**Object-Oriented Testing**

BY DINESH THAKUR Category: Software Engineering

The shift from traditional to object-oriented environment involves looking at and reconsidering old strategies and methods for testing the software. The traditional programming consists of procedures operating on data, while the object-oriented paradigm focuses on objects that are instances of classes. In object-oriented (OO) paradigm, software engineers identify and specify the objects and services provided by each object. In addition, interaction of any two objects and constraints on each identified object are also determined. The main advantages of OO paradigm include increased reusability, reliability, interoperability, and extendibility.

With the adoption of OO paradigm, almost all the phases of software development have changed in their approach, environments, and tools. Though OO paradigm helps make the designing and development of software easier, it may pose new kind of problems. Thus, testing of software developed using OO paradigm has to deal with the new problems also. Note that object-oriented testing can be used to test the object-oriented software as well as conventional software.

OO program should be tested at different levels to uncover all the errors. At the algorithmic level, each module (or method) of every class in the program should be tested in isolation. For this, white-box testing can be applied easily. As classes form the main unit of object-oriented program, testing of classes is the main concern while testing an OO program. At the class level, every class should be tested as an individual entity. At this level, programmers who are involved in the development of class conduct the testing. Test cases can be drawn from requirements specifications, models, and the language used. In addition, structural testing methods such as boundary value analysis are extremely used. After performing the testing at class level, cluster level testing should be performed. As classes are collaborated (or integrated) to form a small subsystem (also known as cluster), testing each cluster individually is necessary. At this level, focus is on testing the components that execute concurrently as well as on the interclass interaction. Hence, testing at this level may be viewed as integration testing where units to be integrated are classes. Once all the clusters in the system are tested, system level testing begins. At this level, interaction among clusters is tested.

Usually, there is a misconception that if individual classes are well designed and have proved to work in isolation, then there is no need to test the interactions between two or more classes when they are integrated. However, this is not true because sometimes there can be errors, which can be detected only through integration of classes. Also, it is possible that if a class does not contain a bug, it may still be used in a wrong way by another class, leading to system failure.

**Developing Test Cases in Object-oriented Testing**

The methods used to design test cases in OO testing are based on the conventional methods. However, these test cases should encompass special features so that they can be used in the object-oriented environment. The points that should be noted while developing test cases in an object-oriented environment are listed below.

1. It should be explicitly specified with each test case which class it should test.
2. Purpose of each test case should be mentioned.
3. External conditions that should exist while conducting a test should be clearly stated with each test case.
4. All the states of object that is to be tested should be specified.
5. Instructions to understand and conduct the test cases should be provided with each test case.

**Object-oriented Testing Methods**

As many organizations are currently using or targeting to switch to the OO paradigm, the importance of OO software testing is increasing. The methods used for performing object-oriented testing are discussed in this section.

                                                      

State-based testing is used to verify whether the methods (a procedure that is executed by an object) of a class are interacting properly with each other. This testing seeks to exercise the transitions among the states of objects based upon the identified inputs.

For this testing, finite-state machine (FSM) or state-transition diagram representing the possible states of the object and how state transition occurs is built. In addition, state-based testing generates test cases, which check whether the method is able to change the state of object as expected. If any method of the class does not change the object state as expected, the method is said to contain errors.

To perform state-based testing, a number of steps are followed, which are listed below.

1. Derive a new class from an existing class with some additional features, which are used to examine and set the state of the object.
2. Next, the test driver is written. This test driver contains a main program to create an object, send messages to set the state of the object, send messages to invoke methods of the class that is being tested and send messages to check the final state of the object.
3. Finally, stubs are written. These stubs call the untested methods.

**Fault-based Testing**

Fault-based testing is used to determine or uncover a set of plausible faults. In other words, the focus of tester in this testing is to detect the presence of possible faults. Fault-based testing starts by examining the analysis and design models of OO software as these models may provide an idea of problems in the implementation of software. With the knowledge of system under test and experience in the application domain, tester designs test cases where each test case targets to uncover some particular faults.

The effectiveness of this testing depends highly on tester experience in application domain and the system under test. This is because if he fails to perceive real faults in the system to be plausible, testing may leave many faults undetected. However, examining analysis and design models may enable tester to detect large number of errors with less effort. As testing only proves the existence and not the absence of errors, this testing approach is considered to be an effective method and hence is often used when security or safety of a system is to be tested.

Integration testing applied for OO software targets to uncover the possible faults in both operation calls and various types of messages (like a message sent to invoke an object). These faults may be unexpected outputs, incorrect messages or operations, and incorrect invocation. The faults can be recognized by determining the behavior of all operations performed to invoke the methods of a class.

**Scenario-based Testing**

Scenario-based testing is used to detect errors that are caused due to incorrect specifications and improper interactions among various segments of the software. Incorrect interactions often lead to incorrect outputs that can cause malfunctioning of some segments of the software. The use of scenarios in testing is a common way of describing how a user might accomplish a task or achieve a goal within a specific context or environment. Note that these scenarios are more context- and user specific instead of being product-specific. Generally, the structure of a scenario includes the following points.

1. A condition under which the scenario runs.
2. A goal to achieve, which can also be a name of the scenario.
3. A set of steps of actions.
4. An end condition at which the goal is achieved.
5. A possible set of extensions written as scenario fragments.

Scenario- based testing combines all the classes that support a use-case (scenarios are subset of use-cases) and executes a test case to test them. Execution of all the test cases ensures that all methods in all the classes are executed at least once during testing. However, testing all the objects (present in the classes combined together) collectively is difficult. Thus, rather than testing all objects collectively, they are tested using either top-down or bottom-up integration approach.

This testing is considered to be the most effective method as scenarios can be organized in such a manner that the most likely scenarios are tested first with unusual or exceptional scenarios considered later in the testing process. This satisfies a fundamental principle of testing that most testing effort should be devoted to those paths of the system that are mostly used.

**Challenges in Testing Object-oriented Programs**

Traditional testing methods are not directly applicable to OO programs as they involve OO concepts including encapsulation, inheritance, and polymorphism. These concepts lead to issues, which are yet to be resolved. Some of these issues are listed below.

1. Encapsulation of attributes and methods in class may create obstacles while testing. As methods are invoked through the object of corresponding class, testing cannot be accomplished without object. In addition, the state of object at the time of invocation of method affects its behavior. Hence, testing depends not only on the object but on the state of object also, which is very difficult to acquire.
2. Inheritance and polymorphism also introduce problems that are not found in traditional software. Test cases designed for base class are not applicable to derived class always (especially, when derived class is used in different context). Thus, most testing methods require some kind of adaptation in order to function properly in an OO environment.