International Journal of Advanced Computer and Mathematical Sciences
ISSN 2230-9624. Vol 3, Issue 4, 2012, pp546-549
http://bipublication.com

TESTING IN OBJECT ORIENTED ENVIRONMENT

Megha Vij and Deepti Bansal

IITT, Pojewal, Nawashahar, Punjab, India
megha.vij0001@gmail.com, deepti_bansi_1988@yahoo.co.in

[Received-02/10/2012, Accepted-29/11/2012]

ABSTRACT:
Software testing is an important software quality assurance activity to ensure that the benefits of Object oriented programming will be realized. Testing object oriented systems is little bit challenging as complexity shifted from functions and procedures as in traditional procedural systems to the interconnections among its components. Object oriented development has presented a numerous variety of new challenges due to its features like encapsulation, inheritance, polymorphism and dynamic binding. Earlier the faults used to be in the software units, whereas the problem now is primarily in the way in which we connect the software. A major challenge to the software developers remains how to reduce the cost while improving the quality of software testing. Software testing is difficult and expensive, and testing object oriented system is even more difficult. This paper highlights

Keywords: Encapsulation, Inheritance, Abstract Classes, Polymorphism.

1. INTRODUCTION

Software testing is an important software quality assurance activity to ensure that the benefits of object oriented programming will be realized. The objective of software testing is to uncover as many errors as possible with a minimum effort and cost. A successful test should show that a program contains bugs rather than showing that the program works. Since software testing consumes 40-50 percent of the development costs, how to reduce its cost and improve its quality has always been a big challenge. OO software testing has to deal with new problems introduced by the powerful features of OO languages. Features such as encapsulation, inheritance, polymorphism, and dynamic binding provide visible benefits in software development by increasing software reliability, reusability, extensibility and interoperability. Software testing is necessary to realize these benefits by uncovering as many programming faults as possible at a minimum cost. However OO features introduce new testing challenges that cannot be easily addressed with the previous techniques used in traditional approaches and require definitions of new techniques, new problems that require new solutions. Therefore requirements for testing object-oriented programs differ from those for testing conventional

programs. Most of the interactions that occur among program units take place through complex state interactions due to inheritance and polymorphism. Encapsulation makes it difficult to understand object interactions and prepare test cases to test such interactions. Inheritance does not guarantee that a method that is tested to be "correct" in the context of the super class will work "correctly" in the context of the sub class.

Polymorphism creates an attribute of an object may refer to more than one type of data, and an operation may have more than one implementation results in lack of controllability. Dynamic binding makes testing more difficult because the exact data type and implementation cannot be determined statically. Abstract classes cannot be instantiated and thus pose challenges for execution base testing.

2. Issues Involved in Object Oriented Testing

When obstacles to testing OO systems are discussed the problems that the OO specific features of inheritance, encapsulation and polymorphism create is usually the focus.

"The combination of polymorphism, inheritance and encapsulation are unique to object oriented languages, presenting opportunities for error that do not exist in conventional languages". Some other complexities of OO systems are also decentralized code, test case identification and raising exceptions.

a. *Decentralized Code:* With procedural systems where all the functions and procedures are centralized, locating the source of a bug does not require us to look outside the walls of the program. The same functionality that was in a single procedural program is broken out into many smaller classes in an OO system. It can be argued that smaller classes ease the process of locating the source of a bug because smaller chunks of code are examined. However, the side effect is that the code is no longer centralized so finding the source of a bug requires looking in many places.

b. *Encapsulation/Data Abstraction:* "Encapsulation is a technique for enforcing information hiding where the interface and implementation of a program unit are syntactically separated". This enables the programmer to hide design decisions within the implementation and to narrow the possible interdependencies with other components by means of interface. If a programmer changes only the implementation of unit leaving the interface same then he needs to retest that unit and any units that explicitly depend on it. Therefore if we modify the super class then it is necessary to retest all its subclasses because they depend on it in the sense that they inherit its methods. Data abstraction refers to the act of representing essential features without including the background details. Also due to data abstraction there is no visibility of the insight of objects. The data is not accessible to the outside world and only those functions which are wrapped in the class can access it. This data hiding makes it difficult for the tester to check what happens inside an object during testing.

c. *Inheritance:* Inheritance is one of the primary strengths of object-oriented programming. "Inheritance means properties defined for a class are inherited by its subclasses, unless it is otherwise stated". So actually it provides the idea of reusability. However method that is tested to be "correct" in the context of the base class does not guaranteed that it will work "correctly" in the context of the derived class. Therefore it is precisely because of inheritance that we find problems arising with respect to testing. It also complicates inter-class testing as multiple classes are coupled through inheritance.

d. *Polymorphism:* Polymorphism means the ability to assume more than one form, both in terms of data and operations. It is the capability of an operation exhibiting different behavior in different instances. However polymorphism results in lack of controllability as actual binding of object reference is not known till run time. In

program based testing as it can lead to messages sent to wrong object.

e. ***Abstract Classes****:* Abstract class is the way to push up common implementation into a base class. Hence adding new objects are easier, because a lot of the common interfaces may already be implemented. These classes are designed only to act as a base class. However, since their features are not fully implemented, these classes cannot be instantiated and thus pose challenges for execution base testing. Only classes derived from the abstract class can be easily tested, but errors can be present also in the super class i.e. abstract class.

3. Conventional Testing vs Object Oriented Testing

Conventional testing is the traditional approach to testing mostly done when water fall life cycle is used for development, while object oriented testing is used when object oriented analysis and design is used for developing enterprise software. Conventional testing focuses more on decomposition and functional approaches as opposed to object oriented testing, which uses composition. The three levels of testing (system, integration, unit) used in conventional testing is not clearly defined when it comes to object oriented testing. The main reason for this is that OO development uses incremental approach, while traditional development follows a sequential approach. In terms of unit testing, object oriented testing looks at much smaller units compared to conventional testing.

4. Model Based Testing For Object Oriented Systems

Model-based testing is a variant of testing that relies on explicit behavior models that encode the intended behavior of a system and possibly the behavior of its environment. Pairs of input and output of the model of the implementation are interpreted as test cases for this implementation: the output of the model is the expected output of the system under test (SUT).Tests can be derived from models in different ways. Because testing is usually experimental and based on heuristics, there is no known single best approach for test derivation. It is common to consolidate all test derivation related parameters into a package that is often known as "test requirements", "test purpose" or even "use case(s)". This package can contain information about those parts of a model that should be focused on, or the conditions for finishing testing (test stopping criteria).

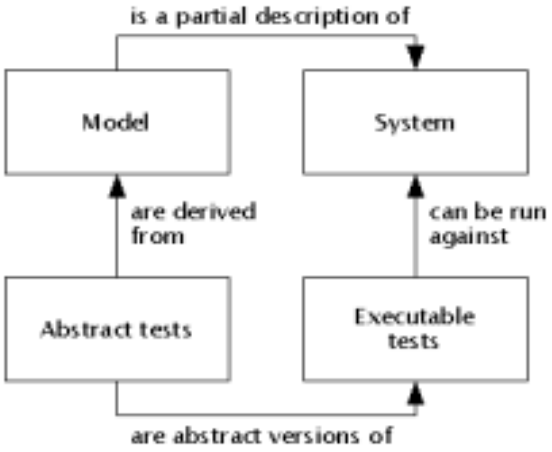

Figure 1: General model-based testing setting

Though, there are many modeling language such as UML, SDL, Z-Specification, we will discuss about use of UML as a test model, which is a semi formal modeling language. There are many phases in the testing process, including unit, function, system, regression, and solution testing. The following table illustrates the differences between these phases, as well as the potential UML diagram for use in the phase.

Test Type	Coverage Criteria	Fault Model	UML Diagram
Unit	code	correctness, error handling ,pre / post conditions, invariants	class and state diagrams
Function	Functional	Functional and API behavior, integration issues	Interaction and class Diagrams
System	Operational Scenarios	workload, contention, synchronization,	use case, activity, and

		recovery	interaction diagrams
Regression	Functional	Unexpected behavior from new / changed function	Interaction and class Diagrams
Solution	Inter System Communication	Interoperability Problems	use case and deployment diagrams

Table 1: Test Phases and UML Diagram

5. REFERENCES

[1] Mrs. Sujata Khatri , Dr. R. S. Chhillar, Mrs. Arti Sangwan. Analysis of factors Affecting Testing in Object Oriented Systems.
[2] Clay E. Williams. Software Testing and UML.
[3] W. Richards Adrion, Martha A. Branstad, and John C. Cherniavsky. Validation, verification, and testing of computer software. *ACM Comput. Surv.*, 14(2):159-192, 1982.
[4] D. Gelperin and B. Hetzel. The growth of software testing. *Commun. ACM*, 31(6):687{695, 1988.
[5] Tim Rentsch. Object oriented programming. *SIGPLAN Not.*, 17(9):51{57, 1982.