A REFLEXIVE ARCHITECTURE FOR OBJECT ORIENTED SOFTWARE TESTING

Mahamaya Mohanty*

Surbhi Agarwal**

Bhavyatta Bhardwaj***

ABSTRACT

This paper on "A Reflexive Architecture for Object Oriented Software Testing"presents a testing mechanism that facilitates creation of difficult-to-achieve states, for execution of state specific tests, during object-oriented software testing. Provisions are made in the software during the design phase and coded during the coding phase of software development. Testing object-oriented systems is more important than testing procedural software, as it promotes reuse. The object oriented software has various features like encapsulation, abstraction, polymorphism, inheritance, dynamic binding etc., which make the testing of object oriented programs difficult and different from the conventional testing methods. A lot of research has been done in the field of object oriented testing and various techniques have been developed for testing of object oriented programs. This paper discusses the different levels of testing object-oriented systems; the object oriented testing problems, various testing techniques and the future directions for testing object oriented systems. This paper presents a scenario-based object-oriented test framework (SOOTF) for adaptive and rapid testing.

Keywords: *Scenario-based Testing, Fault-based Testing, Unit Testing, Integration Testing, System Testing,object, class*

*Assistant professor, Department of Information Technology, Dronacharya College of Engineering, Greater Noida.

****Department of Information Technology, Dronacharya College of Engineering, Greater Noida.

******Department of Information Technology, Dronacharya College of Engineering, Greater Noida

1. INTRODUCTION

Software testing, as crucial part of quality assurance, should also focus on bug prevention. However, this ideal cannot be achieved, and there are always bugs whatever effort spent. Therefore, the pragmatic goal of software testing is to discover[1][2] bugs discover symptoms caused by bugs, and provide clear diagnoses so that bugs can be easily corrected. Bugs are not always obvious. Different bugs can have the same manifestations and one bug can have many symptoms. The symptoms and causes can be disentangled only by more tests, small and detailed tests. The development of a large software system is a time and resource-consuming activity. The insertion of OO technology in the software industry has created new challenges intricacy of inheritance, information hiding and encapsulation, as well as polymorphism, especially, making OO software testing much more difficult than the traditional software testing . Testing is typically the most time-consuming activity in the system development life cycle. The right set of test cases will drastically reduce the time required without decreasing the quality of the deliverable. To identify those test cases, business experts and technology specialists have to work together effectively. A right test case identification creates an environment of mutual understanding and common goals to get the task done quickly. The benefits of test case identification includes the following benefits:

- Optimize test coverage of critical functions
- . Establish the baseline for regression testing
- Reduce the number of errors going into production
- Reduce the time required to execute tests
- Increase reusability of test across related projects
- Increase the quality of the final deliverable
- Test the quality of business requirements before development

Doing testing means doing three things: designing tests, implementing the tests we have designed, and evaluating those tests. Test design is essential to effectiveness. Test implementation is essential to efficiency. Test evaluation improves effectiveness. More people are weak at design than at implementation, though that's not always obvious(because implementation problems are more immediate and pressing).We intend to concentrate on design. Software testing has two main aspects: test data generation and application of a test data adequacy criterion. A test data generation technique is an algorithm that generates test cases, whereas an adequacy criterion is a predicate that determines whether the testing process is finished. Several test data adequacy criteria have been proposed, such as control flow-based and data flow-based criteria. One of the major difficulties in software testing is the automatic generation of test data that satisfy a given adequacy criterion.

2. RELATED WORK

The application of imperative testing techniques has yielded some breakthroughs in object oriented testing (OOT) however, many researchers and practitioners emphasize the need for a uniform OOT process. Most of the work presents either testing theories or testing frameworks**.** Few testing techniques are presented. Testing theories are methodologies to handle testing. Testing frameworks involve a test plan or a test strategy used as an aid in transforming the testing theory into a testing technique. Imperative testing techniques are incorporated into OOT or concentrated on one feature of the object-oriented paradigm. In testing frameworks or theories are addressed. Overall, these efforts lack experimentation and robust features to test inheritance. The need for test plans is suggested but little support for actual test plan generation is included.

3. OBJECT ORIENTED TESTING TECHNIQUE

Two testing techniques as crucial: 1) object level testing (OLT) and 2) testing of inheritance within classes (TIC). Object testing is to the object oriented model what unit testing is to the imperative model. The unit testing technique plus other testing strategies such as path, control, branch, and statement testing are applicable in object testing. With the OLT, the tester tests every method in one object of a class. Interface testing involves the testing of the object's interfaces with other objects in the same class or in a different class. Object interfaces [6] include all of the object's methods that depend on the interaction with other methods in other objects or all of the object's methods whose execution is invoked by other methods in other objects. This test helps to eliminate further unneeded testing of the system at the cluster level, i.e., class integration, once errors are discovered at this level. Interface testing is also beneficial because errors that occur at this low level of code infrastructure cause other errors.

Typical input artifacts for object-oriented design are:

- Conceptual model: Conceptual model is the result of object-oriented analysis, it captures concepts in the problem domain. The conceptual model is explicitly chosen to be independent of implementation details, such as concurrency or data storage.
- Use case: Use case is a description of sequences of events that, taken together, lead to a system doing something useful. Each use case provides one or more scenarios that convey how the system should interact with the users called actors to achieve a

specific business goal or function. Use case actors may be end users or other systems. In many circumstances use cases are further elaborated into use case diagrams. Use case diagrams are used to identify the actor (users or other systems) and the processes they perform.

- System Sequence Diagram: System Sequence diagram (SSD) is a picture that shows, for a particular scenario of a use case, the events that external actors generate, their order, and possible inter-system events.
- User interface documentations (if applicable): Document that shows and describes the look and feel of the end i.e product's user interface. It is not mandatory to have this, but it helps to visualize the end product and therefore helps the designer.
- Relational data model: If an object database is not used, the relational data model should usually be created before the design, since the strategy chosen for object-relational mapping is an output of the OO design process. However, it is possible to develop the relational data model and the object oriented design artifacts in parallel, and the growth of an artifact can stimulate the refinement of other artifacts.

4. PROPOSED STEPS TO TEST OBJECT ORIENTED SOFTWARE

Step1: At first three distinct kinds of software testing performed.

Three distinct kinds of software testing:

Unit Test—testing performed on a single standalone module or unit of code. Before the developers submit the source codes to the release lab to construct a build, they should perform a unit test. Unit testing performed on individual code modules or programming subsystems checks for typographic, syntactic or logical errors, as well as determining whether or not the product matches its overall design as defined in the specification. QE may help development by aiding at the selection of test criteria for the unit testes. Every time Development submits code to the release Lab, they should apply a unit test.

Integration Test---testing performed on group of modules to ensure data and control is passed properly between modules. Integration Test is focus on Feature Test--the testing of individual functions in isolation from one another to evaluate the operation of the product against the product specifications. This section contains a definition of the major product features, including a list of outlines that will be created for use by the test team. Feature tests are written with the express purpose of stressing the product and thereby finding bugs.

System Test---testing concerns issues and behavior that can only be exposed by testing the entire integrated system or a major part of it. It tests the product as a whole to verify that the software meets the specified requirements, testing the product in "real world" applications. System testing builds on feature testing and has several steps—function testing, performance testing, acceptance testing and installation testing, that is to say, including testing for performance, security, accountability, configuration sensitivity, start-up and recovery, etc.

Step2: Secondly testing is performed through three different phases.

Three testing phases :

I .Primitive Test: the focus of testing is on the major features and common operations of the being testing programs or software systems.

II. Detailed Test: when the problems found in primitive test are fixed, there is no big blocking problems. Do more detailed, deeper testing to each functional areas of the system. Test features, performance, user interface, usability, compatibility, installation, stress test, etc. Quite a lot of bugs, various types and serious problems, could be found.

III. Regression Test: The purposes of Regression Testing is to verify that all features work and are properly integrated, all the bugs have been resolved, and the resolutions do not unacceptably affect other areas of the code. When the system becomes stable, few software problems can be found in a certain test cycle, then the software testing goes into Regression test phrase. Test focus is on re-testing all the SPRs to see whether there are any problems introduced by a fix to previous bug.

Final Regression Test: After completing feature testing and implement code freeze, the software enters the final Regression phase. During this phase, QE runs all or most of the tests again for signoff purposes. This cycle looks for bugs that may have been caused in areas affected by a fix. Once these tests are completed, the testing cycle for pre-release is finished.

Step3: Thirdly functional testing is perfomed on the software.

Functional testing:

Automated Test is a test performed by one application against another, enhancing a tester's ability of efficiently and repeatedly exercise to the product functionality. It must be great help to the software project that automatically repeating quite a lot of test to gain testing coverage within a short of time. The benefit of automation is that it can help to guarantee the objective testing over each build of a project.

Step 4: At the fourth stage fault-based testing is performed.

Fault-based Testing :

- incorrect specifications. The product doesn't do what the customer wants. It might do the wrong thing (the calculator implements base 2 logarithms, not the natural logarithms the target market expects) or it might not do some things (such as sensible handling of user input errors).

- interactions among subsystems. These are bugs where some action over there sets up some state that crashes this subsystem over here.

Fault-based testing, which is local in scope and driven more by the product than the customer, finds such problems only by chance. (There are ways to organize testing to increase that chance, but still not enough.) So another type of testing is needed.

Step 5: The next step uses Scenario-Based test design.

Scenario-Based Test Design:

This new type of testing concentrates on what the customer does, not what the product does[9].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.

Step 6: Objects of the entire system are found.

Objects:

Objects are the core of object-oriented systems[5][7].They represent the entities composing the system, whose interactions and characteristics define the behavior of the system itself. Intuitively, an object can be seen as an abstraction representing a real world entity or a conceptual element. An object is characterized by three properties:

(i)State : The state of an object is characterized by the values associated to its attributes. Attributes can be either primitive types or references to other objects. From a theoretical viewpoint, the state of an object should be modifiable and inspected only through the services it provides.

(ii)Services : They represent the functionalities provided by the object. Services are commonly called *methods*, and can be used for either altering or inspecting the state of the object they belong to.

(iii)Identity : This is an intrinsic property of objects,which assures that two different instantiations of a class are always distinguishable, even if their states are identical. Attributes and methods of an object are denoted as its *features*, *members*, or *properties*. Objects have an identity, but are nameless. They are referenced through special entities called *references*.

Here we use the term reference in a general way, without considering how such reference is actually implemented. An object is firstly created by both allocating the resources it needs and defining its
initial state. Then, the methods of the object are used to inspect and/or modify its state.

Step 7: Finally we put all the tested objects in a class.

Classes:

Object-oriented languages provide specific constructs which allow the programmer to define abstract data types and to classify set of object sharing both a common structure and the same behavior. As far as terminology is concerned, we use the term *class* to refer to such construct[10]. Classes can be considered as typed modular templates. The definition of a class implies the definition of a new type. It declares the number and type of attributes together with the provided operations.

5. CONCLUSION

Object-oriented paradigm provides the power for software development but at the same time introduces some brand new problems. These problems include the understanding problem, the complex interdependency problem and the object state behavior testing problem. Object state behavior implies that the effect of an operation on an object may depend on the states of some objects. In this paper we have proposed a decent technique and approach for testing software product. Recent research results indicate that certain object state behavior errors cannot be detected readily by conventional test methods. Therefore, new methodologies and tools are needed to support the testing of object state behavior.

REFERENCES

[1] Design patterns and object-oriented software testing Dasiewicz, P.; Electrical and Computer Engineering, 2005. Canadian Conference on 1-4 May 2005 Page(s):904 - 907

[2] Controllability mechanism for object-oriented software testing Anitha Goel; Gupta, S.C.; Wasan, S.K.; Software Engineering Conference, 2003. Tenth Asia-Pacific 2003 Page(s):98 - 107

[3] Object-oriented software testing-some research and development Kung, D.C.; Pei Hsia; Toyoshima, Y.; Chen, C.; Gao,J.; High-Assurance Systems Engineering Symposium, 1998. Proceedings. Third IEEE International 13-14 Nov. 1998 Page(s):158 - 165 ;

[4] An automatic approach to object-oriented software testing and metrics for C++ inheritance hierarchies Chun-Chia Wang; Shih, T.K.; Yule-Chyun Lin; Pai, W.C.;

Information, Communications and Signal Processing, 1997. ICICS., Proceedings of 1997 International Conference on 9-12 Sept. 1997 Page(s):934 - 938 vol.2

[5] Use object-oriented paradigm to design and implement an algorithm for object-oriented classlevel testing Yu Xia Sun; Huo Yan Chen; Systems, Man and Cybernetics, 2003. IEEE International Conference on Volume 2, 5-8 Oct. 2003 Page(s):1069 - 1074 vol.2

[6] BATOOM: a practical approach to testing object-oriented software Deng Yi; He Zhitao; Technology of Object-Oriented Languages, 1998. TOOLS 27. Proceedings 22-25 Sept. 1998 Page(s):328 – 337

[7] Test order for inter-class integration testing of object-oriented software Kuo-Chung Tai; Daniels, F.J.;Computer Software and Applications Conference, 1997. COMPSAC '97. Proceedings., The Twenty-First Annual International 13-15 Aug. 1997 Page(s):602 - 607

[8] Generating aspects-classes integration testing sequences a collaboration diagram based strategy Massicotte, P.; Badri, M.; Badri, L.;Software Engineering Research, Management and Applications, 2005. Third ACIS International Conference on 11-13 Aug. 2005 Page(s):30 - 37

[9] Scenario based integration testing for object oriented software development Youngchul Kim; Carlson, C.R.;Test Symposium, 1999. (ATS '99) Proceedings. Eighth Asian 16-18 Nov. 1999 Page(s):283 - 288

[10]A framework for specification-based class testing Ling Liu; Huaikou Miao; Xuede Zhan;Engineering of Complex Computer Systems, 2002. Proceedings. Eighth IEEE International Conference on 2-4 Dec. 2002 Page(s):153 – 162