Testing Object Oriented Software: Issues, State-of-the-art and Future

Chandra Mani Sharma
Department of IT
ITS Mohan Nagar
Ghaziabad

Rabins Porwal
Department of IT
ITS Mohan Nagar
New Delhi

Deepika Sharma
Department of IT
GBS Patparganj
New Delhi

ABSTRACT
The major chunk of the time spent in developing software is spent in testing software. It is evident that more than 50% of total time and efforts are consumed by testing phase. Still delivering 100% bug proof software to the client is not feasible. The reseachers in the field of software testing have been striving hard to provide better strategies for software testing with increased accuracy. In recent past the development of object oriented software has surpassed others. It poses a real challenge to test the object oriented software because of their specific nature and need of special parameters and criteria for testing. In the last few years a considerable amount of work has been carried out by various researchers in the field of object oriented software testing. This paper delves deep into the notion of testing object oriented software, reviews state-of-the-art in this field, discusses about the future directions and paves the path of any future research in object oriented software testing.

General Terms
Object Oriented Systems, Security, Reliability, Computers

Keywords
Software Testing, Test Data, Test Case Generation, Testing Concurrent Systems

1. INTRODUCTION
The fundamental objective of software testing is to spot bugs before they yield into malfunctioning of software resulting into minor or major loss. The history of software testing can be traced back to early 1950s when it used to be primarily *debugging oriented*. With the passage of time, orientation of software teating has changed to *demonstration oriented*, *destruction oriented*, *evaluation oriented*, and the modern software testing is *prevention oriented*. The classification of various software testing strategies based on box approach is very popular, under this a testing scheme can either be *white box* or it can be *black box*. In white box testing the code is visible to the tester and it is checked based on the various test cases that every piece of code executes correctely on a reasonable amount of test data. It is similar to testing the nodes in an electronic circuit. The white box testing is suitable at unit, integration and system levels. In black box testing a tester may not be known to the internal implementation of the software and checks the working of software based on some test data. Black box testing includes equivalence class partioning, boundary value analysis, exploratory testing, fuzz testing etc. There exists another concept of Grey box testing which lies somewhere between black box and white box testing. The examples of grey box testing include UML model and state model. In early 1990s the concept of object oriented software development started replacing procedural or function oriented approach. The main impetus behind this shift was the inbult reusability and flexibility of object oriented components. A class and similar constructs once developed could be used at different places without having to rewrite the code. From testing point of view it was believed that object oriented software will require lesser efforts than the procedural counterpart, but very soon contradictions to this notion started emerging. Perry and Kaiser [1] were among the first few ones to show that testing object oriented software was not an easy business and it may require upto even 3-4 times more efforts for testing object oriented software than testing a procedural one. Gradually the field drew the attention of researchers to explore complexities of testing object oriented software.

2. ISSUES IN TESTING OBJECT ORIENTED SOFTWARE
The testing of object oriented software should be different because of many intricacies listed below:

2.1 Problem due to Security in OO Software
In procedural paradigm there is no security of data and it is accessible without any restriction. On the contrary, the accessibility of data in object oriented paradigm is restricted and data in a class is not openly available. This feature of OO is known as *data hiding*. A class may have some public members and some members with restricted access. The data hold by an object at some point of time is known as its state and the behaviour of an object is defined by the functions they can call to change their state. From testing point of view, it is challenging to test all the states and beviours of different objects with restricted access.

```
class Test
{
        private:      int x;
                      int y;
        public: void setdata(int a, int b)
        {
                x=a;
                y=b;
        }
        void putdata()
        {
                cout<<a<<b;
        }
};
```

```
void main()
{
       Test ob1, ob2, ob3;
       ob1.getdata (12, 4);
       ob1.putdata ();
       ob2.getdata (122, 44);
       ob2.putdata ();
       ob1.getdata (1222, 444);
       ob1.putdata ();
}
```

In the above code snippet a class "Test" has been defined with two methods for assigning and accessing values to and from the data members. To test even the simple class scenarios holistic test cases need to be written and it makes wheel inside wheel to test the object oriented classes with limited accessibility.

2.2 Problem due to Association

Association is a horizonatal relationship among various classes. One class can be associated with another if its methods are called by other class and it can call methods of other class. In this way a class can call the methods of many classes and its methods can also be called by others too. It rims fare amount of complexity for testing. During the testing it is a real challenge to devise effective test cases that can test these ramifications of object (class) associations.

2.3 Problem due to Polymorphism

Object oriented code can exhibit two types of polymorphisms; *compile time* and *run time*. For a tester it is imperative to test different versions of operators and methods at compile time as well as at the time of program execution. Test cases can be designed with some effort to test compile time polymorphic scenarios but it is really a Herculian task to devise the test cases to test the run time polymorphic scenarios because in method overriding it is not so easy to predict that which function is to be bound with a funtion call. As this decision is deferred until some data is provided by user at runtime. It ramifies with increase in number of polymorphic constructs (method or operator).

2.4 Problem due to Inheritance

By virtue of Inheritance, the code in object oriented software development can be reused without rewriting it. It considerably reduces the software development time but not the testing efforts. At the time of inception of OO concecpt it was thought that software made using already testing reusable components will require no or very less testing effort. Contradictory to this fact is that it requires altogether new testing regime to test a reusable component (e.g. class) in every new context in wich it is being used. In inheritance and generalization where the code of an inherited class is virtually absent, still rigorous testing is required considering the new context of usage. Fig.1 depicts some of the types of class inheritances. These are single, multi-level and multiple inheritances. The latter two are more complex and ceate more challenging scenario from the point of view of testing.

2.5 Miscellaneous Problems

There are some other problems in testing OO software because of other characteristic features of OO paradigm. One such problem is because of *abstraction*. Abstraction refers to revealing what is important and hiding something which is not relevant or important. In many OO languages, abstraction can be implemented using interfaces and abstract classes. An abstract class provides generality in method usage. The definition of a method is omitted in abstract class and can be provided based on the specific requirement in subsequent inheriting classes. Abstraction is a very useful concept in OO systems but it makes the testing of OO software complex. Apart from abstraction, some modern OO languages provide advanced features such as concurrent code execution (multi-threading), networking, locale adaptability etc. These features are difficult to test for right operation in all possible scenarios. For testing concurrent execution and the possible problems such as deadlock & race conditions, meticulous design of test cases and data is imperative. Each of the core features of OO paradigm has an associated testing cost. Modern research suggests that software testing occupies a major pie of entire software development life cycle (almost 50%). Also testing OO software takes 3 to 4 times more effort than the procedural one. The main objective of any OO software testing approach should be to minimize the effort in testing OO software with *data hiding*, *inheritance*, *association* and *polymorphism* features. Each of these problems can be at various levels including class, integration and system level. Next section of this paper throws some light on state-of-the-art in testing OO software.

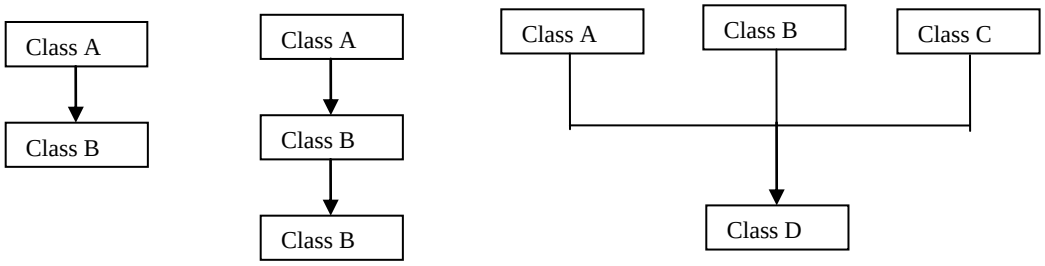

Fig 1: Some Inheritance Types: Single, Multi-level and Multiple Inheritances

3. STATE-OF-THE-ART IN TESTING OO SOFTWARE

In case of large size object-oriented test cases it could be impractical to run all possible test cases to test software. One solution to this problem is *random testing*. In [2], Ciupa et al. provide evidence that the relative number of faults detected by random testing over time is predictable. It shows that random testing finds faults quickly. First failure is likely to be triggered within 30 seconds. Evolutionary algorithms such as *genetic algorithms* have been widely used for procedural software testing. At some places genetic algorithm based approaches to OO software testing have been proposed. Gupta et al. use genetic algorithm to generate test cases for classes in object oriented software [3]. Here tree representation is used with test case statements. Gong et al. [4] proposed a technique

for class testing by using *Event-Driven Petri Nets* (EDPN). This EDPN based approach is sutable to model the changing states and behaviours of objects. The faults are detected by analyzing the differences of test scenario in the dynamic behaviors of EDPNs and the method can select a test case that detects errors described in the fault models. Ghang et al. [5] proposed a method using *relative reliability test* and *operation paths' reliability prediction* to adjust the test allocation of software reliability test in object-oriented programming. In their approach, software reliability test is based on the operational profiles and the relative reliability prediction results of operation paths. For this they used neural network learning algorithm. Chen et al. [6] propsed a technique for automated normal form generation for object oriented software testing. Their work improves the automation, coverage, and adequacy of selecting equivalent fundamental pairs as test cases. Concurrency in object-oriented is another feature which adds to the complexity of testing task. It is an important aspect of software testing to test the deadlock and raceconditions in concurrent software. *Design by Contract* (DbC) refers to that a method should define a contract stating the requirements a client needs to fulfill to use it, the precondition, and the postcondition. Araujo et al. [7] proposed a solution to challenges in of using DbC in concurrent OO software. It is a useful piece of work to investigate impact of contract assertions to detect race conditions, deadlocks and the other sytem functional properties. *Algebraic specifications* have also been used for OO software testing. As class-level testing using algebraic specifications includes two aspects. One is testing of equivalent ground terms and second is testing of nonequivalent ground terms. Chen et al. [8] showed that *given any canonical specification of a class with proper imports, a complete implementation satisfies all the observationally equivalent ground terms if and only if it satisfies all the observationally nonequivalent ground terms.* The findings provide a deeper understanding of software testing based on algebraic specifications, rendering the theory more elegant and complete. Goel at al. [9] discuss about *controllability mechnism* for which provisions are made as early as in design phase and coded in the coding phase of software development. They presented a controllability mechanism to facilitate the creation of difficult-to-achieve states, for execution of state specific tests during object-oriented software testing [9]. As the size of software is getting huge, it is difficult for testers to check out all parts of source code in white-box style during integration testing or system testing period. Black box testing techniques are widely used for testing OO software. Sea et al.[10] in their research paper compare five such black box testing techniques for OO software testing. A large number of test cases and a huge amount of data may be needed when it comes to test large scale enterprize software. Zheng et al.[11] proposesd a framework to integrate data analysis, simulation, and automated generation in OO software development. *SDGen*, an implementation of an automated software tool for the framework, has also been presented in [11]. Now-a-days *persistent objects* have become common place and many of the large object-oriented software systems rely on extensive amount of persistent data [12]. Recently Piccioni et al. [12] discussed the issue in context of ESCHER framework that addresses some issues through an IDE-integrated approach that handles class schema evolution by managing versions of the code and generating transformation functions automatically. Arcuri et al. [13] also discusses the problem of test data generation for testing OO software as automation of this task would therefore be highly desirable.

4. CONCLUSION & FUTURE WORK

This paper introduces testing of OO software. It emphasizes why testing of OO software is different, complex and even unreliable than testing of procedural software. The paper discusses various ramifications which make OO software testing an altogether different arena from testing of procedural software. Because of data hiding, inheritance, association and polymorphism, the testing of OO software is affected and is over headed by various cost and effort parameters. Further the paper discusses various state-of-the-art methods proposed for testing OO software. The approaches proposed for testing OO software are still not mature and there is a fare scope of refinement. For future work, we will use our theory to enhance testing OO software by proposing an optimized approach. In doing this, we will define a new criteria to test OO software at class level/ integration level or system level keeping in mind the various features of OO software and do the experimentation with classes of real life case studies along with the comparison of results with the already proposed approaches in order to evaluate its usability and the accuracy.

5. REFERENCES

[1] D. Perry, G. Kaiser, "Adequate Testing and Object–oriented Programming", Journal of OO-Programming, Vol. 2, No. 5, Jan. 1990, pp. 13-14

[2] I. Ciupa, Alexander Pretschner, Andreas Leitner, Manuel Oriol, Bertrand Meyer, "On the Predictability of Random Tests for Object-Oriented Software", in proc. International Conference on Software Testing, Verification, and Validation, 2008, pp. 72-81.

[3] N.K. Gupta, M.K. Rohil, "Using Genetic Algorithm for Unit Testing of Object Oriented Software", in proc. First International Conference on Emerging Trends in Engineering and Technology, 2008, pp. 308-313

[4] H. Gong, J. Li, "Generating Test Cases of Object-Oriented Software Based on EDPN and Its Mutant", in proc. 9th International Conference for Young Computer Scientists 2008, pp. 1112-1119

[5] W. Zhang, P. Tian, J. Liu, J. Li, "A Method of Software Reliability Test Based on Relative Reliability in Object-Oriented Programming", in proc. Asia-Pacific Conference on Information Processing, 2009, pp. 454-458

[6] H. Y. Chen, T. H. Tse, "Automatic Generation of Normal Forms for Testing Object-Oriented Software", in proc. Ninth International Conference on Quality Software, 2009, pp. 108-116

[7] W. Araujo, L. C. Briand, Y. Labiche "On the Effectiveness of Contracts as Test Oracles in the Detection and Diagnosis of Race Conditions and Deadlocks in Concurrent Object-Oriented Software", in proc. International Symposium on Empirical Software Engineering and Measurement, 2011, pp. 10-19.

[8] H. Y. Chen, T.H. Tse, "Equality to Equals and Unequals: A Revisit of the Equivalence and Nonequivalence Criteria in Class-Level Testing of Object-Oriented Software", IEEE Trans. on Software Engineering, 2013, pp. 1-15

[9] A. Goel, S. C. Gupta, S. K. Wasan, "Controllability Mechanism for Object-Oriented Software Testing", in proc. Tenth Asia-Pacific Software Engineering Conference (APSEC'03), 2003, pp. 98-107.

[10] K. I. Seo, E. M. Choi, "Comparison of Five Black-box Testing Methods for Object-Oriented Software" in proc. Fourth International Conference on Engineering, Research, Management and Applications, 2006, pp.213-220

[11] Y. Zheng, Y. Ma, J. Xue, "Automated Large-Scale Simulation Test-Data Generation for Object-Oriented Software Systems", in proc. First International Symposium on Data, Privacy and E-Commerce, 2007, pp. 74-79.

[12] M. Piccioni, M. Oriol, B. Meyer, "Class Schema Evolution for Persistent Object-Oriented Software: Model, Empirical Study, and Automated Support", IEEE Trans. on Software Engineering, 2013,vol. 39, no. 2, pp. 184-196.

[13] A. Arcuri and X. Yao, "On Test Data Generation of Object-Oriented Software", in proc. Academia and Industry Conference - Practice and Research Techniques, 2007, pp.72-76