Tribhuvan University
Institute of Engineering
Pulchowk Campus
Department of Electronics and Computer Engineering

OBJECT ORIENTED SOFTWARE ENGINEERING

Chapter Six

Object Oriented Testing

by
Santosh Giri
Lecturer, IOE, Pulchowk Campus.

Overview

Overview

What is it?

- The objective of testing, stated simply, is to find the greatest possible number of errors with a manageable amount of effort applied over a realistic time span.
- Each elements of an Object oriented system (subsystems and classes) perform functions that help to achieve system requirements.
- So, It is necessary to test an OO system at a variety of different levels in an effort to uncover errors that may occur as classes collaborate with one another and subsystems communicate across architectural layers.

Who does it?

- Object-oriented testing is performed by software engineers and testing specialists.

Overview

Why is it important?

- You have to execute the program before it gets to the customer with the specific intent of removing all errors, so that the customer will not experience the frustration associated with a poor-quality product.
- In order to find the highest possible number of errors, tests must be conducted systematically and test cases must be designed using disciplined techniques.

What are the steps?

- Testing begins with the review different OO analysis and design models.
- Once code has been generated, OO testing begins “in the small” with class testing. A series of tests are designed that exercise class operations and examine whether errors exist as one class collaborates with other classes.
- As classes are integrated to form a subsystem, testing like thread-based, use-based, and cluster testing, along with fault-based approaches, are applied to fully exercise collaborating classes.
- Finally, use-cases (developed as part of the OO analysis model) are used to uncover errors at the software validation level.

Overview

What is the work product?

- A set of test cases to exercise classes, their collaborations, and behaviors is designed and documented
- Expected results defined; and actual results recorded.

OO Testing strategies

OO testing strategies

Unit Testing

- When object-oriented software is considered, the concept of the unit changes.
- Encapsulation drives the definition of classes and objects. This means that each class and each instance of a class (object) packages attributes (data) and the operations (also known as methods or services) that manipulate these data.
- Rather than testing an individual module(as in classical approach), the smallest testable unit is the encapsulated class or object.
- Class testing for OO software is the equivalent of unit testing for conventional software.

Integration Testing

There are two different strategies for integration testing of OO systems:

i. Thread-based testing

ii. Use-based testing

OO testing strategies

Thread-based testing

- ✓ It integrates the set of classes required to respond to one input or event for the system.
- ✓ Each thread is integrated and tested individually.

Use-based testing

- ✓ It begins the construction of the system by testing independent classes.
- ✓ After the independent classes are tested, the next layer of classes, called dependent classes, that use the independent classes are tested.
- ✓ This sequence of testing layers of dependent classes continues until the entire system is constructed.

Validation/system testing

- ❖ At the validation or system level, the details of class connections disappear.
- ❖ Like conventional validation, the validation of OO software focuses on user-visible actions and user-recognizable output from the system.

Test case design

For

OO software

Test case design for OO software

An overall approach to OO test case design has been defined as:

1. Each test case should be uniquely identified and explicitly associated with the class to be tested.
2. The purpose of the test should be stated.
3. A list of testing steps should be developed for each test and should contain
   - ✓ 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.

Test case design for OO software

Fault-Based Testing

- The objective of fault-based testing within an OO system is to design tests that have a high likelihood of uncovering plausible faults.
- The tester looks for plausible faults and to determine whether these faults exist, test cases are designed to exercise the design or code.
- Consider a simple example

Software engineers often make errors at the boundaries of a problem. For example, when testing a SQRT operation that returns errors for negative numbers, we know to try the boundaries: ***a negative number close to zero and zero itself***. "Zero itself" checks whether the programmer made a mistake like

if (x > 0)

calculate_the_square_root();

instead of the correct statement

if (x >= 0)

calculate_the_square_root();

Test case design for OO software

Scenario-Based Test Design

- Fault-based testing misses two main types of errors:
  1. incorrect specifications and
  2. interactions among subsystems.
- When errors associated with incorrect specification occur, the product doesn't do what the customer wants. It might do the wrong thing or it might omit important functionality.
- But in either circumstance, quality (conformance to requirements) suffers.
- Errors associated with subsystem interaction occur when the behavior of one subsystem creates circumstances (e.g., events, data flow) that cause another subsystem to fail.
- 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.
- test cases are more complex than fault-based tests.

Test case design for OO software

Scenario-Based Test Design (example),

Consider the design of scenario-based tests for a text editor.

Scenario 1

Use-Case:

Fix the Final Draft

Background*:*

The normal scenario is to print the "final" draft, read it, and discover some annoying errors that weren't obvious from the on-screen image. This use-case describes the sequence of events that occurs when this happens.

1. Print the entire document.
2. Move around in the document, changing certain pages (if any).
3. As each page is changed, it's printed.
4. Sometimes a series of pages is printed (if required).

According to above case, the user needs are:

(1) a method for printing single page

(2) a method for printing a range of pages.

Test case design for OO software

Scenario-Based Test Design (example),

- Consider the design of scenario-based tests for a text editor.

Scenario 2

Use-Case:

Print a New Copy

Background:

Someone asks the user for a fresh copy of the document. It must be printed.

1. Open the document.

2. Print it.

3. Close the document.

- In many modern editors, documents remember how they were last printed. By default, they print the same way next time.
- After the **Fix the Final Draft** scenario, just selecting "Print" in the menu and clicking the "Print" button in the dialog box will cause the last corrected page to print again

Test case design for OO software

Scenario-Based Test Design (example),

- For above case, the correct scenario should be like this:

  Use-Case:

  Print a New Copy

  1. Open the document.

  2. Select "Print" in the menu.

  3. "Check" if you're printing a page range; if so, click to print the range of page.

  4. Click on the Print button.

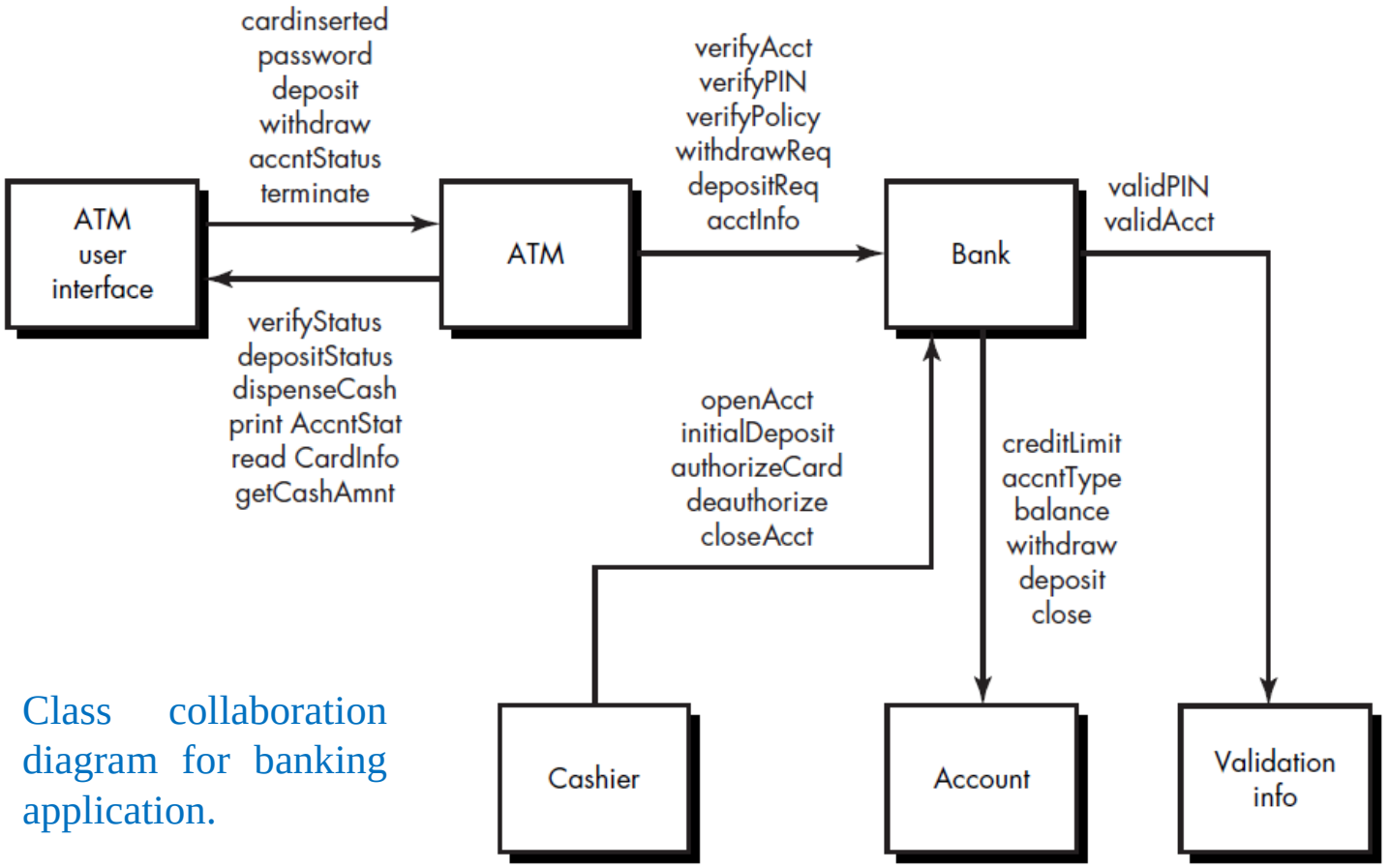
  5. Close the document.

- Still, this scenario indicates a potential specification error. For example, if Customers overlook the check noted in step 3 then printer will print one page (last corrected page) instead of range of pages (suppose 100 pages).
- So the customer will be annoyed when he/she finds one page when he/she wanted was 100 pages. This indicates Annoyed customers signal specification bugs.

Interclass test case design

Inter class test case design

- Interclass testing is the testing of a set of classes composing a system or subsystem, usually performed during integration.
- Test case design becomes more complicated as integration of the OO system begins.



Class collaboration diagram for banking application.

Inter class test case design

- To illustrate “interclass test case design”, we take example of banking system as shown in figure (in previous slide), which include the classes and collaborations between those classes.
- The direction of the arrows in the figure indicates the direction of messages and the labeling indicates the operations that are invoked as a consequence of the collaborations implied by the messages.

Approaches for inter class testing

i. **Multiple Class Testing**
ii. **Tests Derived from Behavior Models**

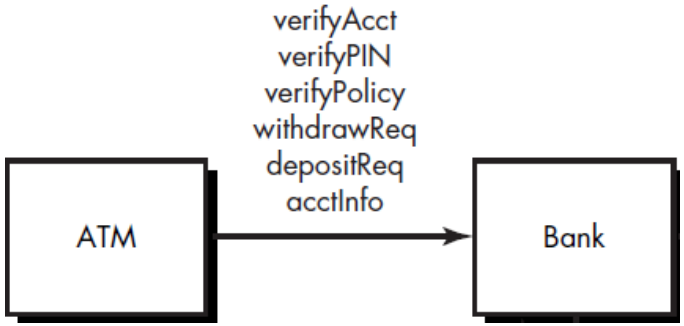
Inter class test case design

Multiple Class Testing

Steps to generate multiple class random test cases:

1. For each client class, use the list of class operations to generate a series of random test sequences. The operations will send messages to other server classes.
2. For each message that is generated, determine the collaborator class and the corresponding operation in the server object.
3. For each operation 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 operations that are invoked and incorporate these into the test sequence.

Consider a sequence of operations for the **bank** class relative to an **ATM** class as shown in figure.



Inter class test case design

Multiple Class Testing

- General sequence of operations for the **bank** class relative to an **ATM** class:

  verifyAcct•verifyPIN•[[verifyPolicy•withdrawReq]|depositReq|acctInfoREQ]^n

- So, A random test case for the **bank** class might be

  test case r_3: *verifyAcct•verifyPIN•depositReq*

- In order to consider the collaborators involved in this ***test case (r_3),*** the messages associated with each of the operations noted in test case r3 is considered. In *test case* r_3,
  - ***Bank*** must collaborate with ***ValidationInfo*** to execute the verifyAcct and verifyPIN.
  - **Bank** must collaborate with **account** to execute depositReq.

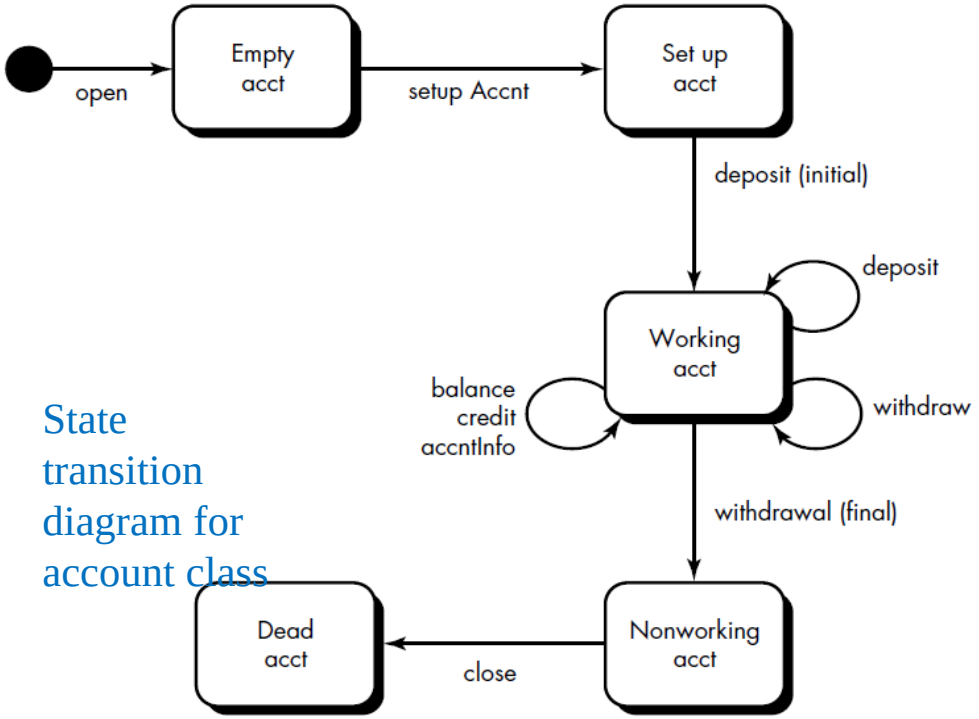
- Hence, a new test case (suppose r_4) that exercises these collaborations is

  r_4: $verifyAcct_{Bank}[validAcct_{ValidationInfo}]$•$verifyPIN_{Bank}$• $[validPin_{ValidationInfo}]$•depositReq•$[deposit_{account}]$

Inter class test case design

Tests Derived from Behavior Models

- As we discussed the use of the state transition diagram is a model that represents the dynamic behavior of a class.
- The STD for a class can be used to help derive a sequence of tests that will exercise the dynamic behavior of the class (and those classes that collaborate with it).
- Figure below illustrates an STD for the **account** class.



State transition diagram for account class

Inter class test case design

Tests Derived from Behavior Models

- Referring to the figure, initial transitions move through the empty acct and setup acct states. the majority of all behavior for instances of the class occurs while in the working acct state.
- A final withdrawal and close cause the account class to make transitions to the nonworking acct and dead acct states, respectively.
- So the tests to be designed should achieve all states i.e., the operation sequences should cause the account class to make transition through all allowable states. So the minimum test sequence should be:

  test case s1: open•setupAccnt•deposit (initial)•withdraw (final)•close

- Additional test sequences can be:

  test case s2: *open•setupAccnt•deposit(initial)•deposit•balance•credit•withdraw (final)•close*

  test case s3: *open•setupAccnt•deposit(initial)•deposit•withdraw•accntInfo•withdraw(final) •close*

- Still more test cases could be derived to ensure that all behaviors for the class have been adequately exercised. In situations in which the class behavior results in a collaboration with one or more classes, multiple STDs are used to track the behavioral flow of the system.