Object Oriented Testing

Ali Kamandi

(kamandi@ce.sharif.edu)

Sharif University of Technology

Object Oriented Testing

- *Find the greatest possible number of errors with a manageable amount of effort applied over a realistic time span.*
- *The nature of OO programs changes both testing strategy and testing tactics.*
- *Do we need less efforts for testing because of greater reuse of design patterns?*
- *Answer is no! Binder argues that more testing is needed to obtain high reliability in OO systems, since each reuse is a new context of usage.*

Testing Object-Oriented Applications: Why is it Different?

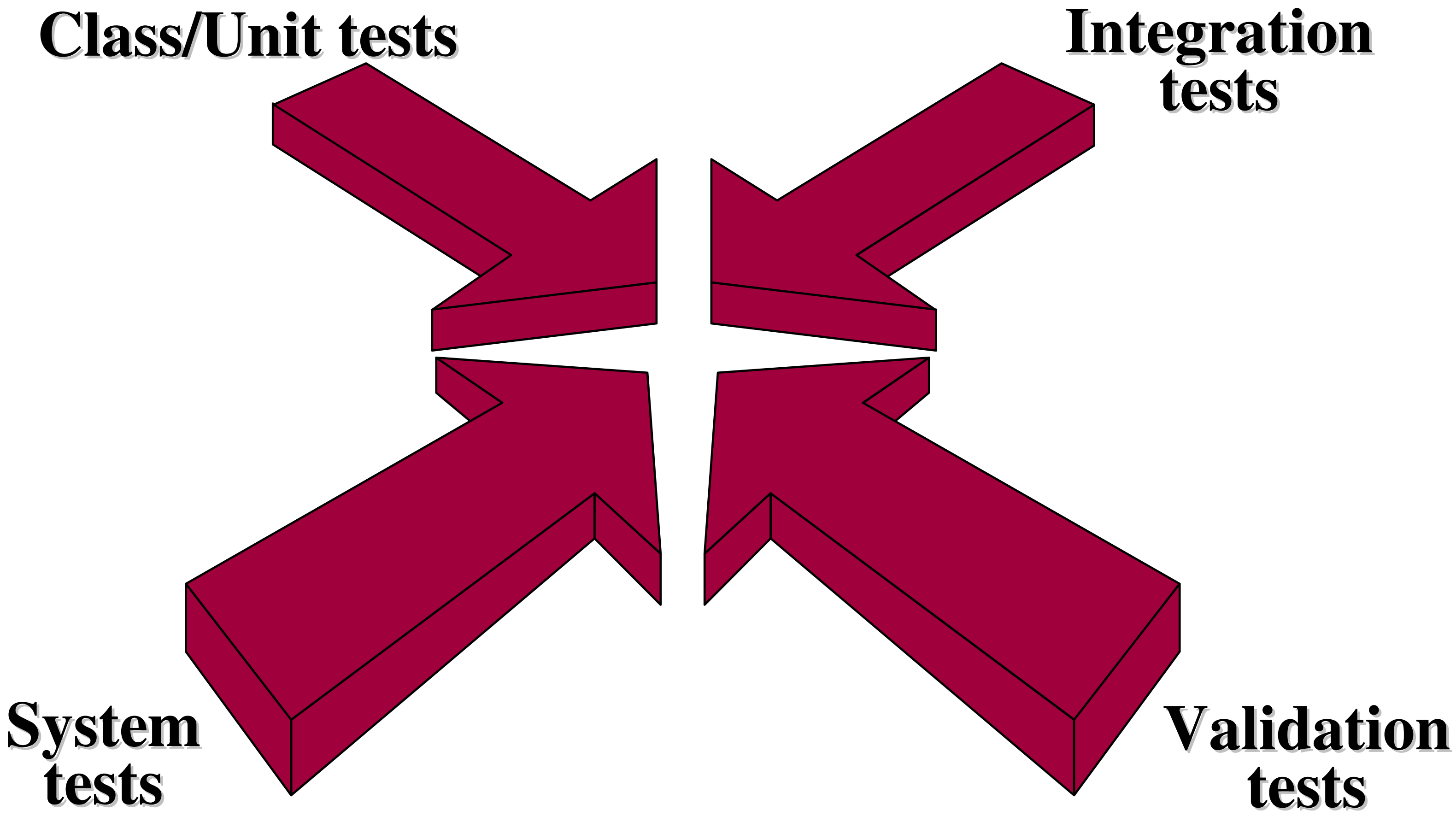
- *No sequential procedural executions*
- *No structure charts to design integration testing*
- *Iterative O-O development and its impact on testing and integration strategies*
- *In OO testing begins by evaluating the OOA and OOD models*
- *What is Unit?*
- *Encapsulation, Inheritance, Polymorphism*

Criteria for Completion of Testing

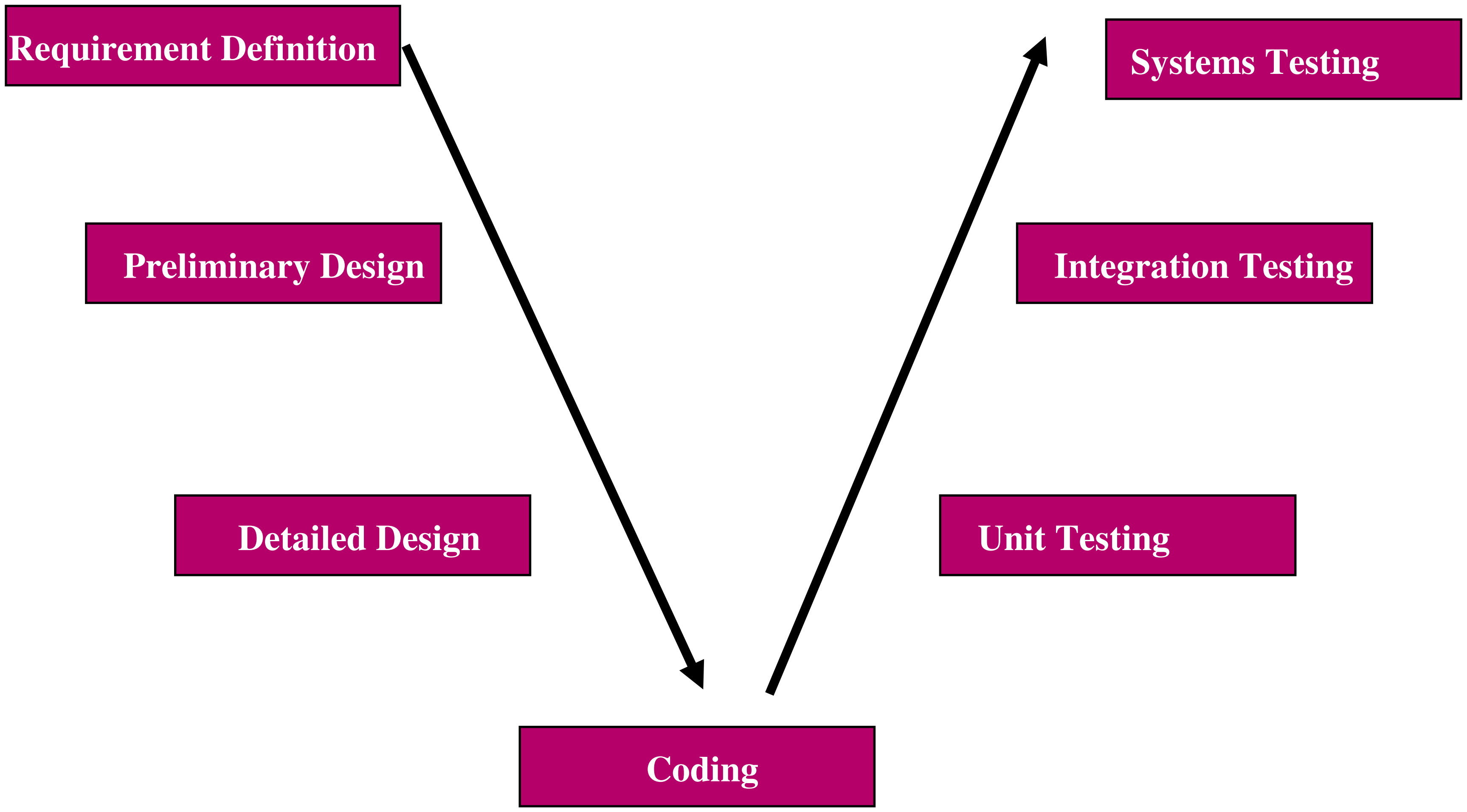
- When are we done testing?
- How to answer this question is still a research question

1. One view: testing is never done… the burden simply shifts from the developer to the customer
2. Or: testing is done when you run out of time or money
3. Or use a statistical model:
   - Assume that errors decay logarithmically with testing time
   - Measure the number of errors in a unit period
   - Fit these measurements to a logarithmic curve
   - Can then say: “with our experimentally valid statistical model we have done sufficient testing to say that with 95% confidence the probability of 1000 CPU hours of failure free operation is at least 0.995”

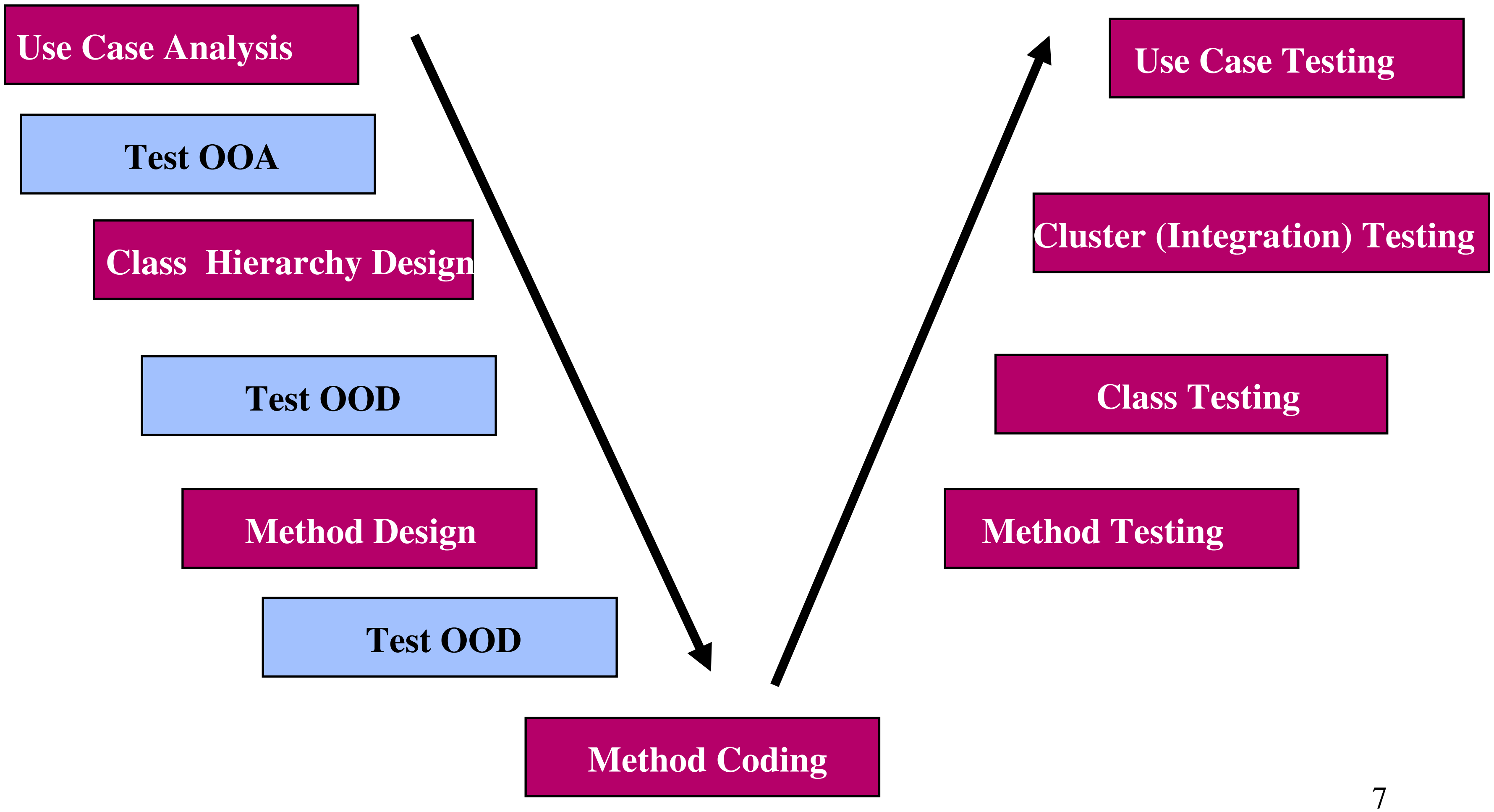
Testing OO Code



The Structured Testing Pyramid



The OO Testing Pyramid



Testing OOA and OOD Models (1)

- The review of OO analysis and design models is especially useful because the same semantic constructs (e.g., classes, attributes, operations, messages) appear at the analysis, design, and code level.
- Analysis and design models cannot be tested in the conventional sense, because they cannot be executed.

Testing OOA and OOD Models (2)

- **Formal technical review can be used to examine the correctness and consistency of both analysis and design models.**
- **Correctness:**
  - **Syntax: Each model is reviewed to ensure that proper modeling conventions have been maintained.**
  - **Semantic: Must be judged based on the model's conformance to the real world problem domain by domain experts.**
- **Consistency:**
  - **May be judged by considering the relationship among entities in the model.**
  - **Each class and its connections to other classes should be examined.**
  - **The Class-responsibility-collaboration model can be used.**
- **Completeness**

Model Testing Approaches

- Testing by comparison

  compares each model to its predecessor or to previous forms of the model

- Testing by inspection

  uses checklists to make sure that the model meets certain criteria

- Testing by verification

  follows certain steps to assure completeness and consistency of one part of the model with another

Examples of Analysis and Design Models to be Tested

- CRC cards
- Class specifications
- Use cases
- State-Transition Models (State Diagram)
  - State transition diagrams for classes, clusters, and subsystems
- Sequence Diagrams

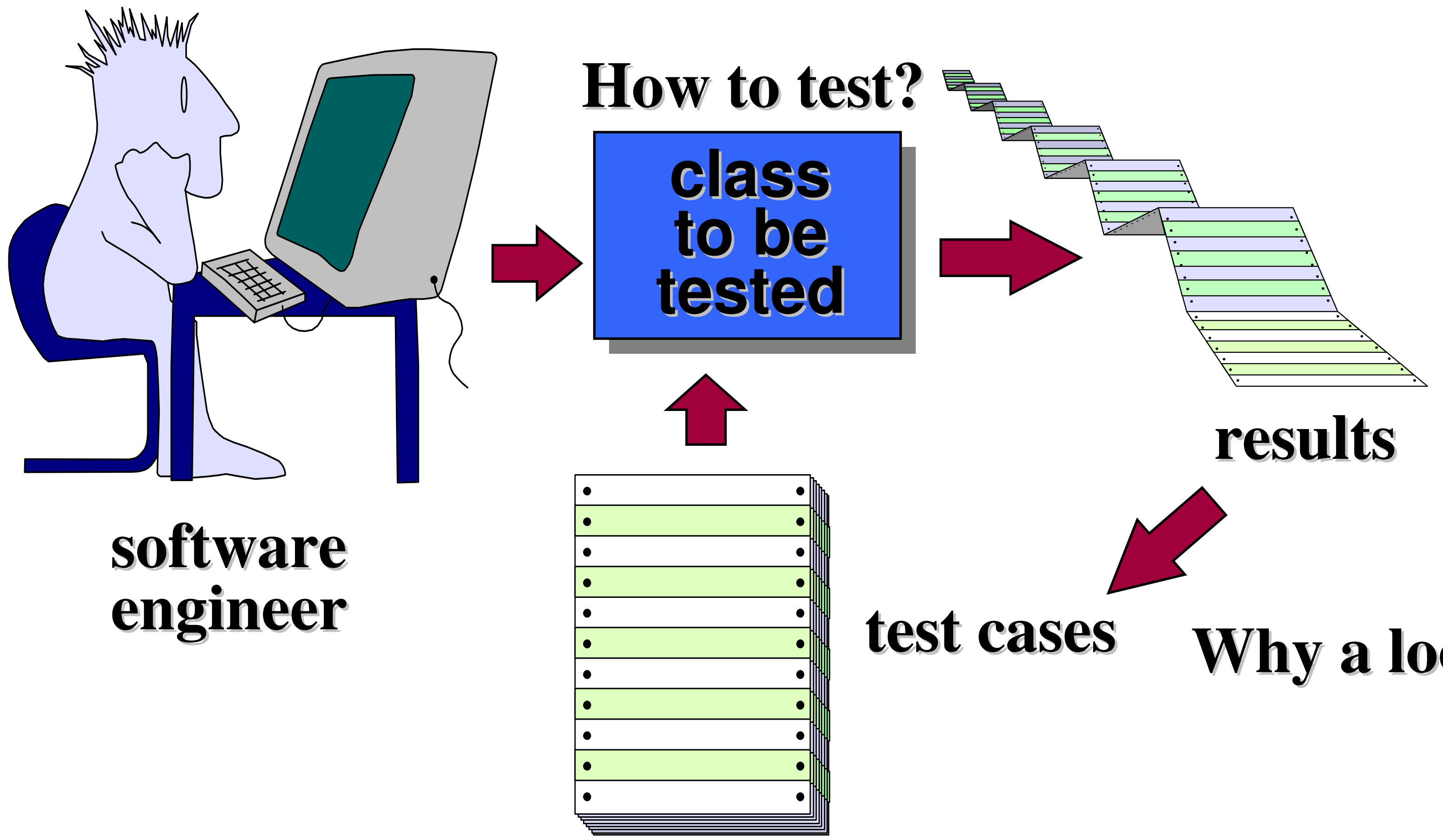
Testing the Class Model

- **CRC Cards: Check that all collaborations are properly represented.**
  - Example: in a point of sale system.
  - A ***read credit card*** responsibility of a ***credit sale*** class is accomplished if satisfied by a ***credit card*** collaborator
- **Have you tested your analysis and design?**
  - If not, who will do it?
- **These steps are applied iteratively to each class and through each evolution of the OOA model.**

Unit Test

- What is a unit?
  - A single, cohesive function?
  - A function whose code fits on one page?
  - The amount of code that can be written in 4 to 40 hours?
  - Code that is assigned to one person?
- We can no longer test a single operation in isolation but rather as part of a class.
- In object-oriented programs, a unit is a method within a class.
- Smallest testable unit is the encapsulated class

Class Testing Process



Methods for Generating Test Cases For Unit Testing

- Statement coverage
- Graph based
  - Condition coverage
  - Path coverage

- All unit testing methods are also applicable to testing methods within a class.

Class Test Case Design

Berard [BER93,94] proposes the following approach:

1. **Identify each test case uniquely**
   - Associate test case explicitly with the class and/or method to be tested
2. **State the purpose of the test**
3. **Each test case should contain:**

   a. list of specified states for the object that is to be tested

   b. A list of messages and operations that will be exercised as a consequence of the test

   c. A list of external conditions for setup (i.e., changes in the environment external to the software that must exist in order to properly conduct the test)

   d. Supplementary information that will aid in understanding or implementing the test

– **Automated unit testing tools facilitate these requirements**

Challenges of Class Testing

- OO class is the target for test case design.
- **Encapsulation:**
  - Difficult to obtain a snapshot of a class without building extra methods which display the classes' state
- **Inheritance and polymorphism:**
  - Each new context of use (subclass) requires re-testing because a method may be implemented differently (polymorphism).
  - Other unaltered methods within the subclass may use the redefined method and need to be tested
- **White box tests:**
  - Basis path, condition and loop tests can all apply to individual methods, but don't test interactions between methods

Notes on OO Testing 1.

- When an operation is invoked, it may be hard to tell exactly what code gets exercised.
- It can be hard to determine the exact type or class of a parameter.

Notes on OO Testing 2.

- **Impact of OO programming on Testing:**
  - OO operations are smaller, more time needed for integration. So integration faults become more plausible.

    In 3 large scale studies: 50% of methods are less than 2 C++ statements or 4 Smalltalk statements (Law of Demeter connection)

- **Use cases can provide useful input in the design of black-box and state-based tests.**

OOT Methods: Class Level 1

- Random testing

  identify operations applicable to a class

  define constraints on their use

  identify a minimum test sequence

    an operation sequence that defines the minimum life history of the class (object)

  generate a variety of random (but valid) test sequences

    exercise other (more complex) class instance life histories

- Example:

  Class: Account

  Operations: open, setup, deposit, withdraw, balance, summarize, creditlimit, close.

  1. *Open – setup – deposit – withdraw – close*
  2. *Open – setup – deposit –[deposit | withdraw | balance | summarize] * – withdraw – close*. Generate random test sequences using this template

OOT Methods: Class Level 2

- **Partition Testing**

  reduces the number of test cases required to test a class in much the same way as equivalence partitioning for conventional software

  state-based partitioning

  categorize and test operations based on their ability to change the state of a class (e.g.: deposit, withdraw)

  attribute-based partitioning

  categorize and test operations based on the attributes that they use (e.g.: creditlimit attribute)

  category-based partitioning

  categorize and test operations based on the generic function each performs. (e.g.: (Init OP: open , setup) (Comp. OP: deposit, withdraw) (queries: balance, summarize, creditlimit) (Termination OP: close))

What Methods to Test Within Classes

- New methods: defined in the class under test Complete testing
- Inherited methods: defined in a superclass of the class under test: Retest only if the methods interacts with new or redefined method.
- Redefined methods: defined in a superclass of but redefined in the class under test :complete Retest reusing tests from the superclass.

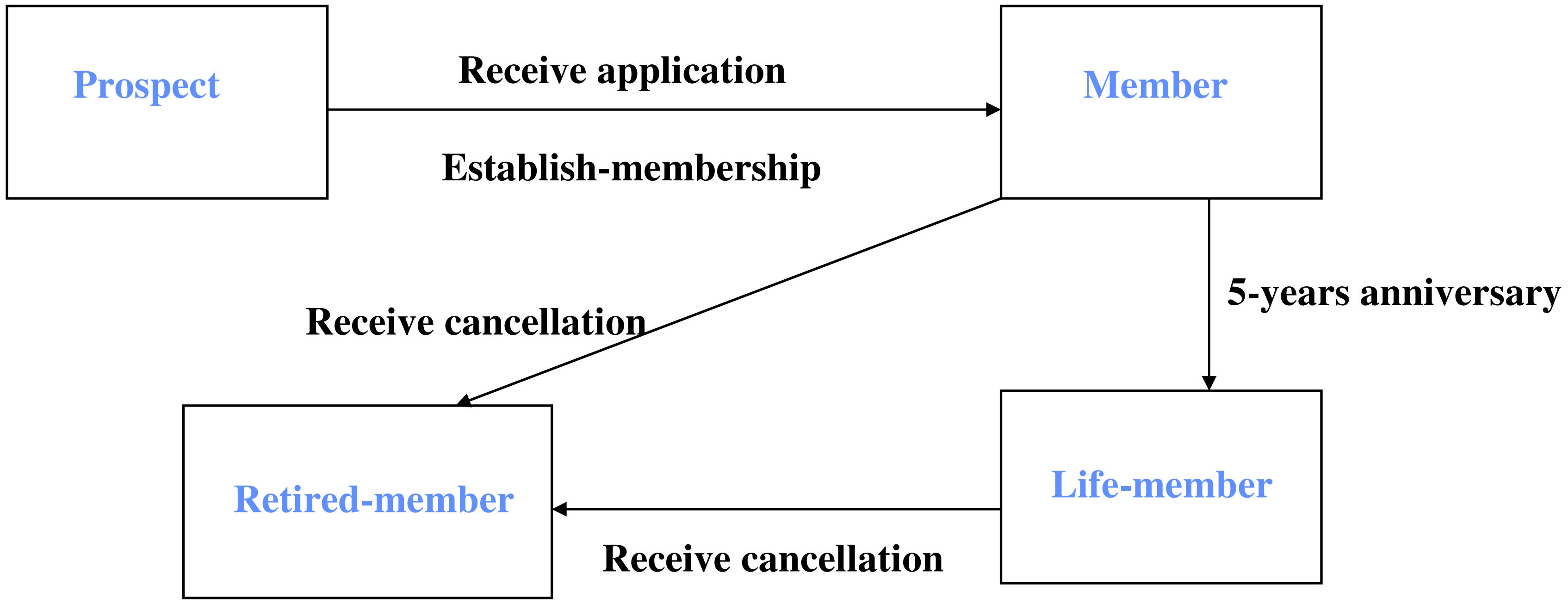
State-Transition Testing

- A state-transition model describes the different states and transitions of a class in the context of its position in the inheritance hierarchy.

  (OMT dynamic model)

- The *state* of an object is the combination of all the attribute values and objects that the object contains.
- An object may *transition* from a state to state as a result of an *event*, which may yield an *action*.

Example


Prospect
Receive application
Establish-membership
Member
5-years anniversary
Receive cancellation
Retired-member
Life-member
Receive cancellation

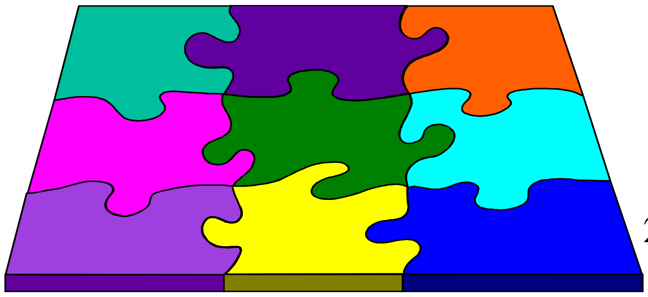
State-Transition Testing

- Create test cases corresponding to each transition path that represent a full object life cycle

- Make sure each transition is exercised at least once.

[2] Integration Testing

- OO does not have a hierarchical control structure so conventional top-down and bottom-up integration tests have little meaning
- Integrating operations one at a time into a class is often impossible because of the direct and indirect interactions of the components that make up the class.
- Integration applied three different incremental strategies:
  - Thread-based testing: integrates classes required to respond to one input or event
  - Use-based testing: integrates classes required by one use case
  - Cluster testing: integrates classes required to demonstrate one collaboration



Types of Errors Found During Integration Testing

- **Messaging errors:**

  Failure to meet a requirement, i.e., **no method to send or receive a message**

- **Incompatible method and message in sender and receiver**

- **Incorrect instantiation or destruction of objects**

Types of Errors Found During Integration Testing (cont.)

- User interface errors:

  A given sequence of user actions does not have the expected effect on the component.

Random Integration Testing

- Multiple Class Random Testing
  1. For each client class, use the list of class methods to generate a series of random test sequences. Methods will send messages to other server classes.
  2. For each message that is generated, determine the collaborating class and the corresponding method in the server object.
  3. For each method in the server object (that has been invoked by messages sent from the client object), determine the messages that it transmits
  4. For each of the messages, determine the next level of methods that are invoked and incorporate these into the test sequence

Cluster (Integration) Testing

- A cluster is a collection of classes (possibly from different systems) cooperating with each other via messaging.
- It assumes that each class has been tested individually
- Cluster testing is considered a second level of integration testing

Methods for Forming Clusters

- Function-based clustering
  - Based on requirements and use cases
  - Difficult to perform if requirements were not available during the design phase
- Subject-based clustering
  - Based on subject areas that need to test separately
- Project Schedule-based clustering

Techniques for Object-Oriented Integration Testing

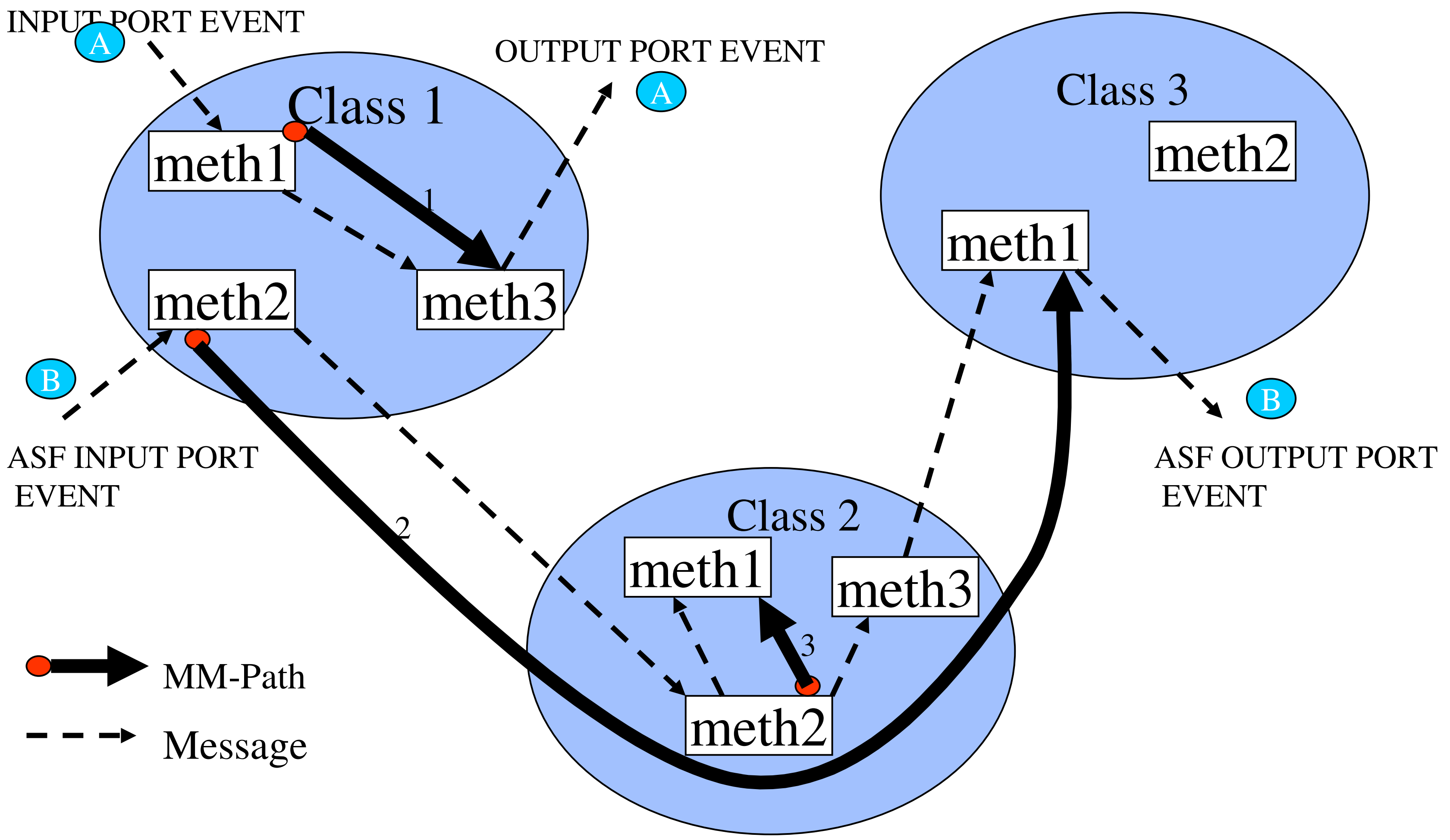
- **Message Quiescence**
- **Event Quiescence**

Message Quiescence

- A Method/Message path (MM-Path) is a sequence of method executions linked by messages.
- An MM-Path starts with a method and ends when it reaches a method that does not issue a message of its own, i.e., reaches a *message Quiescence*.

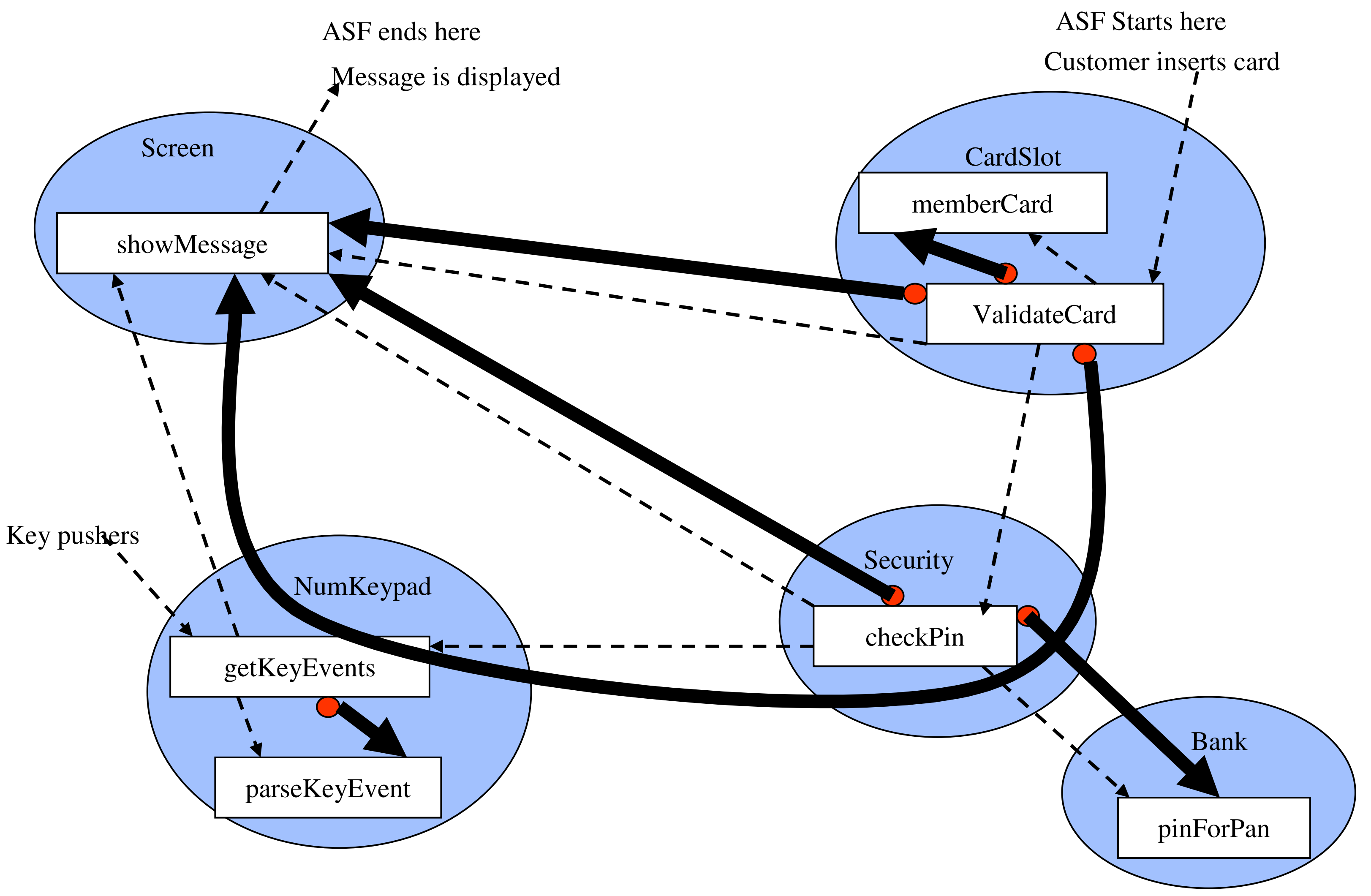
Event Quiescence

- **An input port event followed by a set of MM-Paths, and terminated by an output event is called *Atomic System Function (ASF)*.**


INPUT PORT EVENT
A
OUTPUT PORT EVENT
A
Class 1
meth1
1
meth2
meth3
B
ASF INPUT PORT EVENT
2
Class 3
meth2
meth1
B
ASF OUTPUT PORT EVENT
Class 2
meth1
meth3
3
meth2
MM-Path
Message

ATM PIN Entry

- Customer enters card (event)
- Screen requesting PIN entry is displayed
- An interleaved sequence of digit key toughs with audible and visual feedback
- Possibility of cancellation by customer
- System disposition (valid PIN or card retained)


ASF ends here
Message is displayed
ASF Starts here
Customer inserts card
Screen
showMessage
CardSlot
memberCard
ValidateCard
Key pushers
NumKeypad
getKeyEvents
parseKeyEvent
Security
checkPin
Bank
pinForPan

[3] Validation Testing

- Are we building the right product?
- Validation succeeds when software functions in a manner that can be reasonably expected by the customer.
- Focus on user-visible actions and user-recognizable outputs
- Details of class connections disappear at this level
- Apply:
  - Use-case scenarios from the software requirements spec
  - Black-box testing to create a deficiency list
  - Acceptance tests through alpha (at developer’s site) and beta (at customer’s site) testing with actual customers

[4] System Testing

- Software may be part of a larger system. This often leads to “finger pointing” by other system dev teams
- Finger pointing defence:
  1. Design error-handling paths that test external information
  2. Conduct a series of tests that simulate bad data
  3. Record the results of tests to use as evidence
- Types of System Testing:
  - Recovery testing: how well and quickly does the system recover from faults
  - Security testing: verify that protection mechanisms built into the system will protect from unauthorized access (hackers, disgruntled employees, fraudsters)
  - Stress testing: place abnormal load on the system
  - Performance testing: investigate the run-time performance within the context of an integrated system

System Testing

- All rules and methods of traditional systems testing are also applicable applicable to object-oriented.

- Use cases provide a more precise mechanism for writing test cases.

Example

Use Case # 1 of the *Telephone Banking System*:

Task:	**Making a balance transfer**
User Class:	**Current customer**
Frequency:	**one per week per customer**

User Action	*System Response*
User dials the number	System plays greeting and ask for account number
User enters invalid account number	System informs user and ask for account number again
User enter a valid account number	System asks for PIN#

Automated Testing

- Junit at Junit.org
- CPPUnit on SourceForge.net,
- NUnit for .Net
- Rational Robot

OO Testing Methodology

- Jorgensen and Erickson propose 5 levels:

A Method	Unit testing
Message Quiescence	Integration
Event Quiescence	Integration
Thread testing	System
Thread interaction	System

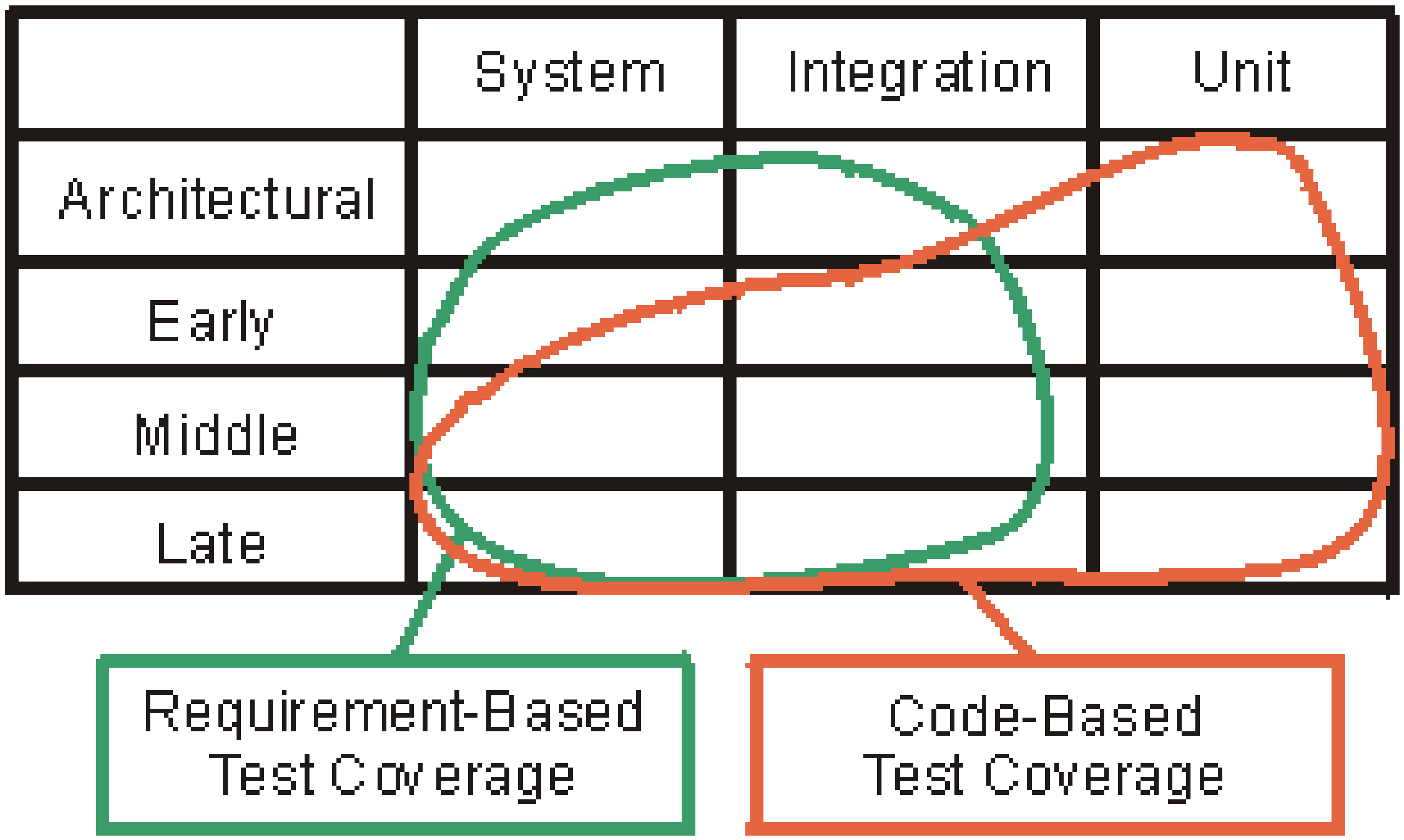
Stages of Test [RUP]

- Developer Testing
- Unit Test
- Integration Test
- System Test
- Acceptance Test

Test Planning [RUP]

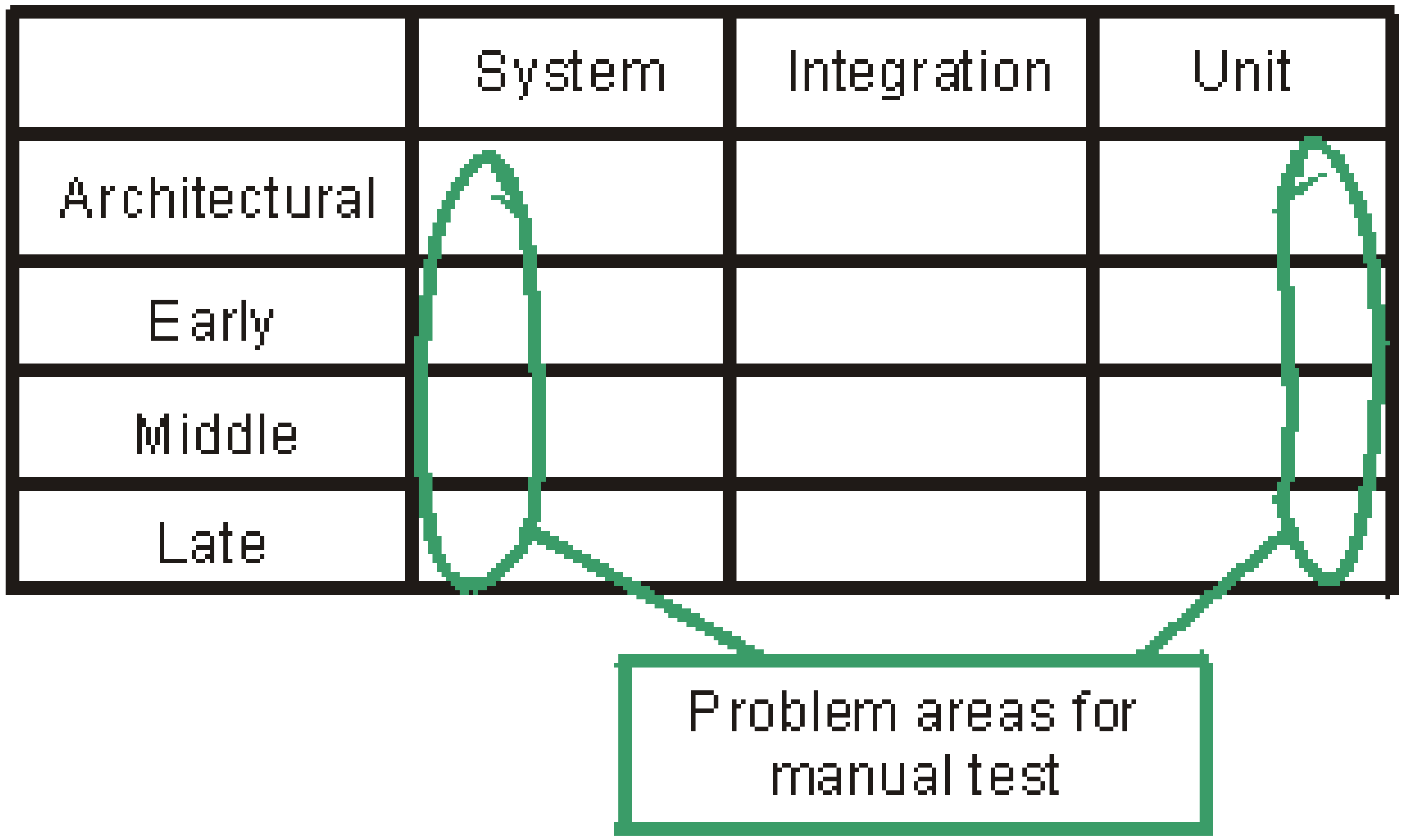
- What iteration you are in?
- What stage of test (unit test, integration test, system test) you are performing?
- Types of test (functional, stress, volume, performance, usability, distribution, and so on).
- Evaluation criteria used (code-based test coverage, requirements-based test coverage, number of defects, mean time between failure, and so on.)
- Testing techniques used (manual and automated)

Completion Criteria [RUP]



	System	Integration	Unit
Architectural			
Early			
Middle			
Late			

Manual/Automated Testing [RUP]



	System	Integration	Unit
Architectural			
Early			
Middle			
Late			

Problem areas for manual test

The END