CHAPTER 2

INHERITANCE RELATED BUGS TESTING MODEL

2.1 INTRODUCTION

In this thesis, a model is proposed after analyzing the pitfalls in object oriented programs. The proposed innovative model is used for testing inheritance related bugs in object oriented programs and is named as “Inheritance Related Bugs Testing” (IRBT) model. These bugs will not be found by traditional object oriented language compilers or editors. The developer can use the proposed model for analyzing whether the developed object oriented program is ready for customer use and software maintenance.

2.2 INHERITANCE RELATED BUGS TESTING MODEL

Data flow testing will be very useful to analyze Object Oriented Programs. The Inheritance Related Bugs Testing (IRBT)” model structure is illustrated in Figure 2.1. It consists of the following four major modules:

1. Source code analysis of object oriented programs
2. Whole program analysis and fragment analysis testing
3. Analysis of inheritance property of object oriented program
4. Testing inheritance related bugs in object oriented programs

Before testing any program in detail for the existence of bugs, it is essential to extract preliminary information about the program to be tested. The existence of polymorphism can also be identified by module 1.
Figure 2.1 Inheritances Related Bugs Testing Model (IRBT Model)
The second module is Whole Program Analysis in which the program is analyzed for number of classes, methods, attributes, comment lines and objects. By using this module, analyzing pieces of code called **Fragment Analysis** is done. For example, Class is a fragment in Java program. The output is, name of the class, the methods and objects in the class as well as lines of code. This output can be used for finding the software metrics.

In third module, analyzing the OOP for complete information like classes, methods in a class, attributes in a method and levels of hierarchies is carried out. It identifies inheritance levels which will be useful to find level related bugs.

The final module in the model identifies implementation inheritance related bugs namely Naked Access, Inadvertent Binding, Name Conflicts, Spaghetti Bug and Naughty Children.

**Naked Access**: Super class instance variables are visible in a subclass and subclass methods update these variables directly. This leads to a bug known as Naked Access. **Inadvertent binding**: If a base class data member and derived class data member have the same name, it makes confusion on usage of the data members known as inadvertent binding. **Name conflicts**: The class extends itself, is called name conflicts in inheritance relationship. **Spaghetti bug**: If the inheritance level exceeds more than six (based on the experiences gained by the different development team) there is a possibility of spaghetti bug, it leads to complexity in further extension. **Naughty children**: A subclass which does not accept all messages that the super class accepts is known as Naughty children.

The proposed IRBT model can be used effectively for Object Oriented Programs written using the same language but having different
structures. Hence, the presented model permits the developer to give different Object Oriented Programs which have different structures in different modules as input. The idea behind providing such flexibility is to help the developer to perform their required testing. This idea is further explained below with examples

**Example 1:** If the developer requires parameters to find out the software metrics alone as output, then only module 2 can be used separately.

**Example 2:** If the OOP needs to be tested for an inheritance levels only, then usage of module 3 alone is sufficient.

This thesis discusses the usage of the IRBT model in the ways mentioned above.

### 2.3 SOURCE CODE ANALYSIS OF OBJECT ORIENTED PROGRAMS

Source code analysis gives information about the source code written using object oriented language. Source code is completely analyzed. Most existing program analysis provides syntactic structure of the program. With the data flow analysis, semantic structure analysis is possible. Preliminary information is needed for testing. In this thesis object oriented program source code is analyzed. This is implemented by Source Code Analyzer, a static tool useful for analyzing an Object Oriented Programs. Input is an Object Oriented Program and the outputs are number of lines, number of semicolons, number of function braces and data type used.
2.4 WHOLE PROGRAM ANALYSIS AND FRAGMENT ANALYSIS TESTING

Whole program analysis and fragment analysis testing will be useful to find out the object oriented property of a program and software metrics parameters. Whole program is analyzed for classes, methods, objects and it is used for further analysis. Fragment is the one which divides a program into small pieces. Fragment analysis information are class name, methods, lines of code and objects in a particular fragment. By using specific program, metric parameter can be identified. In this work whole Java program is the input and every line of codes is analyzed by using compiler techniques. Outputs are classes, methods in a class, attributes in a method and object. In fragment analysis, a program is divided into pieces. For example in Java language, it is divided into classes. Lines of code, attributes and methods are found in a class. All these information are stored on a knowledge base. Knowledge base stores information about program analysis and it can be retrieved for further usage.

2.5 ANALYSIS OF INHERITANCE PROPERTY OF OBJECT ORIENTED PROGRAMS

Object oriented software development is different from traditional development products. In object oriented software polymorphism, inheritance, dynamic binding are the important features and inheritance property is the main feature. Data flow testing is an appropriate testing method. This testing analyzes the structure of the software and gives the flow of property. This work is designed to detect the set of classes and packages and output is hierarchies of the classes, methods, attributes detected by using data flow testing. Class hierarchy analysis is used to find relationship between the classes. This testing is performed by three major analysis such as knowledge base preparation, code analysis and level analysis. The code analysis is
designed to extract details from the code. The knowledge base preparation is designed to prepare the knowledge base about the program details. The level analysis is designed to extract bugs related information from the database and it focuses mainly on Java programs. It is a static testing tool. Input of the tool is OOP. Output of the tool is attribute description in classes, methods description, classes with levels list. It shows the hierarchy of the classes. Inheritance property analysis is carried out for analyzing Sunsoft Java program. In addition it has got facility of analyzing third party programs also.

2.6  TESTING FOR INHERITANCE RELATED BUGS IN OOP

The compilers usually detect the syntax oriented errors. Some of the property bugs are located in the program. This work is designed to detect the bugs with reference to the inheritance property. Implementation phase inheritance related bugs such as Naked Access, Naughty Children, Inadvertent Binding, Name Conflicts and Spaghetti Inheritance bugs are detected by this module. The tool is developed as four major modules. The testing is conducted using the four modules viz. Function _model, Class _model, Variable_ model and Interface_ model.

2.7  CONCLUSION

Testing is an important phase of software development. Methods of testing are available in the literature for procedure oriented programs. So far no method has been developed for testing an object oriented program. An innovative model IRBT has been developed for the first time to test inheritance related bugs in object oriented programming. IRBT model is useful not only for testing Naked Access, Inadvertent Binding, Naughty Children, Name Conflicts and Spaghetti Bug of an Object Oriented Program but also for testing and analysis of source code and inheritance property analysis.