Volume 2, No. 5, May 2011

Journal of Global Research in Computer Science

ISSN-2229-371X

RESEARCH PAPER

Available Online at www.jgrcs.info

DETECTING ANOMALIES DURING MULTIPLE INHERITANCE

Shubpreet Kaur[1 *], Shivani Goel[2]

[1] MTech Student, Thapar University, Patiala, Punjab, India
[1]shubpreetkaur@gmail.com,
[2] Assistant Professor, Department of Computer Science, Thapar University, Patiala, Punjab, India
[2]shivani@thapar.edu

Abstract: Object oriented software is developed with iterative and recursive increments. Object oriented software testing starts modules with unit testing in which each module is tested first then modules are integrated that forms integration testing and then they are collaborated to make a system which begins with system testing. In multiple inheritance, while collaborating various base classes to a derived class, there comes static and dynamic anomalies i.e. objects and the values assigned to the objects vary. So an approach is discussed to detect such anomalies. And one of the major challenges in software testing is the generation of test cases. Here we generating test cases firstly with unit testing approach, then integration and then with system testing approach. By testing in this way it improves the quality of software assure the high reliability of software. In this paper, our focus is on classes, objects, inheritance, method overriding, and polymorphism

Keywords: Generation of test cases, integration testing, Multiple inheritance, Object oriented testing, Software testing, Static and dynamic anomalies, system testing, unit testing,

INTRODUCTION

Object oriented system has concept of classes and objects. Object oriented systems have interaction among classes and objects. Classes are the containers of data members and member functions. Objects are the run time entities which are associated with various classes and they can call their data members and member functions with which they are associated. Objects keeps the track of various member function and data members and retain their value in object only.

Testing of classes can be done at three levels - unit testing, integration testing and system testing. Unit testing is testing each module of a class with the object created by the class itself. Integration testing is the merging of two units or classes. In object oriented systems, one of the simplest ways of integrating two classes is using inheritance in which the parent class is the base class and child class is inheriting the features of parent class known as derived class. Initially suppose we have two classes. So their will be two objects. But with integration, there is an increase in classes and objects which complicate testing and the chances of bugs and anomalies increase. Minimum number of objects gives better results with accuracy.

System testing comes after unit testing and integration testing. After merging units i.e. classes, we will get a bigger unit. System testing is testing the whole program. A number of classes are integrated together apart from inheritance to make a system. Calling various class objects with only correct functions is a challenge for system testing.

For all type of testing, the documentation part of the behavior of the program in different situations like desired and wrong function call or sequence call is called a test case. A test case tells in which cases the software will pass or fail. Pass means it will succeed giving the desired results. A failed test case means a failure in execution of that test path. The failure of a test case indicates that either it is an error or an anomaly.

LITERATURE SURVEY

Inheritance and polymorphism provide simplest ways of reuse in object oriented systems. Various issues and problems associated with testing polymorphic behavior of classes is discussed by Saini [11]. His approach is based on single inheritance. Various problems and issues on multilevel inheritance are reported by Alexander and Offutt [7]. Stroustrup, believed that multiple inheritance complicates a programming language significantly, is hard to implement, and is expensive to run [13].Christian focuses on object oriented systems with integration testing as object oriented

technology is too complex and it is a recursive and iterative process from analysis, design, implementation and testing view. There are many modules have interactions which increases the complexity and makes integration testing difficult and creates problem with generation of test cases[2]. Alexander describes the syntactic patterns for each object oriented fault type saying that the software contains an anomaly and possibly a fault [9]. Rountev present a general approach for adapting whole program class analysis to operate on program fragments in polymorphism [1]. Pressman suggests that integration of the system as a whole and then testing is a big bang approach in which all the units are taken together and tested together which is doomed to failure. So always use incremental integration i.e. collaborate two units and then apply integration testing. Fault based testing is discussed by him which identifies the method calls, objects which has high likelihood of uncovering plausible faults [6]. There have been some conflicts in ideas, concepts, and opinions among researchers regarding object oriented programming [12]. Robert V. Binder focuses on test case design - heuristic and formal techniques to develop test cases from object-oriented representations and implementations, testability - factors in controllability and observability [10].

In our earlier paper, an approach to identify anomalies in multilevel inheritance is discussed [3]. We are going to enhance this work on multiple inheritance in this paper. There are two approaches proposed here. First is based on detecting anomalies during and static and dynamic binding in multiple inheritance. Second is detecting anomalies during using unit, integration and system testing.

MULTIPLE INHERITANCE

Multiple inheritance is the ability of a class to have more than one base class (super class) as shown in Figure.1.

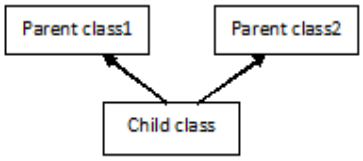

Figure. 1. Multiple inheritance

TYPES OF ANOMALIES

Static anomaly- when two or more base classes have same function and the derived class gets confused to which function it should call if required as shown in Figure.2..

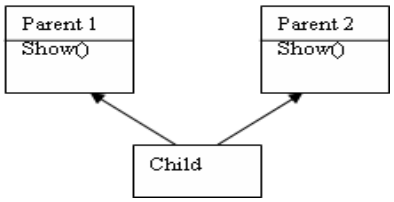

Figure.2. Static Anomaly in multiple inheritance

Dynamic anomaly-when we are getting wrong or garbage values at run time. In this case anomaly occurs with wrong calling of object if there exists same or similar function name in multiple classes. Figure.3. summarizes this all.

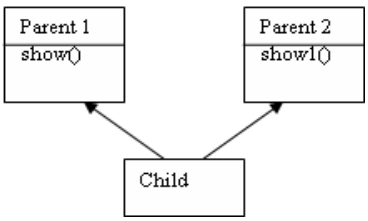

Figure.3. Dynamic anomaly in multiple inheritance

ALGORITHMS FOR ANOMALY DETECTION IN MULTIPLE INHERITANCES

An algorithm is designed for detecting the anomalies in multiple inheritance. For this various definitions used in the algorithm are discussed

Some Definitions

Type Family (TF): It is a set of classes that share a common behavior with respect to a base class A (we call it family (A)). Each descendent of A is a member B of A's family. If B is in A's family, polymorphism means that any instance of B may be freely used wherever an instance of A is expected. Every class A defines a type family, and that type family includes at least A [11].

Let A is a class in some type family.

Used by: A state variable v ε A is used by some method A if v is used in some expression in m.

Defined by: A state variable v ε A is defined by some method m ε A if m assigns first legal value to v.

Dependency of Methods: Two methods m,n ε A, we say that m is dependent on n if m uses a state variable v ε A which is defined by n.

Algorithm for static anomaly detection

1. **initialize** result=false
2. **for** every class ε TF **do**

3. **for** all methods m_i ε A and A is the child class or leaf node **do**
4. **if** there exists multiple parents for some p between 2 to n of child class **then**
5. **if** each parent class is publicly inherited **then**
6. **and** if method m_k of parent class is also defined by some other parent class ε TF and child is the descendent of parent but the child class does not contain the method m_k and there exists a call from child class object to method of m_i parent class **then**
7. result=true
8. **end** for
9. **end** for
10. **end** for
11. **if** there exists a call to method with the object specified with the class name **then**
12. **return** result

Explanation: Let's suppose initially there is no error in the program. So, for that we initialized result is false. Then we are checking for every class that belongs to the same type family starting from child class. To check for all the methods defined in the child class. Here child class is the leaf node. If there exists multiple parents (2 or more than 2) of that child class then there can be the chances of ambiguity if both the parents have the same method present in both the classes but not defined in the child class. If child class also has the same method name defined then there will be no ambiguity at all because the call will go only to the method present in the child class. Other case is to check whether the parent's classes are publically inherited or not. If the all parent classes are publically inherited then there will be the chance of an error. Suppose there are two parent classes of one child class. One class is publically inherited and other is privately inherited then in that case there will be no error but in that case there can occur the problem of dynamic anomaly that if inherited the parent class privately and we are using its data members in the child class then instead of inheriting the original value the garbage value will be taken. Now we are checking for the case of ambiguity. For this firstly we check for the existence of multiple parents and if those are publicly inherited. Then checking for the methods if they are present in all the parent classes but not in the child class. And child is the descendent of parent and we are calling the method of parent class with the object of child class. Then we can say there exists the static ambiguity which is a compile time error. And the result changes from false to true that there is an ambiguity. If the method is called with the specified object with the class name and method then there will be no ambiguity if same method name is defined in the other parent classes.

Algorithm for dynamic anomaly detection

1. **initialize** result=false
2. **for** every class ε TF **do**
3. **for** all methods m_i ε A and A is the child class or leaf node **do**
4. **if** there exists multiple parents for some p between 1 to n of child class **then**
5. **for** parent class is inherited for some p between 1 to n **do**
6. **for** each state variable $\mathbf{v_j}$ ε parent used by m_i **do**
7. **if** v_j is defined by m_k ε parent for some k between 1 to m
8. **and** if m_k is overridden by child class ε TF and child is the descendent of parent **then**
9. **if** there exists a call with child class object from m_i to m_k of parent **then**
10. result=true
11. **end** for
12. **end** for
13. **end** for
14. **if** there exists a call to method with the object specifying the class name **then**
15. result =false
16. **return** result

Explanation: Here also suppose initially there is no error in the program. So, for that we initialized result is false. Then we are checking for every class that belongs to the same type family starting from child class. To check for all the methods defined in the child class. Here child class is the leaf node. If there exists multiple parents (2 or more than 2) of that child class then there can be the chances of anomaly that whether the parent class is publicly inherited or not. Anomaly exists because of the wrong object call is given. So for that we are checking the state variable that belongs to child class methods. So for this we are checking the state variable used by the parent and the variable v_j is defined by method m_k of parent class and this method is overridden by child class and there exists a call with child class object from m_i to m_k of parent class. And the result returns true i.e. there exists data flow anomaly. And if there exists a call to method with the object specifying the class name there would be no data flow anomaly. And the result is false.

VERIFICATION OF THE PROPOSED ALGORITHMS USING A CASE STUDY

Consider an example of multiple inheritance: There are two base classes one is Library and other is Canteen and there is one derived class which is inheriting both these class as depicting in Figure.4. Class Library has object l, class Canteen has object c and class Student has object s.

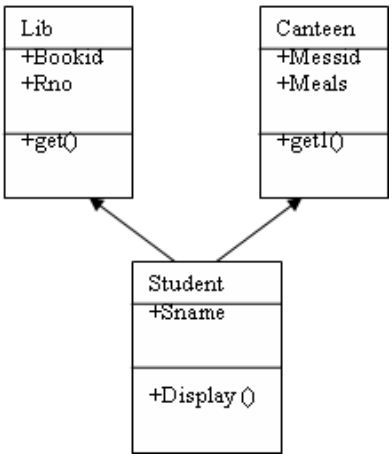

Figure.4. Case study of Multiple inheritance

The correct sequence call of methods is indicated in Figure 5.

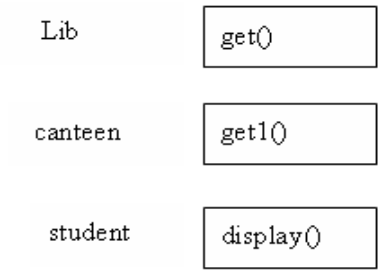

Figure. 5 Correct Sequence call of methods

Unit Testing First we will start with unit testing. At unit testing we will test each class with its object itself.

In unit testing, all the classes are tested with their own objects. All classes are getting pass status because we are initializing them and displaying them in the same class. The case of display() function in student class (which is derived of the get and get1 base classes). It is displaying the correct result of sname (i.e. student name) and other parameters as garbage because will be inheriting their features in the later part.

Test case ID	Sequence call	Expected value	Actual value	Status
1	1.get()	bid=1,rno=22	bid=1,rno=22,	pass
2	c.get1()	messid=20,meals=bread	messid=20,meals=bread	pass
3	s.dispaly()	Rno=gv,sname=anu,messid=gv,bid=gv	Rno=gv,sname=anu,messid=gv,bid=gv	pass

Table 1: Test cases of Unit testing in Multiple Inheritance

Integration Testing : Here we are testing base classes with derived classes and objects of both.

In this only two cases are passing out of six where we are calling them in correct sequence. Other are the incorrect sequence calls which will give garbage value. Example we are calling display() function first and get() function later then it gives a garbage value. But in case if we are handling large project in that case detecting such a mistake can be the difficult task.

Test case ID	Sequence call	Expected value	Actual value	Status
1	1.get()-> s.display()	bid=1,rno=22,rno=22,sname=anu,messid=gv,bid=1	bid=1,rno=22,rno=gv,sname=gv,messid=gv,bid=gv	fail
2	c.get1()-> s.display()	messid=20,meals=bread,sname=anu,rno=22,messid=gv,bid=1	messid=20,meals=bread,rno=22,sname=anu,messid=gv,bid=1	fail
3	s.display()->1.get()	bid=1,rno=22,bid=1rno=22,sname=anu,messid=gv,bid=1	bid=1,rno=22,bid=1,rno=gv,sname=gv,messid=gv,bid=gv	fail
4	s.display()-> c.get1	rno=22,sname=anu,messid=gv,bid=5,messid=20,meals=bread	rno=gv,sname=gv,messid=gv,bid=gv,messid=20,meals=bread	fail
5	s.get()-> s.display()	bid=1,rno=22,rno=22,sname=anu,messid=gv,bid=5	bid=1,rno=22,rno=22,sname=anu,messid=gv,bid=1	pass
6	s.get1()-> s.display()	messid=20,meals=bread,sname=anu,rno=22,messid=20,bid=1	messid=20,meals=bread,sname=anu,rno=22,messid=20,bid=1	pass

Volume 2, No. 5, May 2011

Journal of Global Research in Computer Science

ISSN-2229-371X

RESEARCH PAPER

Available Online at www.jgrcs.info

Table 2: Test cases of Integration testing in Multiple Inheritance

System Testing **:** Here all the classes are tested with all the objects in all possible sequence.

In system testing also, we are getting only two test cases having pass status. Others are displaying garbage value because of wrong function call.

Test case ID	Sequence call	Expected value	Actual value	Status
1	s.get()->s.get1()->s.display()	bid=1,rno=22,bid=1,rno=22,sname=anu,messid=4,meals=bread,bid=1,bid=1,rno=22,sname=anu,messid=4,meals=bread	bid=1,rno=22,bid=1,rno=22,sname=anu,messid=4,meals=bread,bid=1,bid=1,rno=22,sname=anu,messid=4,meals=bread	pass
2	s.get1()->s.get()->s.display()	messid=4,meals=bread,bid=1,rno=22,bid=1,bid=1,rno=22,bid=1,rno=22,sname=anu,messid=4,bid=1	bid=1,rno=22,bid=1,rno=22,sname=anu,messid=4,meals=bread,bid=1,bid=1,rno=22,sname=anu,messid=4,meals=bread	pass
3	s.display()->s.get()->s.get1()	messid=4,meals=bread,bid=1,rno=22,bid=5,bid=1,rno=22,bid=5,rno=22,sname=anu,messid=4,bid=1	bid=1,rno=22,bid=1,rno=22,sname=anu,messid=4,meals=bread,bid=1,bid=1,rno=22,sname=anu,messid=4,meals=bread	fail
4	s.display()->s.get1()->s.get()	messid=4,meals=bread,bid=1,rno=22,bid=1,bid=1,rno=22,bid=5,rno=22,sname=anu,messid=4,bid=1	bid=1,rno=22,bid=1,rno=22,sname=anu,messid=4,meals=bread,bid=1,bid=1,rno=22,sname=anu,messid=4,meals=bread	fail
5	s.get()->s.display()->s.get1()	messid=4,meals=bread,bid=1,rno=22,bid=1,bid=1,rno=22,bid=5,rno=22,sname=anu,messid=4,bid=1	bid=1,rno=22,bid=1,rno=22,sname=anu,messid=4,meals=bread,bid=1,bid=1,rno=22,sname=anu,messid=4,meals=bread	fail
6	s.get1()->s.display()->s.get()	messid=4,meals=bread,bid=1,rno=22,bid=1,bid=1,rno=22,bid=1,rno=22,sname=anu,messid=4,bid=1	bid=1,rno=22,bid=1,rno=22,sname=anu,messid=4,meals=bread,bid=1,bid=1,rno=22,sname=anu,messid=4,meals=bread	fail

Table 3: Test cases of System testing in Multiple Inheritance

HERE WE ARE DISCUSSING COMMON ANOMALIES IDENTIFIED FROM ABOVE TEST CASES:

- An error comes in integration testing that we are calling an object with base class object and then using other function with derived class object. Example is shown in Figure 6.

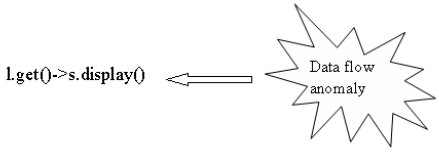

Figure.6.

Data flow anomaly 1

- Figure. 7 shows a case in which base class does not calls a derived class but a derived class can always call a base class.

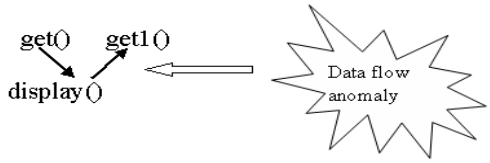

Figure. 7 Data flow anomaly 2

- When a base class calls other base class in multiple inheritance , an anomaly occurs (Figure 8).

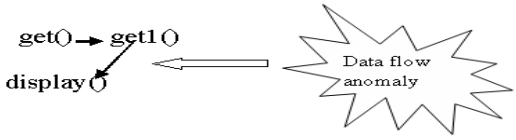

Figure.8. Data flow anomaly 3

- No anomaly: When both the base classes are inherited in derived class and they are calling with the required object or with the derived class object as stated in Figure 9.

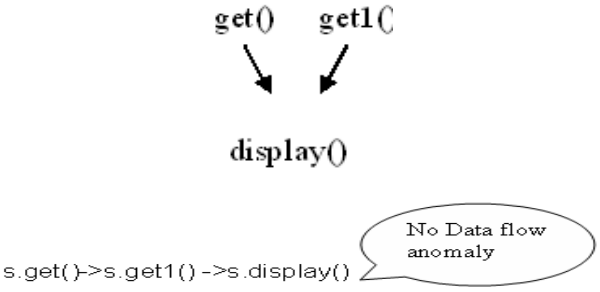

Figure.9. No Data flow anomaly

The above cases indicates that correct sequence call is essential for successful inheritance. A model of Do's and Don'ts in multiple inheritance is proposed here.

Do's in Multiple Inheritance	Don'ts in Multiple Inheritance
-Try to reduce the number of objects -Call Base class function with Base class object -Check the calling object to which class it is associated	-Don't call Base Class 1 function with Base class 2 object -Don't call Derived class function with Base class object -Don't keep the same function name in more than 1 Base class

CONCLUSION

In this paper, static and dynamic anomalies of multiple inheritance are discussed. Algorithms are presented with which can detect static and dynamic anomalies. They are verified using unit, integration and system testing. So to make system error free, unit, integration and system testing is applied. These algorithms trap all the static and dynamic errors. The test cases are shown which tells testing done in unit, integration and system testing that in which cases the code will fail and pass.

REFERENCES

[1] Atanas Rountev, Ana Milanova, Barbara G. Ryder, Fragment Class Analysis for Testing of Polymorphism in Java Software, IEEE Transaction, June 2004, vol. 30, no.6, pp.372-387

[2] Bucanac. Christian, "Object oriented testing report",Software verification and validation, Version. 0.2, December 1998, pp. 3-7.

[3] Kaur. Shubpreet,Goel. Shivani,"Testing Anomalies in Multiple and Multilevel Inheritance", International Journal of Computers and Communications, May 2011.

[4] McGregor D., John, A practical guide to testing object oriented systems

[5] Meyer, B., Object-Oriented Software Construction. Prentice Hall, second ed., Apr. 1997.

[6] Pressman, R.S. Software Engineering: A Practitioner's Approach. Mc Graw Hill, second ed., 2005.

[7] R. Alexander and J. Offutt.," Criteria for Testing Polymorphic Relationships", In Proceedings of the 11th international Symposium on Software Reliability Engineering (Issre'00) (October 08 - 11, 2000). ISSRE. IEEE Computer Society, Washington, DC, pp.15-23.

[8] R. Binder, Testing object-oriented software: a survey, Journal of Software Testing, verification and Reliability, 1996, Vol 6, pp.125–252.

[9] R. T. Alexander, J. Offutt, and J. M. Bieman, "Syntactic Fault Patterns in OO Programs", Proceedings of the 8th International Conference on Engineering of Complex Computer Software (ICECCS '02), Greenbelt, MD, November 2002.

[10] R. V. Binder, Testing Object-Oriented Systems Models, Patterns, and Tools, Addison-Wesley, NY USA, 1999.

[11] Saini D.K, Testing Polymorphism in Object Oriented Systems for Improving software Quality, - ACM SIGSOFT Software Engineering Notes, 2009.

[12] S Supavita , Object-Oriented Software and UML-Based Testing: A Survey Report, 2009.

[13] Stroustrup, B., Multiple Inheritance for C++, Published in the May 1999 issue of "The C/C++ Users Journal".