UNIT 4

Levels of Testing,
Integration Testing

contents

- **Levels of Testing, Integration Testing:** Traditional view of testing levels, Alternative life-cycle models, The SATM system, Separating integration and system testing. A closer look at the SATM system, Decomposition-based,call graph-based, Path-based integrations.

Goals/Purpose of Integration Testing

- Presumes previously tested units
- Not system testing
- Tests functionality "between" unit and system levels
- Basis for test case identification?

Testing Level Assumptions and Objectives

- Unit assumptions
  - All other units are correct
  - Compiles correctly
- Integration assumptions
  - Unit testing complete
- System assumptions
  - Integration testing complete
  - Tests occur at port boundary
- Unit goals
  - Correct unit function
  - Coverage metrics satisfied
- Integration goals
  - Interfaces correct
  - Correct function across units
  - Fault isolation support
- System goals
  - Correct system functions
  - Non-functional requirements tested
  - Customer satisfaction.

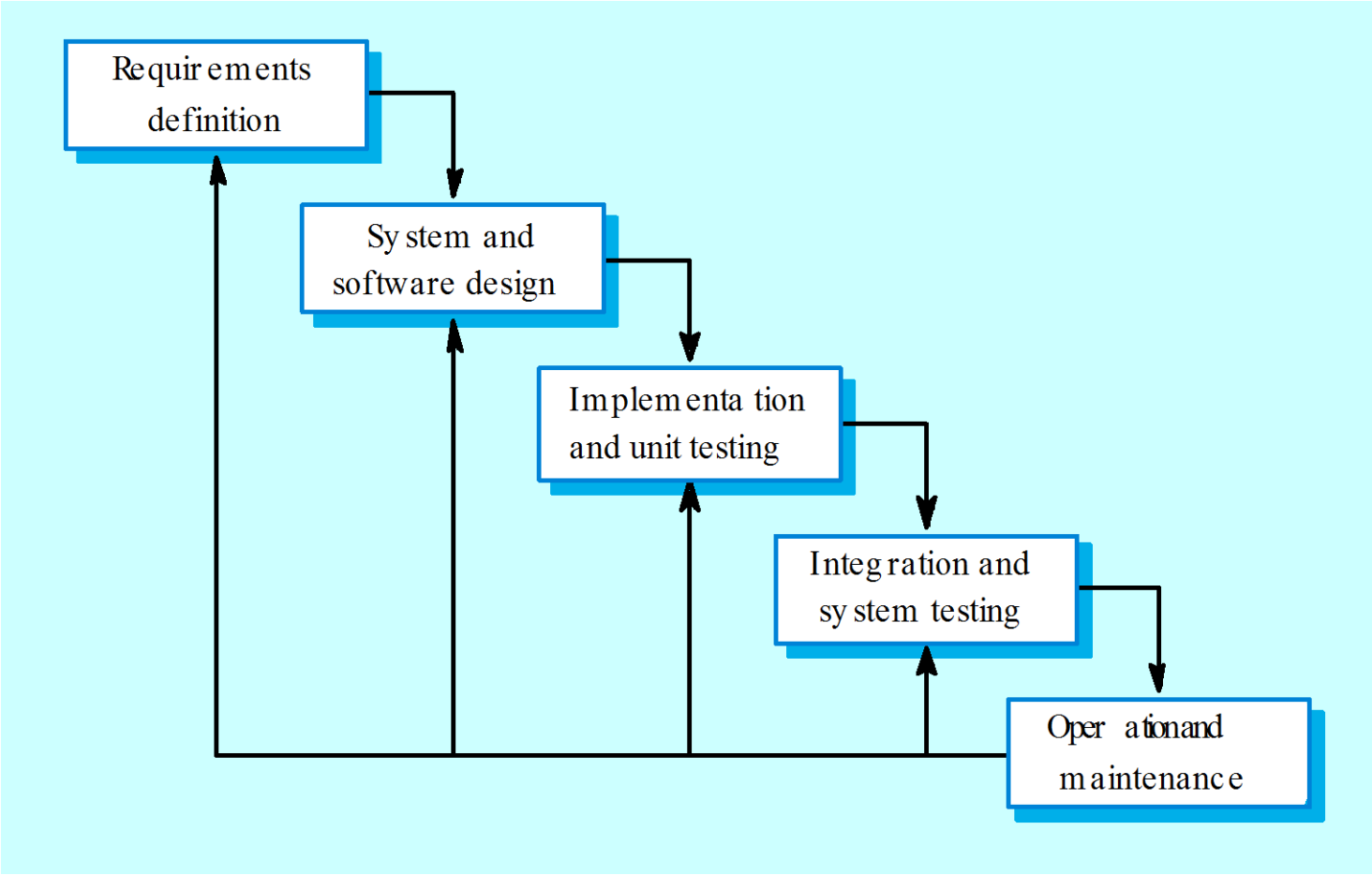
The software process

- A structured set of activities required to develop a software system
  - Specification;
  - Design;
  - Validation;
  - Evolution.
- A software process model is an abstract representation of a process. It presents a description of a process from some particular perspective.

Approaches to Integration Testing
("source" of test cases)

- Functional Decomposition (most commonly described in the literature)
  - Top-down
  - Bottom-up
  - Sandwich
  - "Big bang"
- Call graph
  - Pairwise integration
  - Neighborhood integration
- Paths
  - MM-Paths
  - Atomic System Functions

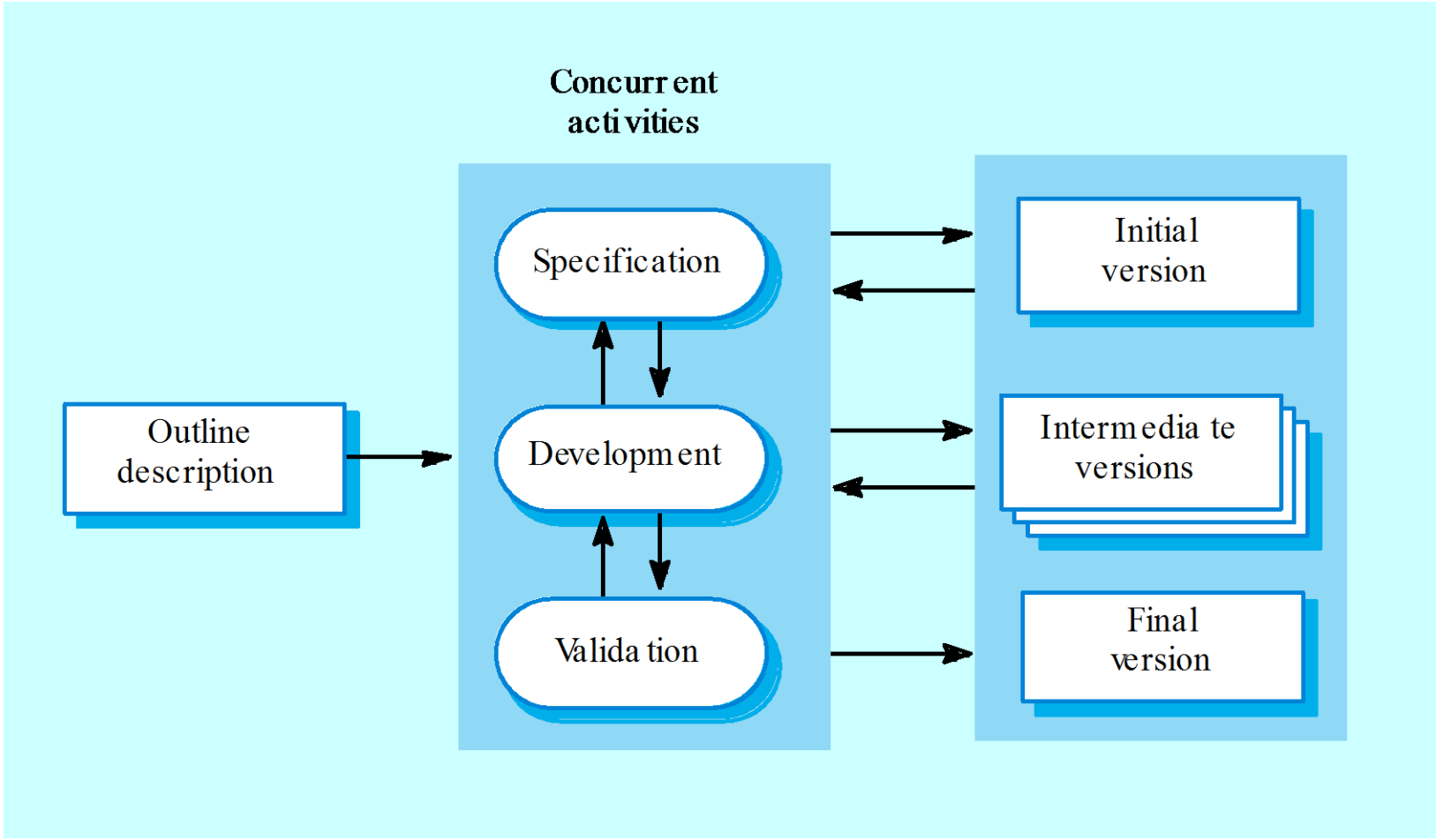
Waterfall model



Waterfall model phases

- Requirements analysis and definition
- System and software design
- Implementation and unit testing
- Integration and system testing
- Operation and maintenance
- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway. One phase has to be complete before moving onto the next phase.

Evolutionary development



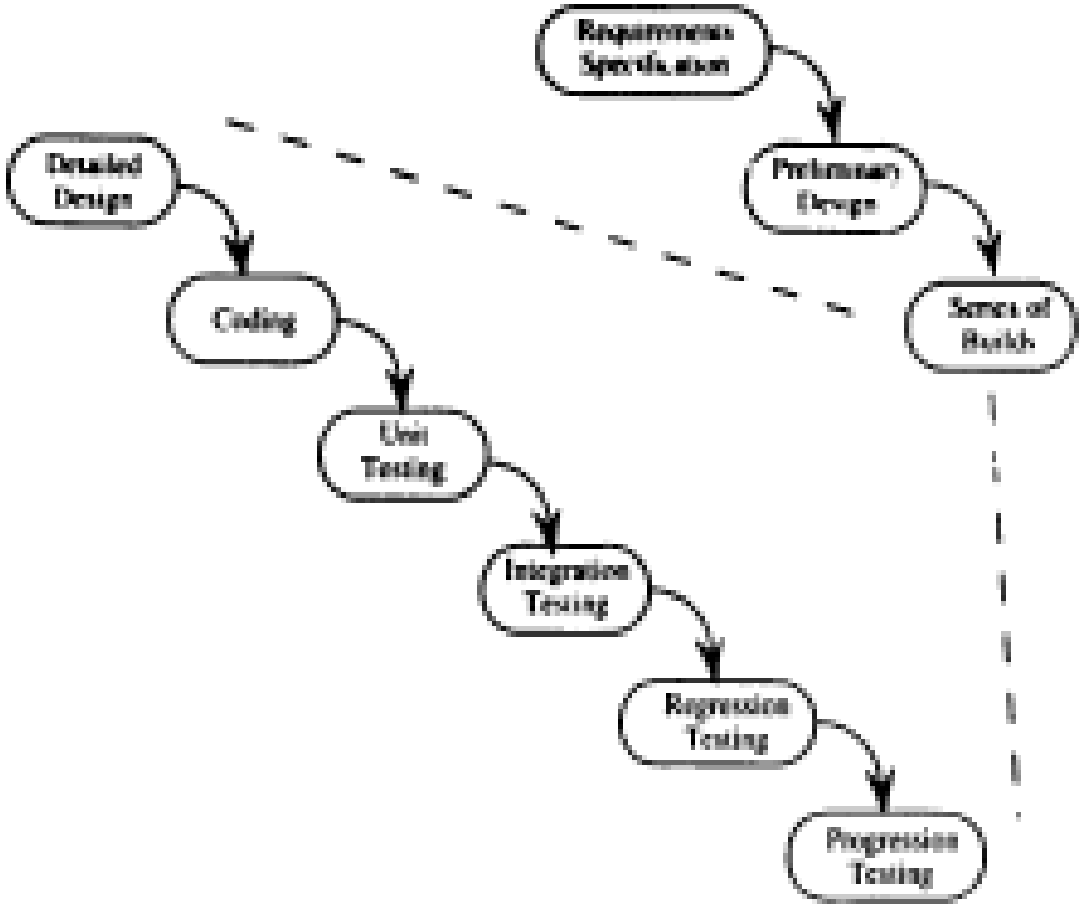
Evolutionary development

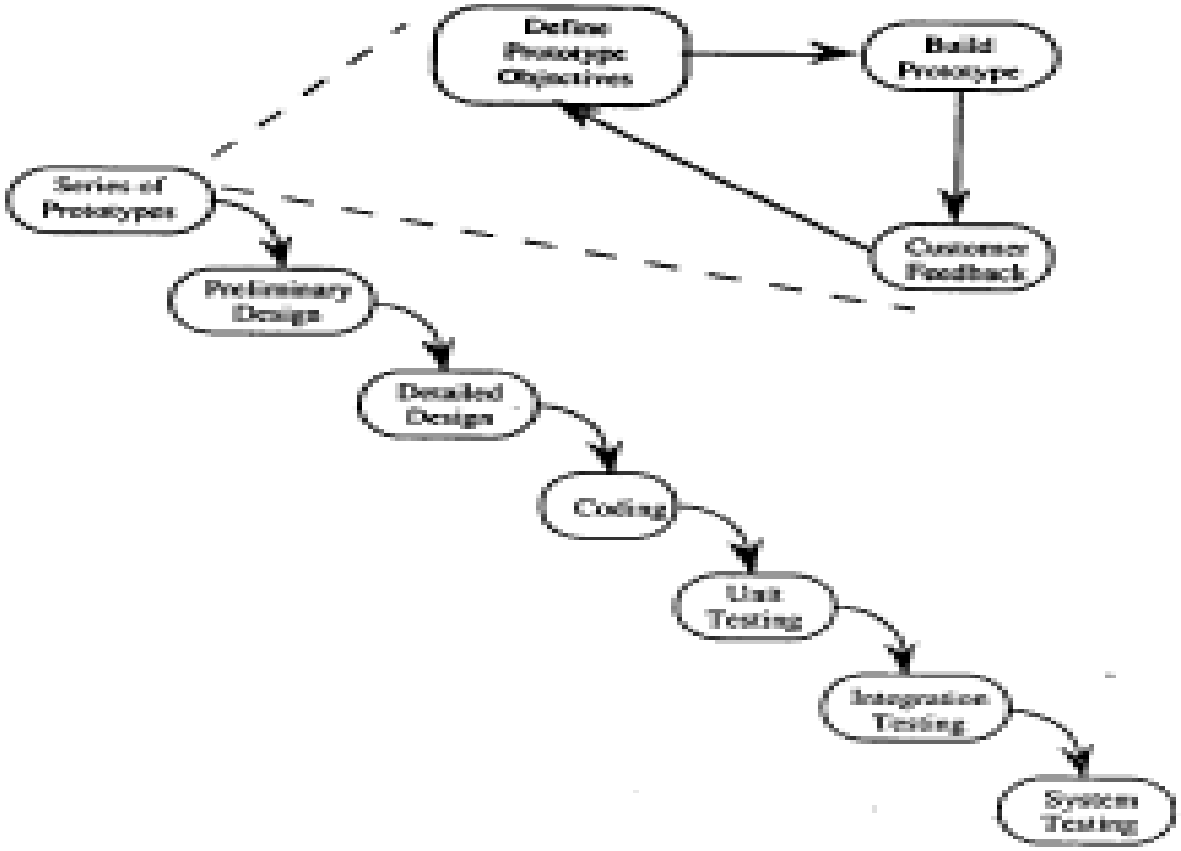
- Problems
  - Lack of process visibility;
  - Systems are often poorly structured;
  - Special skills (e.g. in languages for rapid prototyping) may be required.
- Applicability
  - For small or medium-size interactive systems;
  - For parts of large systems (e.g. the user interface);
  - For short-lifetime systems.

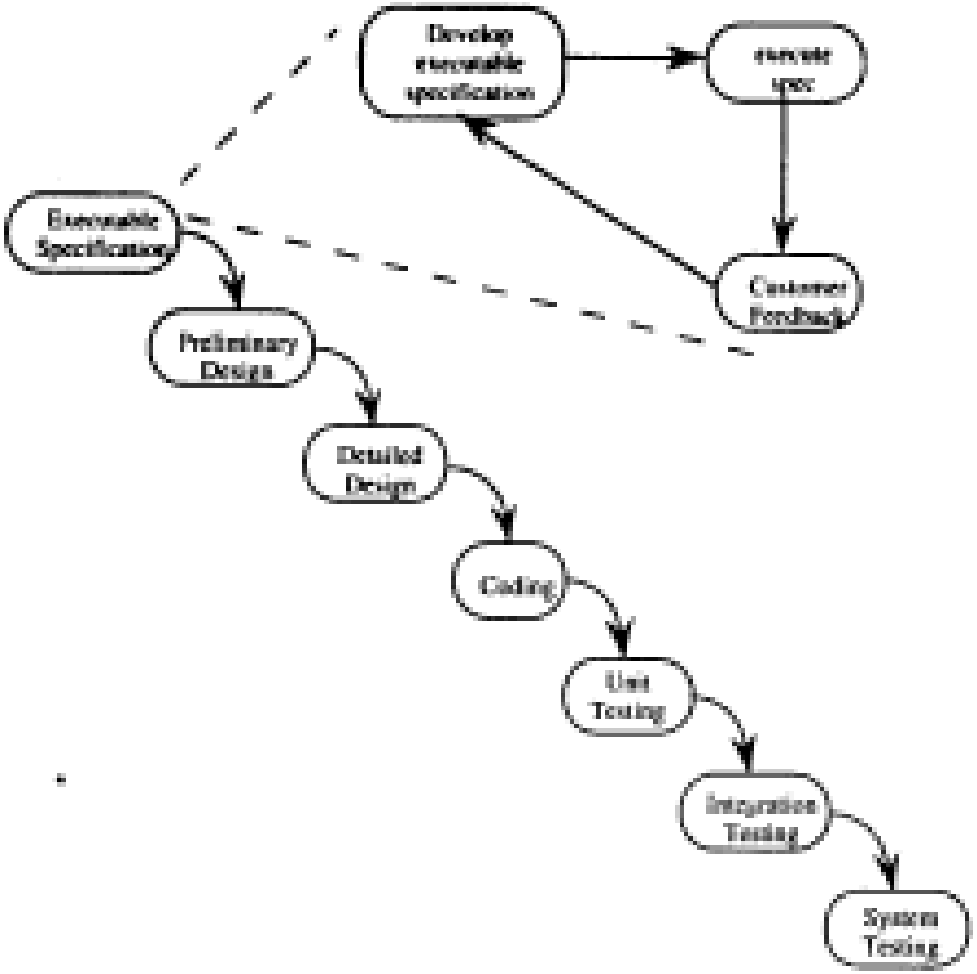
Spiral development

- Process is represented as a spiral rather than as a sequence of activities with backtracking.
- Each loop in the spiral represents a phase in the process.
- No fixed phases such as specification or design - loops in the spiral are chosen depending on what is required.
- Risks are explicitly assessed and resolved throughout the process.

Waterfall spin-offs




Define Prototype Objectives
Build Prototype
Series of Prototypes
Customer Feedback
Preliminary Design
Detailed Design
Coding
Unit Testing
Integration Testing
System Testing


Develop executable specification
execute spec
Executable Specification
Customer Feedback
Preliminary Design
Detailed Design
Coding
Unit Testing
Integration Testing
System Testing

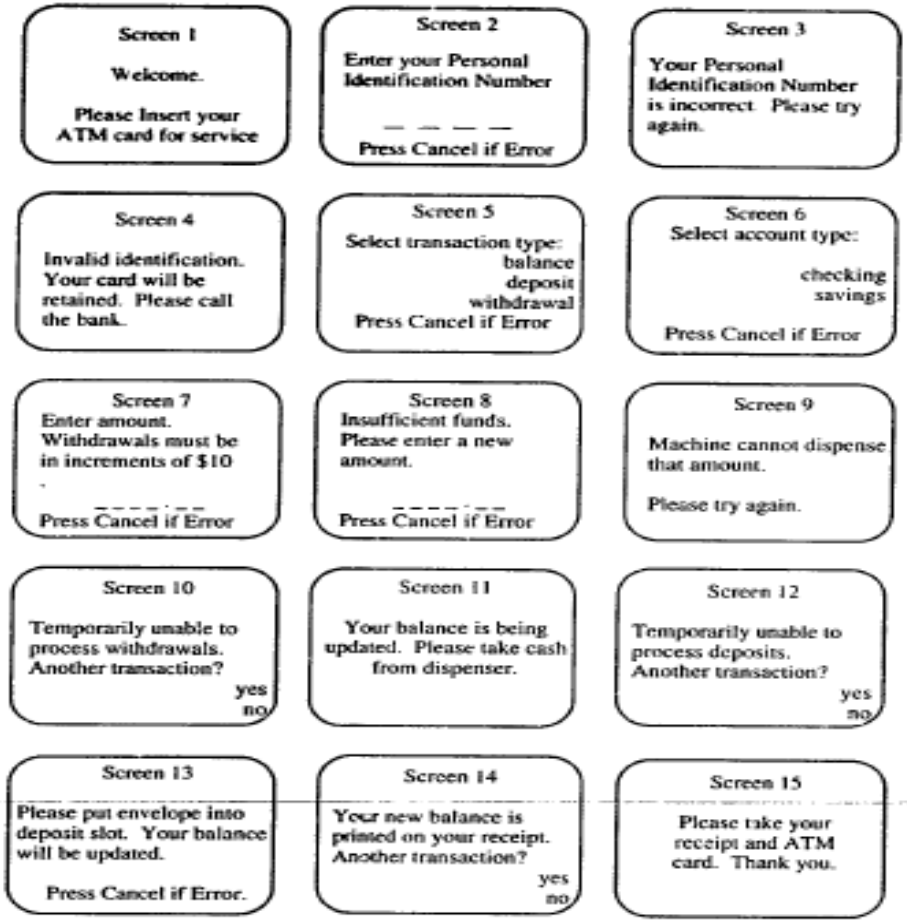


Figure 12.6 Screens for the SATM system.

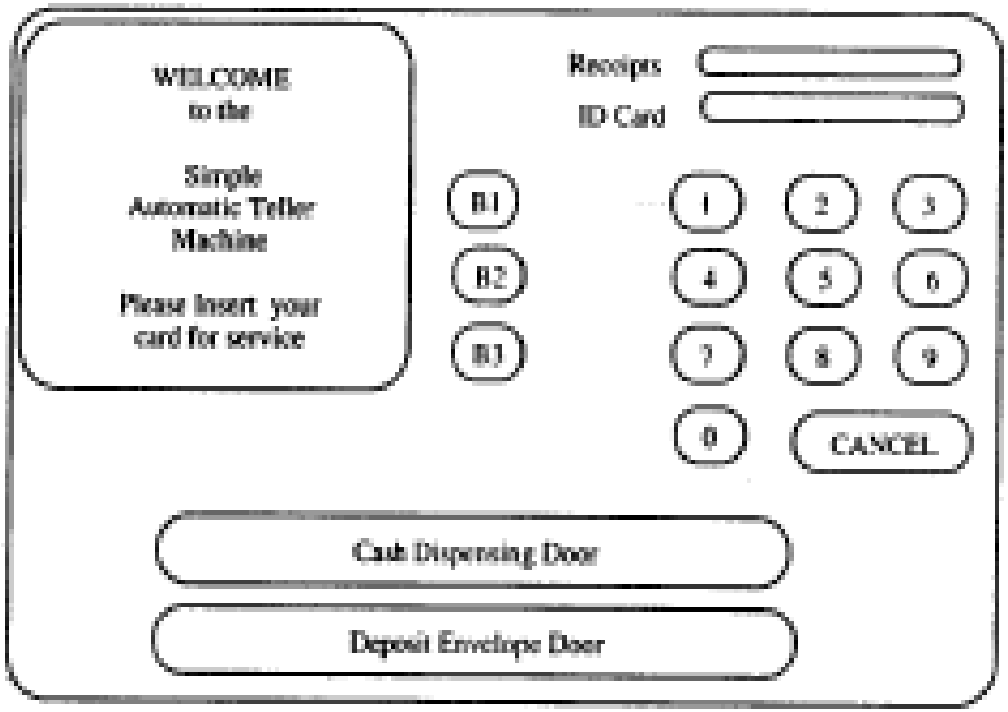


Figure 12.7 The SATM terminal.

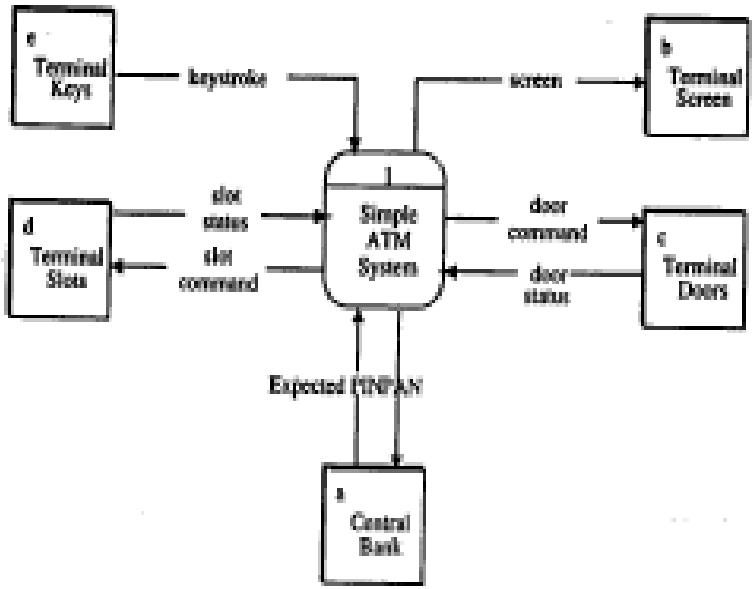


Figure 12.8 Context diagram of the SATM system.

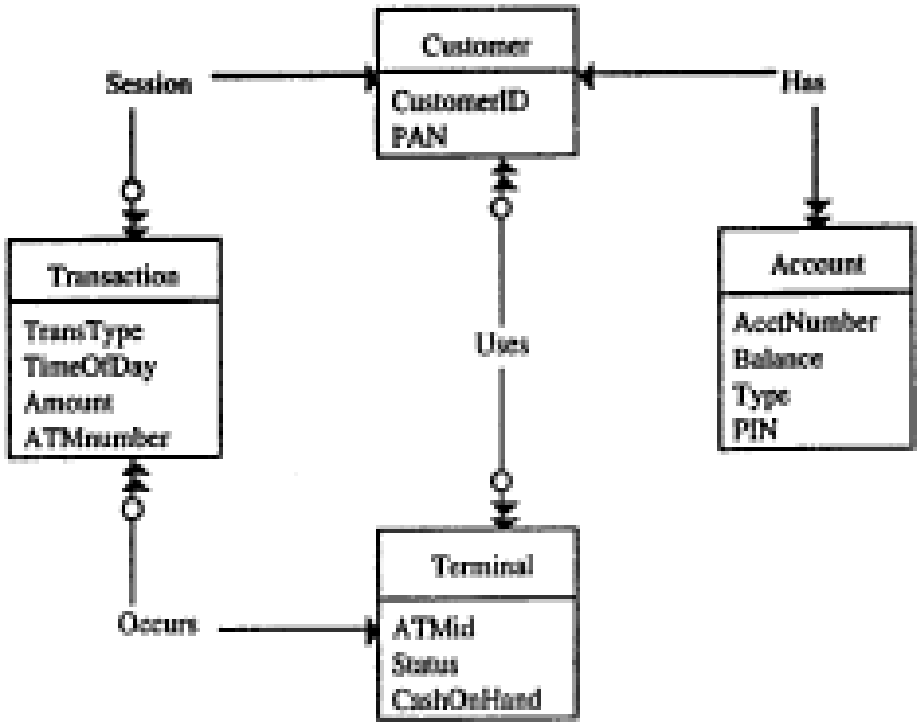


Figure 12.10 Entity/relationship model of the SATM system.

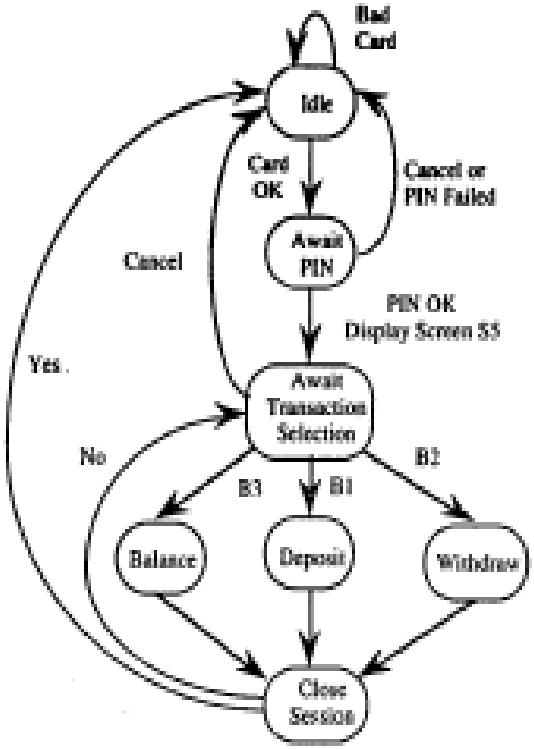


Figure 12.11 Upper-level SATM finite state machine.

```
Main Program
State = AwaitCard
Case State
Case 1: AwaitCard
            ScreenDriver(1, null)
            WatchCardSlot(CardSlotStatus)
            Do While CardSlotStatus is Idle
                WatchCardSlot(CardSlotStatus)
            End While
            ControlCardRoller(accept)
            ValidateCard(CardOK, PAN)
            If CardOK
                Then State = AwaitPIN
                Else    ControlCardRoller(eject)
            EndIf
            State = AwaitCard
Case 2: AwaitPIN
            ValidatePIN(PINok, PAN)
            If PINok
                Then    ScreenDriver(2, null)
                        State = AwaitTrans
                Else    ScreenDriver(4, null)
            EndIf
            State = AwaitCard
Case 3: AwaitTrans
            ManageTransaction
            State = CloseSession
Case 4: CloseSession
            If NewTransactionRequest
                Then    State = AwaitTrans
                Else    PrintReceipt
            EndIf
            PostTransactionLocal
            CloseSession
            ControlCardRoller(eject)
            State = AwaitCard
End Case (State)
End. (Main program SATM)
```

```
Procedure ValidatePIN(PINok, PAN)
GetPINforPAN(PAN, ExpectedPIN)
Try = First
Case Try of
Case 1: First
        ScreenDriver(2, null)
        GetPIN(EnteredPIN)
        If EnteredPIN = ExpectedPIN
            Then    PINok = True
            Else    ScreenDriver(3, null)
        EndIf
        Try = Second
Case 2: Second
        ScreenDriver(2, null)
        GetPIN(EnteredPIN)
        If EnteredPIN = ExpectedPIN
            Then    PINok = True
            Else    ScreenDriver(3, null)
        EndIf
        Try = Third
Case 3: Third
        ScreenDriver(2, null)
        GetPIN(EnteredPIN)
        If EnteredPIN = ExpectedPIN
            Then    PINok = True
            Else    ScreenDriver(4, null)
                    PINok = False
        EndIf
EndCase (Try)
End.    (Procedure ValidatePIN)
```

System Testing

Contents

1.System Testing Overview

2.Functional system Testing

3.Non-functional system Testing

System Testing overview

- Testing complete system.
- It is done by independent Tester.
- Bring in customer perspective in testing.
- Objective is to find product level defects and in building the confidence before the product is released to the customer.
- Provide a fresh pair of eyes to discover defects not found earlier by testing.

Contents

- A closer look at the SATM system
- Decomposition based Integration

Closer look at SATM SYSTEM

- The unit Calling graph is the directed graph in which nodes are program units and edges corresponds to program calls.

Table 13.1 SATM Units and Abbreviated Names

Unit Number	*Level Number*	*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
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

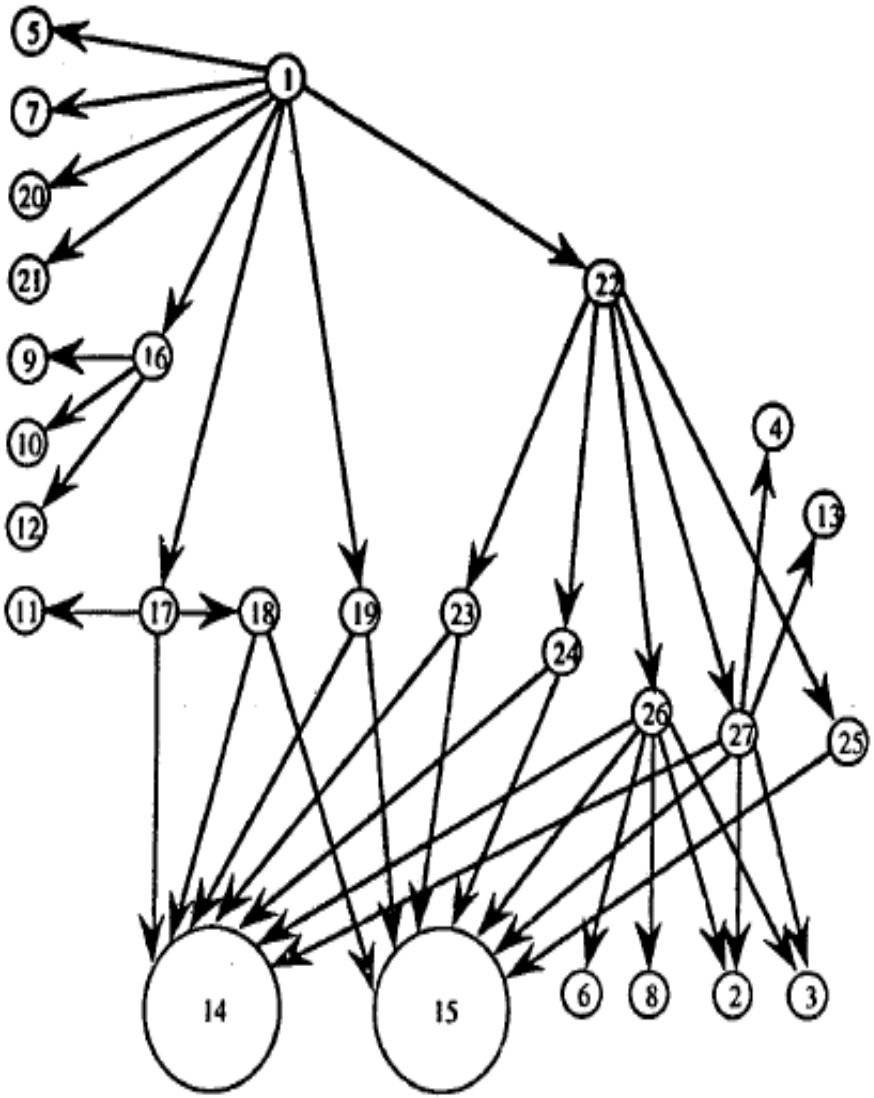


Table 13.2 Adjacency Matrix for the SATM Call Graph

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1				X		X							X		X	X		X	X	X	X					
2																										
3																										
4																										
5																										
6																										
7																										
8																										
9																										
10																										
11																										
12																										
13																										
14																										
15																										
16								X	X		X															
17										X			X				X									
18													X	X												
19													X	X												
20																										
21																										
22													X	X								X	X	X	X	X
23													X	X												
24													X	X												
25														X												
26	X	X			X		X					X	X	X												
27	X	X	X		X		X					X	X	X												

Decomposition Based Integration

- The goal is to interface Among Separately tested units.
- We can obtain four integration strategies based on functional decomposition tree

1. Top down
2. Bottom Up
3. Sandwich
4. Big bang

Top down

- Top down integration begin with the main program.
- Any lower level unit that is called by main program appear to be stub
- Stub: they are piece of throwaway code that emulate a called unit
- It follows breadth first traversal

Module	*Uses Modules*
SATM Main	WatchCardSlot
	Control Card Roller
	Screen Driver
	Validate Card
	Validate PIN
	Manage Transaction
	New Transaction Request
ValidatePIN	GetPINforPAN
	GetPIN
	Screen Driver
GetPIN	KeySensor
	Screen Driver

```
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
```

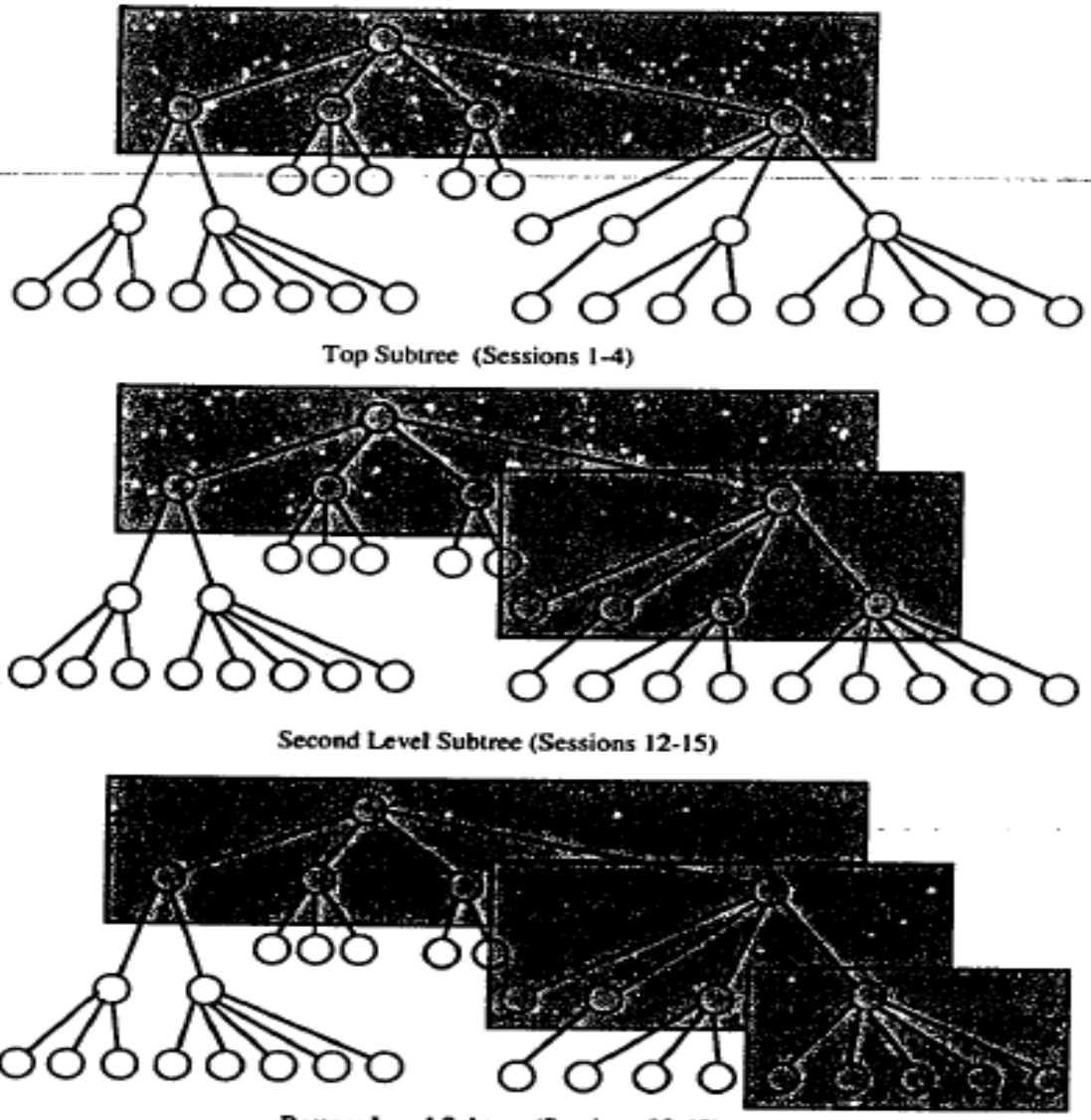


Figure 13.3 Top-down integration.

Bottom up Integration

- It’s mirror image to the top down approach.
- Only change is we use Drivers instead of stubs.

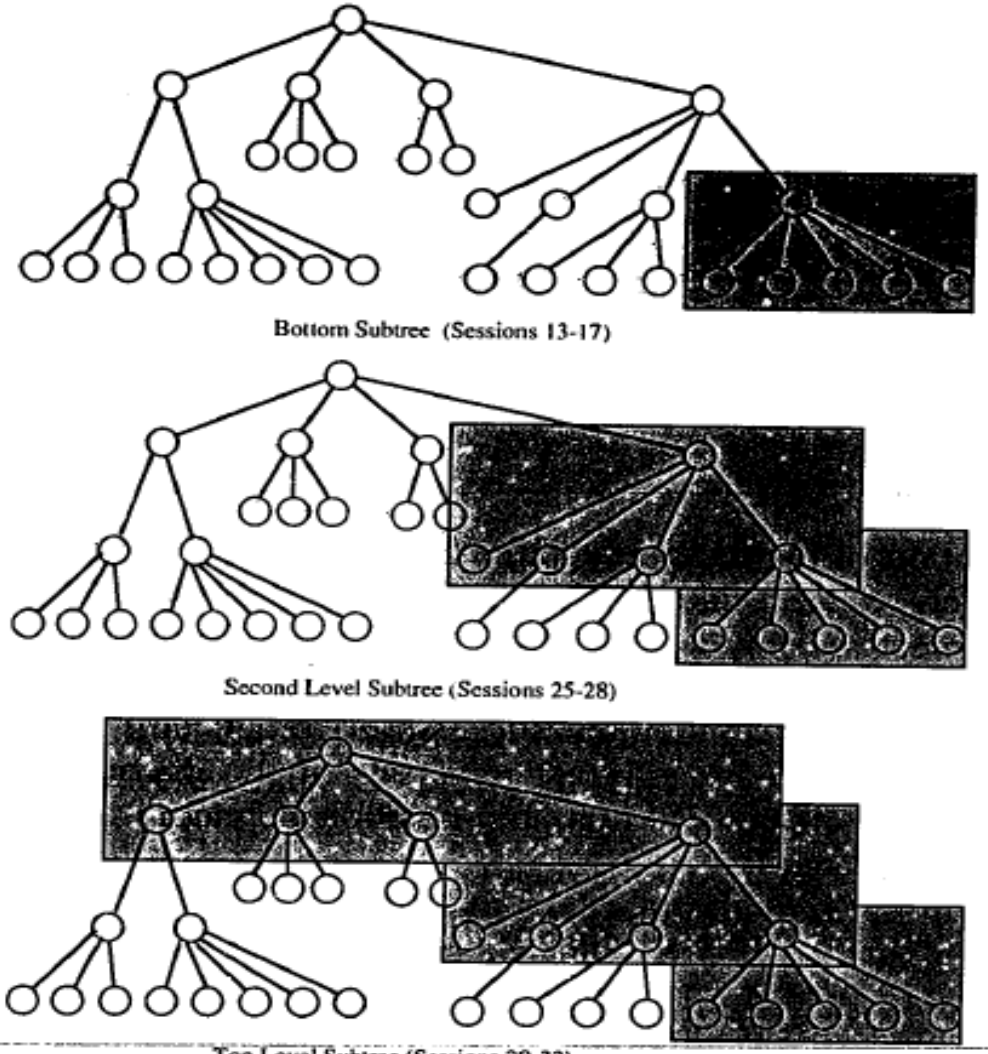


Figure 13.4 Bottom-up integration.

Top-Down Integration Mechanism

- Breadth-first traversal of the functional decomposition tree.
- First step: Check main program logic, with all called
- units replaced by stubs that always return correct values.
- Move down one level

– replace one stub at a time with actual code.

– any fault must be in the newly integrated unit

Bottom-Up Integration Mechanism

- Reverse of top-down integration
- Start at leaves of the functional decomposition tree.
- Driver units...
  - call next level unit
  - serve as a small test bed
  - “drive” the unit with inputs
  - drivers know expected outputs
- As with top-down integration, one driver unit at a time is replaced with actual code.
- Any fault is (most likely) in the newly integrated code.

Call Graph-Based Integration

- Definition: The *Call Graph* of a program is a directed graph in which
  – nodes are unit
  – edges correspond to actual program calls (or messages)
- Call Graph Integration avoids the possibility of impossible edges in decomposition-based integration.
- Can still use the notions of stubs and drivers.
- Can still traverse the Call Graph in a top-down or bottom-up strategy.

Call Graph-Based Integration (continued)

- Two strategies
  - Pair-wise integration
  - Neighborhood integration
- Degrees of nodes in the Call Graph indicate integration sessions
  - isLeap and weekDay are each used by three units
- Possible strategies
  - test high indegree nodes first, or at least,
  - pay special attention to “popular” nodes

Pair-Wise Integration

- By definition, and edge in the Call Graph refers to

an interface between the units that are the endpoints of the edge.

- Every edge represents a pair of units to test.
- Still might need stubs and drivers
- Fault isolation is localized to the pair being integrated

Neighborhood Integration

- The neighborhood (or radius 1) of a node in a graph is the set of nodes that are one edge away from the given node.
- This can be extended to larger sets by choosing larger values for the radius.
- Stub and driver effort is reduced.

Path-Based Integration

- Wanted: an integration testing level construct

similar to DD-Paths for unit testing...

– extend the symbiosis of spec-based and code-based

testing to the integration level

– greater emphasis on behavioral threads

– shift emphasis from interface testing to interactions (confutations)

among units

- Need some new definitions

– source and sink nodes in a program graph

– module (unit) execution path

– generalized message

– MM-Path