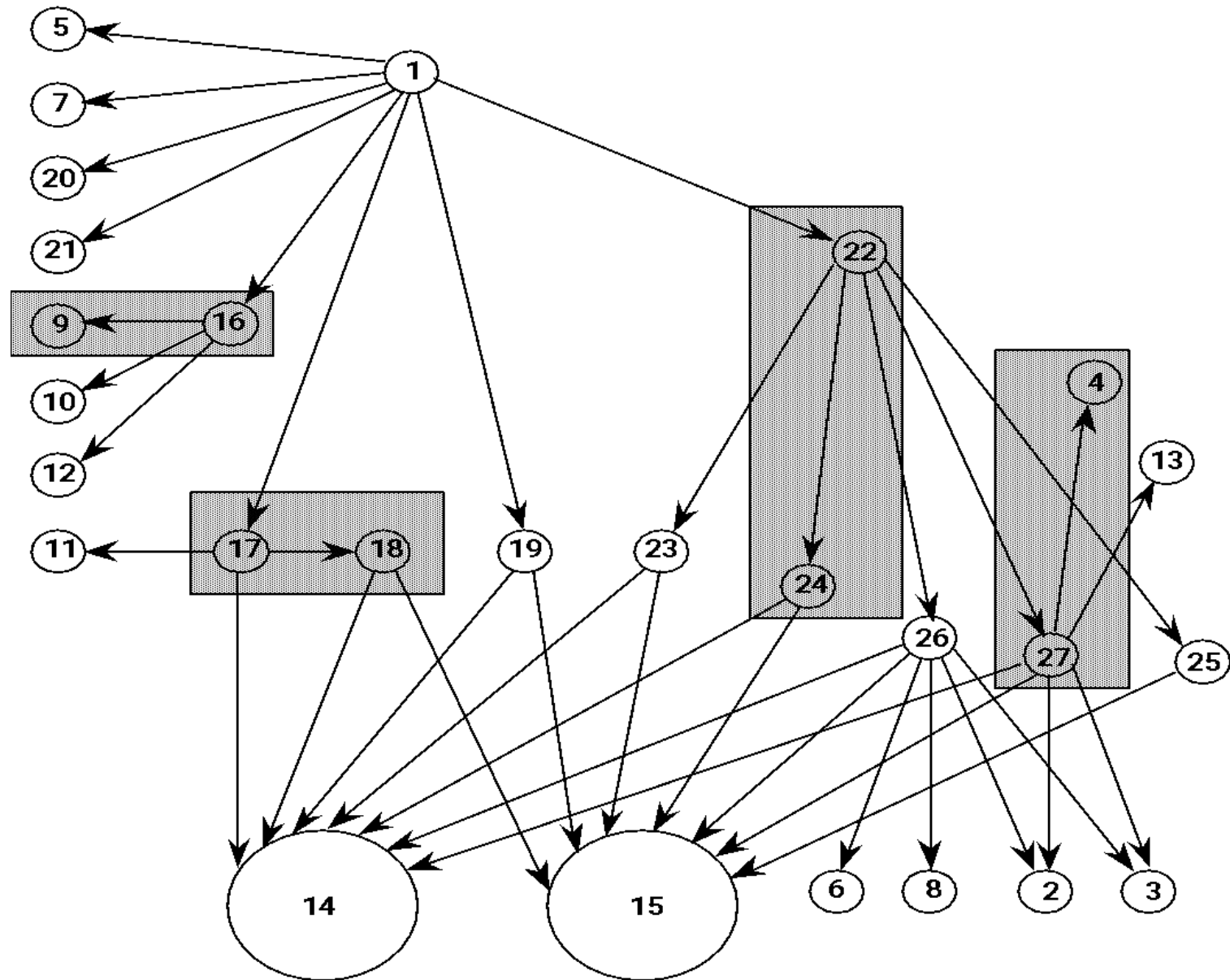
Call graph (CG): approaches

- Two main approaches based on Call Graph
 - Pair-wise integration
 - Neighborhood integration

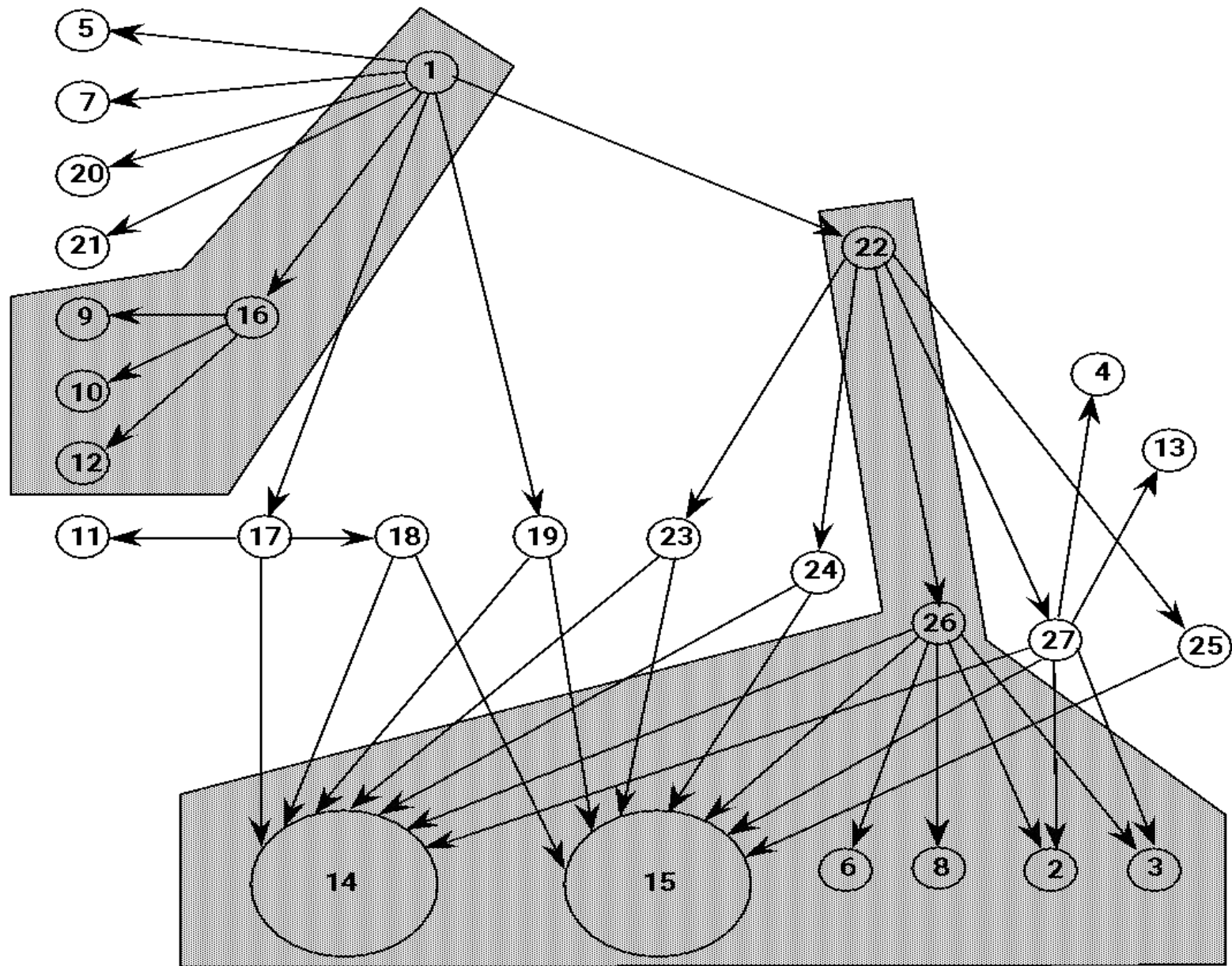
Call Graph of the SATM System



Some Pair-wise Integration Sessions



Two Neighborhood Integration Sessions



SATM Neighborhoods

Node	Predecessors	Successors
16	1	9, 10, 12
17	1	11, 14, 18
18	17	14, 15
19	1	14, 15
23	22	14, 15
24	22	14, 15
26	22	14, 15, 6, 8, 2, 3
27	22	14, 15, 2, 3, 4, 13
25	22	15
22	1	23, 24, 26, 27, 25
1	n/a	5, 7, 2, 21, 16, 17, 19, 22

Pros and cons

- Benefits (GOOD)
 - Mostly behavioral than structural
 - Eliminates sub/drive overhead
 - Works well with incremental development method such as Build and composition
- Liabilities (BAD)
 - The fault isolation
 - E.g.,
 - Fault in one node appearing in several neighborhood

Path-based Integration Testing

- The hybrid approach (i.e., structural and behavioral) is an ideal one integration testing
- The focus is on the interactions among the units
 - Interfaces are structural
 - Interactions are behavioral
- With unit testing, some path of source statements is traversed
 - What happens when there is a call to another unit?
 - Ignore the single-entry/single exit
 - Use exist follows by an entry
 - Suppress the call statement

The path based integration testing (definition)

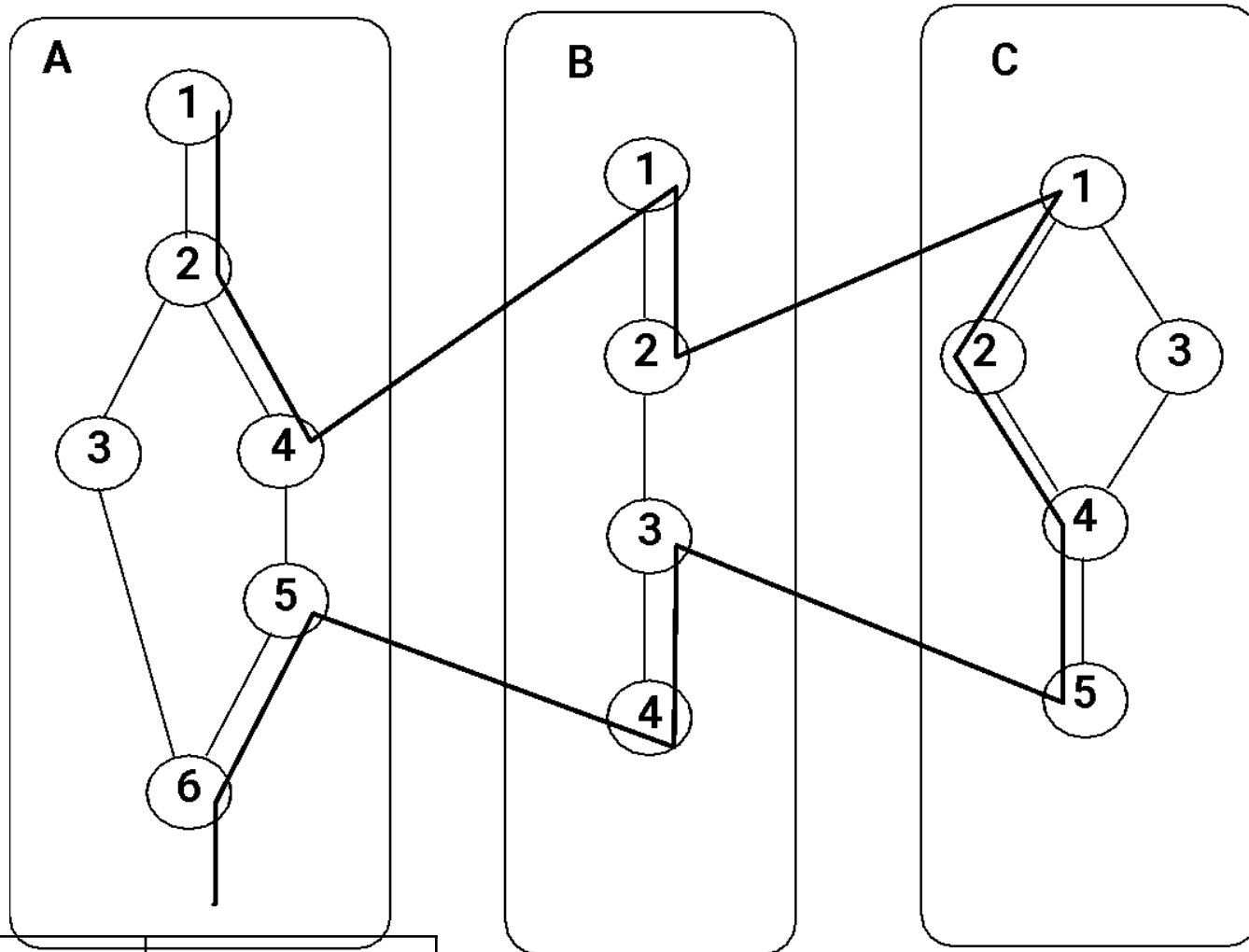
- MM-Path (definition)
 - An interleaved sequence of module execution paths (MEP) and messages
- MM-Path can be used
 - to describe sequences of module execution paths including transfers of control among units using messages
 - To represents feasible execution paths that cross unit boundaries
 - To extent program graph
 - Where
 - nodes = execution paths
 - edges = messages
- Atomic system function (ASF)
 - An action that is observable at the system level in terms of port input and output events



MM-Path Graph (definition)

- Given a set of units, their MM-Path graph is the directed graph in which nodes are module execution paths (MM-PATHS) and edges represents messages/returns from one unit to another
- Supports composition of units

An MM-Path



Module	Source node	Sink node
A	1,5	4,6
B	1,3	2,4
C	1	4

Figure 13.4

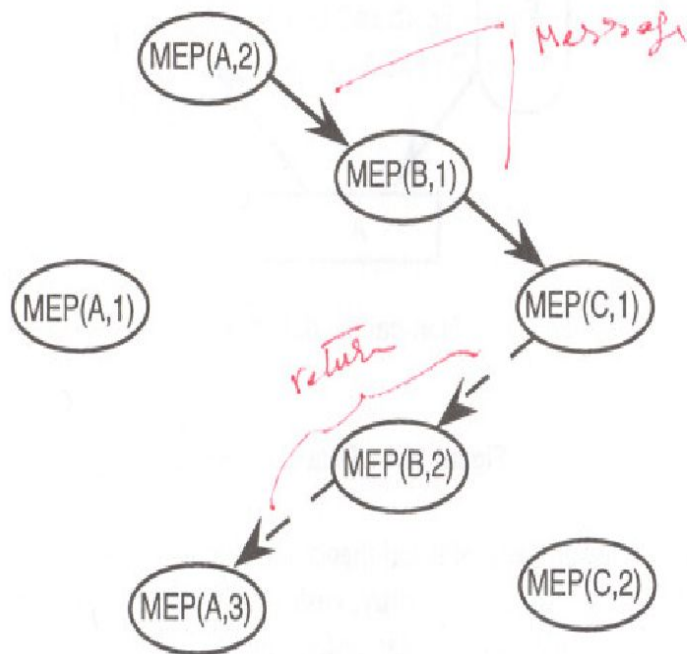


Figure 13.4 MM-Path Graph Derived from Figure 13.3

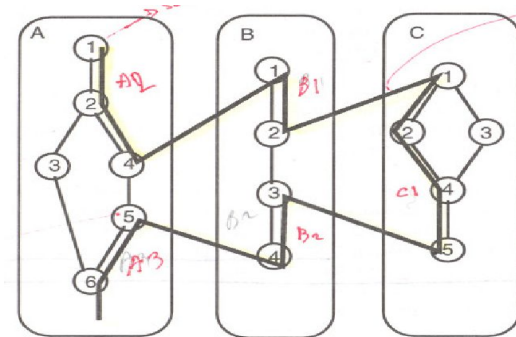


Figure 13.3 MM-Path Across Three Units

$MEP(A,1) = \langle 1, 2, 3, 6 \rangle$
 $MEP(A,2) = \langle 1, 2, 4 \rangle$
 $MEP(A,3) = \langle 5, 6 \rangle$
 $MEP(B,1) = \langle 1, 2 \rangle$
 $MEP(B,2) = \langle 3, 4 \rangle$
 $MEP(C,1) = \langle 1, 2, 4, 5 \rangle$
 $MEP(C,2) = \langle 1, 3, 4, 5 \rangle$

ation testing codes of the DD Path graph d

PDL Description of SATM Main Program

```
1. Main Program
2. State = AwaitCard
3. CASE State OF
4. AwaitCard:  ScreenDriver(1, null)
5.             WatchCardSlot(CardSlotStatus)
6.             WHILE CardSlotStatus is Idle DO
7.                 WatchCardSlot(CardSlotStatus)
8.                 ControlCardRoller(accept)
9.                 ValidateCard(CardOK, PAN)
10.                IF CardOK THEN State = AwaitPIN
11.                    ELSE ControlCardRoller(eject)
12.                        State = AwaitCard
13. AwaitPIN:   ValidatePIN(PINok, PAN)
14.             IF PINok THEN ScreenDriver(5, null)
15.                 State = AwaitTrans
16.             ELSE ScreenDriver(4, null)
17.                 State = AwaitCard
18. AwaitTrans: ManageTransaction
19.             State = CloseSession
20. CloseSession: IF NewTransactionRequest
21.                THEN State = AwaitTrans
22.                ELSE PrintReceipt
23.                PostTransactionLocal
24.                CloseSession
25.                ControlCardRoller(eject)
26.                State = AwaitCard
27. End, (CASE State)
28. END. (Main program SATM)
```

PDL Description of ValidatePIN Procedure

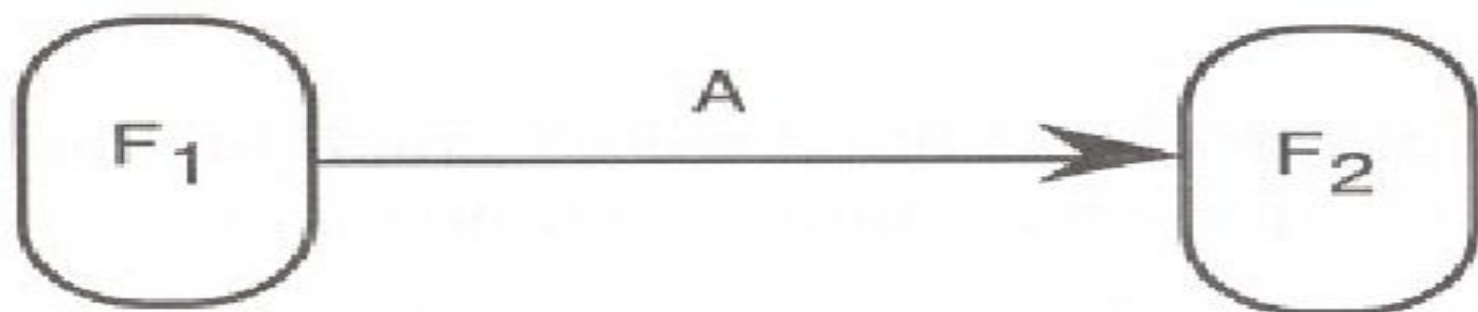
```
29. Procedure ValidatePIN(PINok, PAN)
30. GetPINforPAN(PAN, ExpectedPIN)
31. Try = First
32. CASE Try OF
33. First: ScreenDriver(2, null)
34.     GetPIN(EnteredPIN)
35.     IF EnteredPIN = ExpectedPIN
36.         THEN PINok = TRUE
37.         RETURN
38.     ELSE ScreenDriver(3, null)
39.         Try = Second
40. Second: ScreenDriver(2, null)
41.     GetPIN(EnteredPIN)
42.     IF EnteredPIN = ExpectedPIN
43.         THEN PINok = TRUE
44.         RETURN
45.     ELSE ScreenDriver(3, null)
46.         Try = Third
47. Third: ScreenDriver(2, null)
48.     GetPIN(EnteredPIN)
49.     IF EnteredPIN = ExpectedPIN
50.         THEN PINok = TRUE
51.         RETURN
52.     ELSE ScreenDriver(4, null)
53.         PINok = FALSE
54. END, (CASE Try)
55. END. (Procedure ValidatePIN)
```

PDL Description of GetPIN Procedure

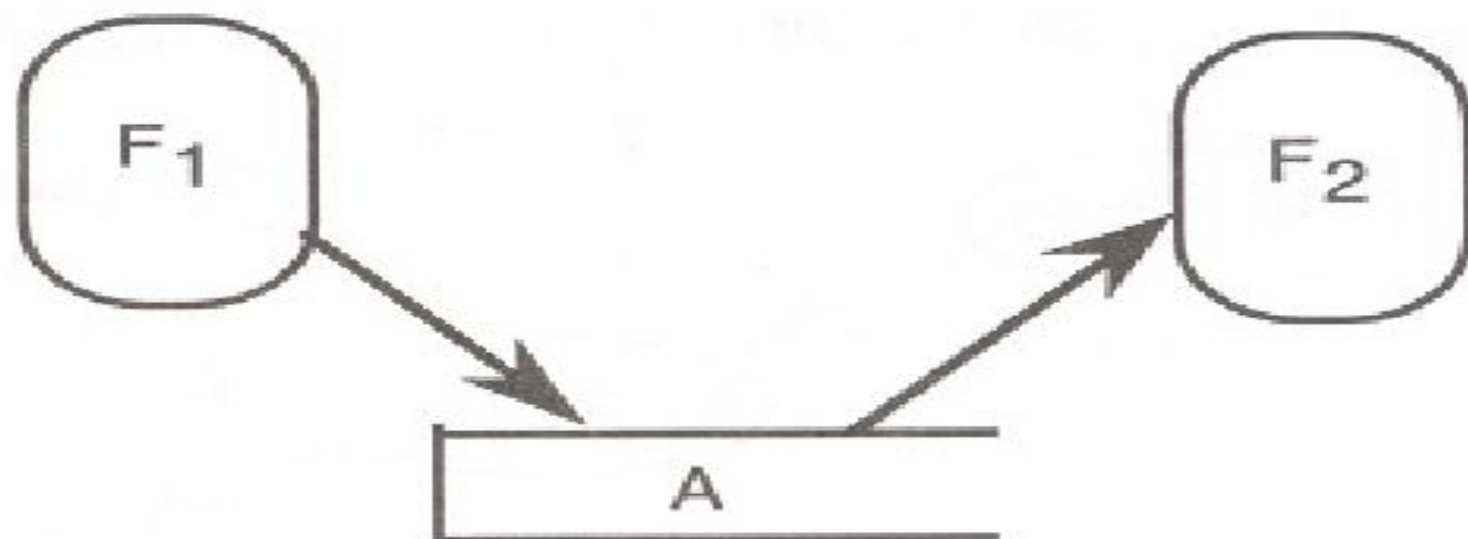
```
56. Procedure GetPIN(EnteredPIN, CancelHit)
57. Local Data: DigitKeys = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
58. BEGIN
59.   CancelHit = FALSE
60.   EnteredPIN = null string
61.   DigitsRcvd=0
62.   WHILE NOT(DigitsRcvd=4 OR CancelHit) DO
63.     BEGIN
64.       KeySensor(KeyHit)
65.       IF KeyHit IN DigitKeys
66.         THEN BEGIN
67.           EnteredPIN = EnteredPIN + KeyHit
68.           INCREMENT(DigitsRcvd)
69.           IF DigitsRcvd=1 THEN ScreenDriver(2,'X---')
70.           IF DigitsRcvd=2 THEN ScreenDriver(2,'XX--')
71.           IF DigitsRcvd=3 THEN ScreenDriver(2,'XXX-')
72.           IF DigitsRcvd=4 THEN ScreenDriver(2,'XXXX')
73.         END
74.     END {WHILE}
75. END. (Procedure GetPIN)
```

More on MM-Path

- MM-Path issues
 - How long is an MM-Path?
 - What is the endpoint?
- The following observable behavior that can be used as endpoints
 - Event quiescence (event inactivity)
 - System level event
 - Happens when system is idle/waiting
 - Message quiescence (msg inactivity)
 - Unit that sends no messages is reached
 - Data quiescence (data inactivity)
 - Happens when a sequences of processing generate a stored data that is not immediately used
 - E.g. account balance that is not used immediately



Causal data flow



Non-causal data flow

Figure 13.5 Data Quiescence



MM-Path Guidelines

- MM-Path Guiltiness
 - Points of quiescence are natural endpoints for an MM-path
 - atomic system functions (system behavior) are considered as an upper limit for MM-Paths
 - MM-Paths should not cross ASF boundaries



Pros and cons

- (GOOD) hybrid approach
- (GOOD) The approach works equally well for software testing developed by waterfall model
- (GOOD) Testing closely coupled with actual system behavior
- (BAD) Identification of MM-Paths which can be offset by elimination of sub/drivers

Number of Integration Testing Sessions

1. Based on Functional Decomposition

Top-Down	nodes-leaves+edges
Bottom-Up	nodes-leaves+edge
Sandwich	(Max is number of subtrees)
Big Bang	1

2. Based on Call Graph

Pair-wise	number of edges
Neighborhood	nodes - sink nodes

3. Based on Paths

MM-Paths	(application dependent)
Atomic System Functions	(application dependent)

Effort and Throw-away Code

Method	Sessions	Stubs/Drivers
Top-Down	42	32 stubs
Bottom-Up	42	10 drivers
Sandwich	?	none
Big Bang	1	none
Pair-wise	39	pair dependent
Neighborhood	11	neighborhood dependent
MM-Path	1 per MM-Path	1 driver per MM-Path
ASF	1 per ASF	none

