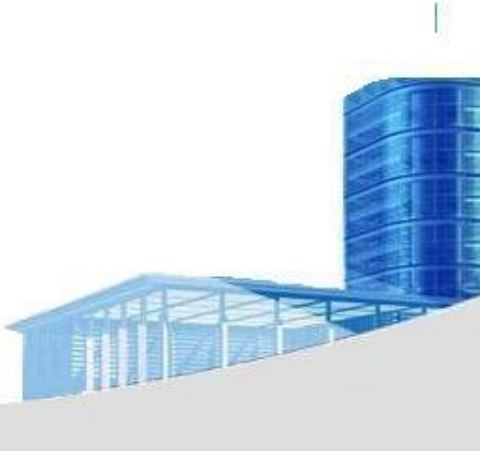
Ch.24 Testing Object-Oriented Applications



- **OO Testing**
  - To adequately test OO systems, three things must be done:
    - the definition of testing must be broadened to include error discovery techniques applied to object-oriented analysis and design models
    - the strategy for unit and integration testing must change significantly, and
    - the design of test cases must account for the unique characteristics of OO software.

- **‘Testing’ OO Models**

  - 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
  - Therefore, a problem in the definition of class attributes that is uncovered during analysis will circumvent side affects that might occur if the problem were not discovered until design or code (or even the next iteration of analysis).

- **Correctness of OO Models**
  - During analysis and design, semantic correctness can be asesssed based on the model's conformance to the real world problem domain.
  - If the model accurately reflects the real world (to a level of detail that is appropriate to the stage of development at which the model is reviewed) then it is semantically correct.
  - To determine whether the model does, in fact, reflect real world requirements, it should be presented to problem domain experts who will examine the class definitions and hierarchy for omissions and ambiguity.
  - Class relationships (instance connections) are evaluated to determine whether they accurately reflect real-world object connections.

- **Class Model Consistency**
  - Revisit the CRC model and the object-relationship model.
  - Inspect the description of each CRC index card to determine if a delegated responsibility is part of the collaborator's definition.
  - Invert the connection to ensure that each collaborator that is asked for service is receiving requests from a reasonable source.
  - Using the inverted connections examined in the preceding step, determine whether other classes might be required or whether responsibilities are properly grouped among the classes.
  - Determine whether widely requested responsibilities might be combined into a single responsibility.

- **OO Testing Strategies**
  - Unit testing
    - the concept of the unit changes
    - the smallest testable unit is the encapsulated class
    - a single operation can no longer be tested in isolation (the conventional view of unit testing) but rather, as part of a class
  - Integration Testing
    - ***Thread-based testing*** integrates the set of classes required to respond to one input or event for the system
    - ***Use-based testing*** begins the construction of the system by testing those classes (called independent classes) that use very few (if any) of server classes. After the independent classes are tested, the next layer of classes, called dependent classes
    - ***Cluster testing*** [McG94] defines a cluster of collaborating classes (determined by examining the CRC and object-relationship model) is exercised by designing test cases that attempt to uncover errors in the collaborations.

- **OO Testing Strategies**
  - Validation Testing
  - details of class connections disappear
  - draw upon use cases (Chapters 5 and 6) that are part of the requirements model
  - Conventional black-box testing methods (Chapter 18) can be used to drive validation tests

- **OOT Methods**
  - *Berard [Ber93] proposes the following approach:*

  - Each test case should be uniquely identified and should be explicitly associated with the class to be tested,
  - The purpose of the test should be stated,
  - A list of testing steps should be developed for each test and should contain [BER94]:
    - a list of specified states for the object that is to be tested
    - a list of messages and operations that will be exercised as a consequence of the test
    - a list of exceptions that may occur as the object is tested
    - a list of external conditions (i.e., changes in the environment external to the software that must exist in order to properly conduct the test)
    - supplementary information that will aid in understanding or implementing the test.

- **Testing Methods**
  - *Fault-based testing*
    - The tester looks for plausible faults (i.e., aspects of the implementation of the system that may result in defects). To determine whether these faults exist, test cases are designed to exercise the design or code.
  - *Class Testing and the Class Hierarchy*
    - Inheritance does not obviate the need for thorough testing of all derived classes. In fact, it can actually complicate the testing process.
  - *Scenario-Based Test Design*
    - Scenario-based testing concentrates on what the user does, not what the product does. This means capturing the tasks (via use-cases) that the user has to perform, then applying them and their variants as tests.

- **OOT Methods: Random Testing**

  - 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

- **OOT Methods: Partition Testing**
  - 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
    - attribute-based partitioning
      - categorize and test operations based on the attributes that they use
    - category-based partitioning
      - categorize and test operations based on the generic function each performs

- **OOT Methods: Inter-Class Testing**
  - Inter-class testing
    - For each client class, use the list of class operators to generate a series of random test sequences. The operators will send messages to other server classes.
    - For each message that is generated, determine the collaborator class and the corresponding operator in the server object.
    - For each operator in the server object (that has been invoked by messages sent from the client object), determine the messages that it transmits.
    - For each of the messages, determine the next level of operators that are invoked and incorporate these into the test sequence

• OOT Methods: Behavior Testing

The tests to be designed should achieve all state coverage [KIR94]. That is, the operation sequences should cause the Account class to make transition through all allowable states

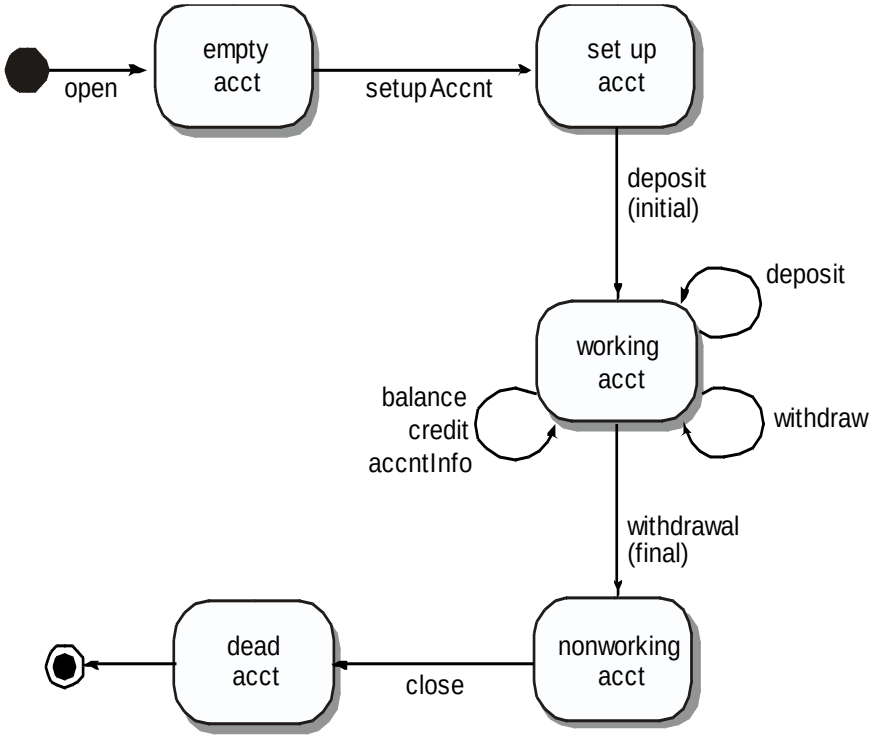


Figure 14.3 State diagram for Account class (adapted from [KIR94])