ISSN:2229-6093

Prashant ,Int.J.Computer Technology & Applications,Vol 3 (2), 746-749

Object Oriented Testing Techniques: Survey and Challenges

Prashant,
Dept. Of I.T.,
Gurgaon College of Engg.,
Gurgaon, Haryana.
prashantvats12345@gmail.com

Abstract**:** Object-oriented programs involve many unique features that are not present in their conventional counterparts. Examples are message passing, synchronization, dynamic binding, object instantiation, persistence, encapsulation, inheritance, and polymorphism. Testing for such program is, therefore, more difficult than that for conventional programs. Object-orientation has rapidly become accepted as the preferred paradigm for large-scale system design. In this paper we have discussed about how testing is being carried out in the Object Oriented environment. To accommodate this, several new techniques have been proposed like fault-based techniques, Scenario based, Surface structure testing, and Deep structural testing.

Keywords-F*ault-based Testing, Scenario-based Testing, Surface Structure Testing***.***

1. INTRODUCTION

The testing of software is an important means of assessing the software to determine its Quality. With the development of Fourth generation languages (4GL), which speeds up the implementation process, the proportion of time devoted to testing increased. As the amount of maintenance and upgrade of existing systems grow, significant amount of testing will also be needed to verify systems after changes are made [1]. Most testing techniques were originally developed for the imperative programming paradigm, with relative less consideration to object-oriented features such as message passing, synchronization, dynamic binding, object instantiation, persistence, encapsulation, inheritance, and polymorphism. Objects may interact with one another with unforeseen combinations and invocations. The testing of concurrent object-oriented systems has become a most challenging task. Object-orientation has rapidly become accepted as the preferred paradigm for large scale system design. The reasons for this are well known and understood. First, classes provide an excellent structuring mechanism. They allow a system to be divided into well-defined units, which may then be implemented separately. Second, classes support information hiding. Third, object-orientation encourages and supports software reuse. This may be achieved either through the simple reuse of a class in a library, or via inheritance, whereby a new class may be created as an extension of an existing one [2]. These might cause some types of faults that are difficult to detect using traditional testing techniques. To overcome these deficiencies, it is necessary to adopt an object-oriented testing technique that takes these features into account.

2. TROUBLE MAKERS OF OBJECT ORIENTED SOFTWARE

Following are trouble makers of OO Software

2.1 *Encapsulation* A wrapping up of data and functions into a single unit is known as encapsulation. This restricts visibility of object states and also restricts observability of intermediate test results. Fault discovery is more difficult in this case.

2.2 *Polymorphism* Polymorphism is one of the crucial features of OOP. It simply means one name multiple forms. Because of polymorphism, all possible bindings have to be tested. All potential execution paths and potential errors have to be tested.

2.3 *Inheritance* The mechanism of deriving a new class from an old one is called inheritance. The old class is referred to as the base class and the new one is called the derived class or the subclass. Inheritance results in invisible dependencies between super/sub-classes. Inheritance results in reduced code redundancy, which results in increased code dependencies. If the function is erroneous in the base class, it will be inherited in the derived class too. A subclass can't be tested without its super classes.

3. THE TEST MODEL AND ITS CAPABILITIES

The tools for automated testing are based upon certain models of software/programs and algorithms. This mathematically defined test model consists of following types of diagrams:

3.1 Class Diagram: A class diagram or an object relation diagram (ORD) represents the relationships between the various classes and its type. Types of relationships are mainly: inheritance, aggregation, and association. In object oriented programs there are three different relationships between classes they are inheritance, aggregation and association.

3.2 Control Flow Graph: A control flow graph represents the control structure of a member function and its interface to other member functions so that a tester will know which is used and/or updated and which other functions are invoked by the member function.

3.3 State Transition Diagram**:** A STD or an Object State Diagram (OSD) represents the state behavior of an object class. Now the state of a class is embodied in its member variables which are shared among its methods. The OSD shows the various states of a class (various member variable values), and transitions between them (method invocations).

4 . OBJECT ORIENTED TESTING TECHNIQUES

Object – Oriented programming is centered around concepts like Object, Class, Message, Interfaces, Inheritance, Polymorphism etc., Traditional testing techniques can be adopted in Object Oriented environment by using the following techniques:

- Method testing
- Class testing
- Interaction testing
- System testing
- Acceptance testing

4.1 Method Testing: Each individual method of the OO software has to be tested by the programmer. This testing ensures Statement Coverage to ensure that all statements have been traversed atleast once, Decision Coverage to ensure all conditional executions and Path Coverage to ensure the execution the true and false part of the loop.

4.2 Class Testing: Class testing is performed on the smallest testable unit in the encapsulated class. Each operation as part of a class hierarchy has to be tested because its class hierarchy defines its context of use. New methods, inherited methods and redefined methods within the class have to be tested. This testing is performed using the following approaches:

- Test each method (and constructor) within a class
- Test the state behavior (attributes) of the class between methods

Class testing is different from conventional testing in that Conventional testing focuses on input-process-output, whereas class testing focuses on each method.

Test cases should be designed so that they are explicitly associated with the class and/or method to be tested. The purpose of the test should be clearly stated. Each test case should contain:

- 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 for setup (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

Since object oriented software is rich in encapsulation, Inheritance and Polymorphism the following challenges are faced while performing class testing.

- It is difficult to obtain a snapshot of a class without building extra methods that display the classes' state.
- 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.
- Basis path, condition, data flow and loop tests can all apply to individual methods, but can't test interactions between methods

4.3 Integration Testing: Object Orientation does not have a hierarchical control structure so conventional top-down and bottom up integration tests have little meaning. Integration testing can be applied in three different incremental strategies:

- Thread-based testing, which integrates classes required to respond to one input or event.
- Use-based testing, which integrates classes required by one use case.
- Cluster testing, which integrates classes required to demonstrate one collaboration.

Integration testing is performed using the following methods:

- 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.

- For each message that is generated, determine the collaborating class and the corresponding method in the server object.
- For each method 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 methods that are invoked and add these into the test sequence.

4.4 System Testing: All rules and methods of traditional systems testing are also applicable to object-oriented systems. Various types of System Testing include:

- 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

4.5 Regression Testing: Regression testing is performed similar to traditional systems to make sure previous functionality still works after new functionality is added. Changing a class that has been tested implies that the unit tests should be rerun. Depending on what has changed, the test scenarios may have to be altered to support this test. In addition, the integration test should be redone for that suite of classes.

5. SPECIALIZED TECHNIQUES FOR OBJECT ORIENTED ENVIRONMENT

5.1 Fault – Based Testing: Any product must conform to Customer requirements. Hence, testing should begin with the analysis model itself to uncover errors. Fault – Based testing is the method used to design tests that have a high probability finding probable errors of the software[3]. Fault – Based testing should begin with the analysis and design models. This type of testing can be based on the specification (user's manuals, etc.) or the code. It works best when based on both.

5.2 Scenario – Based Testing: This new type of testing concentrates on what the customer does, not what the product does. It means capturing the tasks (use cases, if you will) the customer has to perform, then using them and their variants as tests. Of course, this design work is best done before you've implemented the product. It's really an offshoot of a careful attempt at "requirements elicitation". These scenarios will also tend to flush out interaction bugs. They are more complex and more realistic than fault based tests often are. They tend to exercise multiple subsystems in a single test, exactly because that's what users do. The tests won't find everything, but they will at least cover the higher visibility interaction bugs[4].

5.3 Surface Structure Testing: Object-oriented programming arguably encourages a different style of interface. Rather than performing functions, users may be given objects to fool around with in a direct manipulation kind of way. But whatever the interface, the tests are still based on user tasks. Capturing those will continue to involve understanding, watching, and talking with the representative user (and as many non representative users as are worth considering)[4] There will surely be some difference in detail. For example, in a conventional system with a "verbish" interface, one might use the list of all commands as a testing checklist. If he had no test scenarios that exercise a command, he perhaps missed some tasks (or the interface has useless commands). In a "nounish" interface, he might use the list of all objects as a testing checklist. A basic principle of testing is that we must trick our self into seeing the product in a new way. If the product has a direct manipulation interface, we'll test it better if we pretend functions are independent of objects. We'd ask questions like, "Might the user want to use this function - which applies only to the Scanner object - while working with the Printer?" Whatever the interface style, we should use both objects and functions as clues leading to overlooked tasks. [4]

5.4 Deep (architectural) structure: Test design based on the surface structure will miss things. User tasks will be overlooked. Important variants that should be tested won't be. Particular subsystem interactions won't be probed. Looking at the deep structure might reveal those oversights. Various constructs that can be tested using Deep testing are,

- A class diagram describes relationships between objects. An object of one class may use or contain an object of another class.
- The object diagram and interaction diagram give more detail about relationships between objects.
- A class diagram that shows inheritance structure.
- State charts (enhanced state machines) are a way of describing many tasks in a compact picture. If there is a state transition exercised by no test, why not? Again, the detailed state charts for objects buried deep in the system

are not likely to be as useful as those for the major, user-visible objects.

6. CONCLUSION

In Object Oriented environment the main troublemakers that cause problems for testing are Inheritance, Polymorphism and Encapsulation. We studied the problems that are created by these elements. A detailed study of the testing techniques available to test programs developed under OO environment have been made. Some of the specialized techniques available to test OO software have also been discussed.

REFERENCES

[1] A. J. J. Marciniak, "Encyclopedia of software engineering", Volume 2, New York, NY: Wiley, 1994, pp.1327-1358
[2] E. F. Miller, "Introduction to Software Testing Technology," *Tutorial: Software testing & Validation Techniques*, Second Edition, IEEE Catalog No. EHO 180-0, pp. 4-16
[3] Roger S.Pressman "Software Engineering – A Practitioner's Approach" McGraw Hill International Edition.
[4]http://www.exampler.com/testing-com/ ritings/2-scen.htm