Creation of a Test Bed Environment for Core Java Applications using White Box Testing Approaches

Priya R. L[1], Dhanamma Jagli[2]

[1]Assistant professor, Department of Computer Engineering, V. E. S. Institute of Technology, Mumbai, India.
priyas_24@rediffmail.com
[2]Assistant professor, Department of MCA,
V. E. S. Institute of Technology, Mumbai, India.
dhana1210@yahoo.com

ABSTRACT

A Test Bed Environment allows for rigorous, transparent, and replicable testing of scientific theories. However, in software development these test beds can be specified hardware and software environment for the application under test. Though the existing open source test bed environments in Integrated Development Environment (IDE)s are capable of supporting the development of Java application types, test reports are generated by third party developers. They do not enhance the utility and the performance of the system constructed. Our proposed system, we have created a customized test bed environment for core java application programs used to generate the test case report using generated control flow graph. This can be obtained by developing a new mini compiler with additional features.

KEYWORDS

Test Bed, Token Parser, Flow graph analyzer, Watch variable Handler, Path Synthesizer, Plan synthesizer, Test case generator.

1. INTRODUCTION

Software testing is a process of inspecting the performance of software. As a part of any software development process, software testing represents an opportunity to deliver quality software and substantially reduce development cost. It is a great challenge to design test bed environments to achieve effective testing whereby most of the defects in the system are revealed by utilizing a limited quantity of resources.

The existing systems such as Net Beans and Eclipse have their own text editor for writing java programs and generating test cases for the same. But the drawback is that they cannot read as an input any other java source file written in any of the text editors like notepad, command prompt. Our system overcomes NetBeans and Eclipse drawbacks by building up a test bed environment that accepts java source file written in any text editor, check for the syntax, structure of the same and hence generate the test case reports for the core java program.

The rest of the paper is organized as follows; Section 2 presents about the proposed system. The architecture and implementation details are described in sections 3 and 4 respectively. In section 5, presents the conclusion. Section 6 we outline further enhancement in the area of test bed environment.

2. PROPOSED SYSTEM

System Strategy

We proposed to browse the *.java file as an input file. This .java file is then passed to the token parser, flow graph analyzer and watch variable handler. *.xml file is generated by them which are then passed to path synthesizer and plan synthesizer. Control flow is then passed to test case generator engine generating test case reports and modified *.java files.

White box Testing

White box testing strategy deals with the internal logic and structure of the code is also called as glass, structural, open box or clears box testing. The tests written based on the white box testing strategy incorporate coverage of the code written, branches, paths, statements and internal logic of the code etc. This testing needs the tester to look into the code and find out which unit or statement or chunk of the code is malfunctioning.

Control flow Graph

Control flow (or alternatively, flow of control) refers to the order in which the individual statements, instructions, or function calls of an imperative or a declarative program are executed or evaluated.

3. SYSTEM ARCHITECTURE

Figure below gives an architectural view of the system. .java file is been browsed for which the test cases are to be generated. This .java file is then passed to the token parser. Tokens generated here are then passed to the flow graph analyzer and watch variable handler. .xml files generated by them are then passed to path synthesizer and plan synthesizer. Control flow is then passed to test case generator engine. Test case engine then generates the test case reports and .java files.

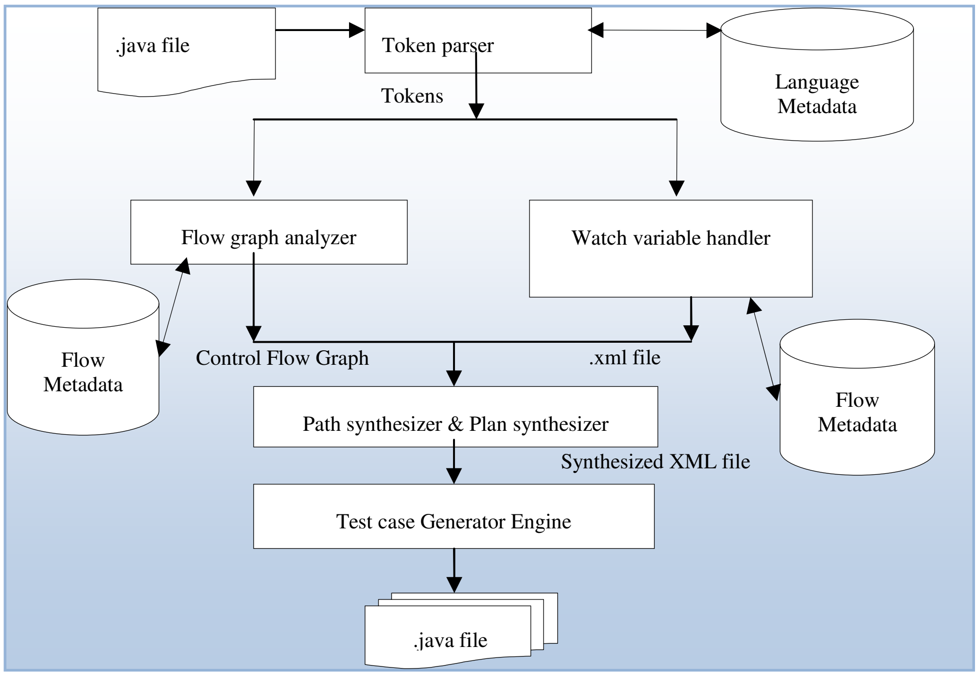


Figure 1.System Architecture

3.1 SYSTEM IMPLEMENTATION

A. Browsing .java files

A java file for which the test cases are to be generated is written in any text editor and is compiled. This java program is then browsed by the system. The system is not concerned about the editor file in which the program is written.

B. Token parser

The token parser reads in a sequence of characters and produces a sequence of objects called "Tokens". The rule used to break the sequence of characters into the sequence of tokens depends on the language. The parser refers the language repository which stores the tokens for java language. The tokens generated by the token parser are then passed on to the flow analyzer and watch variable handler.

C. Flow graph analyzer

Flow graph analyzer uses the token produced by the token parser and analyzes the control flow of the program using control table, functions and 'JGraphX' Application Program Interface (API). Control table gives the syntax of the controls and function gives the flow of the function along with the proper entry and exit conditions.

D. Watch variable handler

Watch variable handler will take up the tokens produced by the token parser and will give the changes in the values of the variable occurring in a particular loop or class or global variable and also identifies the type of variable. It will also generate the .xml file for the same.

E. Path synthesizer and Plan synthesizer

The .xml files being received from flow graph analyzer and watch variable handler are combined by the path synthesizer and plan synthesizer to generate another .xml file containing the flow analysis along with the changes in variable value and variable type. This .xml file is given to test case generator engine.

F. Test case generator engine

The syntax analyzer passes the result to the Test case generator engine. The test case generator engine makes the graphical visualization of the result obtained from syntax analyzer and then performs cyclomatic testing. The test report is then been generated.

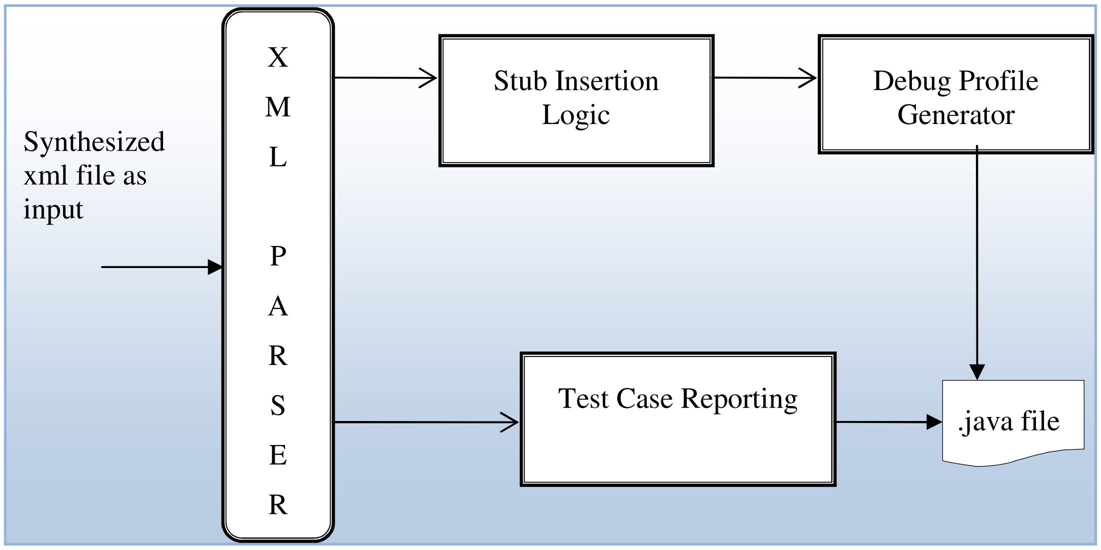


Figure 2.Test case Generator Engine

i. XML Parser

The Synthesized XML file obtained from the path synthesizer and plan synthesizer is given to XML parser. SAXP is one of Java XML programming API, provides capability of validating and parsing XML documents. Finally, it creates Java objects and manipulates them.

ii. Stub Insertion Logic

Stubs are classes that provide replacement implementations for the generated java objects by the xml parser. This stub is a replacement class is given as input to debug profile generator.

iii. Debug Profile Generation

Debug profile generator gives a modified .java file to the user with some additional information related to stub output. This helps to detect errors and causes of errors.

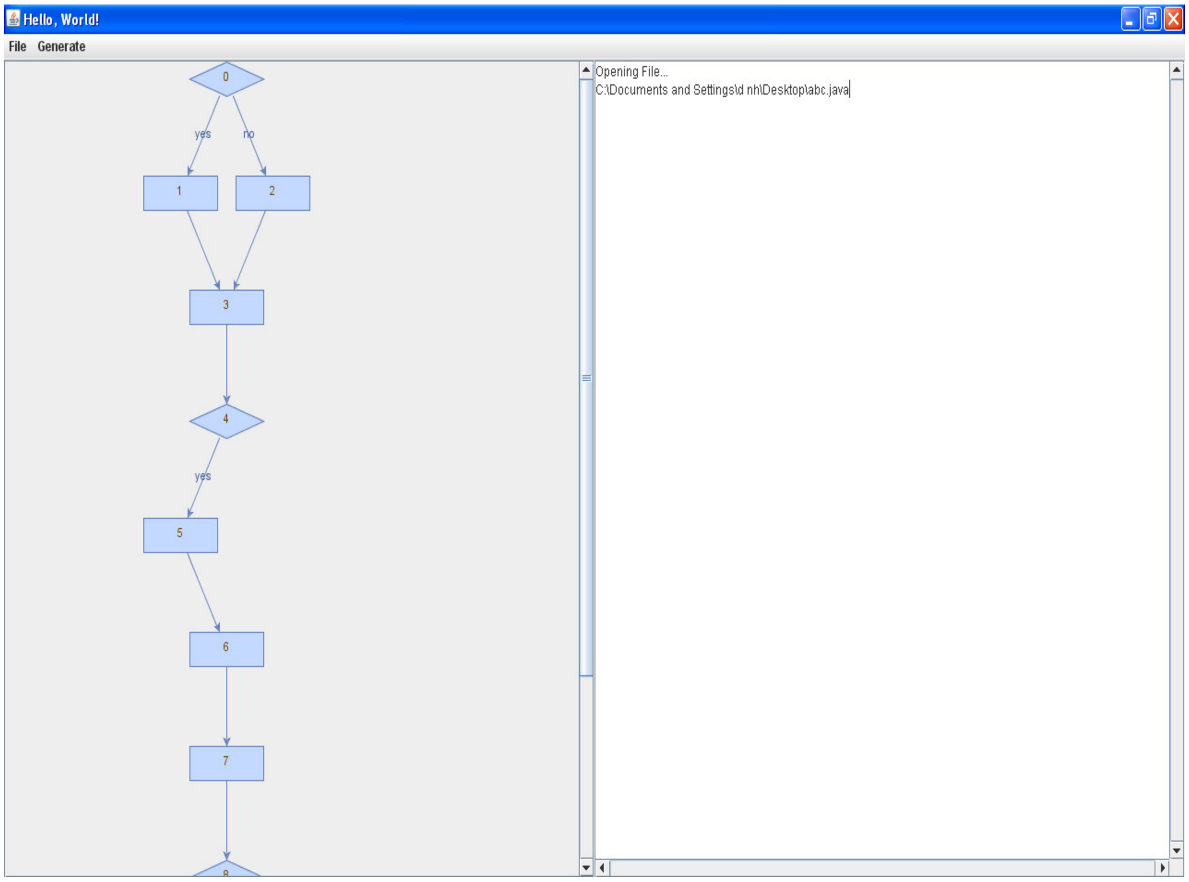
iv.Test Case Reporting

Test case reporting takes the input from xml parser to directly generate the test reports for the user or tester. Report can also identify any remaining deficiencies, limitations or constraints that were detected by the testing performed.

4. RELATED WORK

We have been successful in building up an IDE or an application that accepts java source file written in any text editor and we check for the syntax and structure of the same. Thus, our application can now generate the control flow graph for the java programs as shown in the figure 3.

The table given below illustrates the comparison between automated testing tools available and that generated by our system.



Opening File...
C:\Documents and Settings\d nh\Desktop\abc.java

Comparison

	Advantages	Disadvantages
Netbeans	It has its own text editor, rich IDE support for J2EE application. Multi browser support	It requires plug-ins to support other server-side languages such as ASP, PHP etc.
Eclipse	Multi browser & different language support. Has its own text editor and very user- friendly.	Plug-ins are used to generate test reports, which cannot accept the input file from other text editors like command prompt, notepad etc.
Sahi	Multi browser support - Has its own IDE - Record and playback tests	Confusing interface - Least developed/smallest community
Our system	Has its own test bed, independent of the editor file.	Supports only Java programs.

5. CONCLUSION

The existing systems such as Net Beans and Eclipse which are two of them have their own text editor for writing java programs and generating test cases for the same. But the drawback is that they cannot read as an input from any other java source file written in any of the text editors like notepad, command prompt etc.

6. FURTHER ENCHANCEMENTS

We have developed the test cases for Java it is possible to extend the project domain to JavaEE and Java ME. We aspire to generalize our domain for any program file extensions like .CPP or .C taken as an input for the application. These all can form sub-module and all the test generation of different languages can be integrated. It is also extend for resolving expressions with the search mechanism. The Speed at which the report is generated will be increased efficiently. In future improvement in the system Syntax analyzer can check for compilation errors.

REFERENCES

[1] Andreas S. Andreou, Christos Schizas, Gianna Ioakim, Extending and enhancing a basic program.
[2] Hiroaki Hashiura, Saekomatusuura ,Seiichikomiya, A Tool for Diagnosing the Quality of Java Program and a Method for its Effective Utilization in Education.
[3] Bor-Yuan Tsai∗, Simon Stobart, Norman Parrington, and Ian Mitchell, An Automatic Test Case Generator Derived from State-Based Testing.
[4] Bor-Yuan Tsai*, Simon Stobart, Norman Parrington and Ian Mitchell, Automated Class Testing
[5] Using Threaded Multi-way Trees to Represent the Behaviour of State Machines.
[6] Sinnott, R.O. (2003) Architecting specifications for test case generation. In: Cerone, A. and Lindsay, P.A. (eds.) First International Conference on Software Engineering and Formal Methods Proceedings: Brisbane, Australia, September 22 to 27, 2003. IEEE Computer Society, LosAlamitos, USA, pp. 24-32. ISBN 0769519490 http://eprints.gla.ac.uk/7292/
[7] Hojun Jaygarl, Kai-Shin Lu, Carl K. Chang, GenRed: A Tool for Generating and Reducing Object-Oriented Test Cases. Stefan Wappler and Joachim Wegener Evolutionary Unit Testing Of Object-Oriented Software Using A Hybrid Evolutionary Algorithm, 2006 IEEE Congress on Evolutionary Computation Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada July 16-21, 2006

Authors

Ms. Priya R. L, Lecturer, V.E.S Institute of Technology, having 8 years of teaching experience for undergraduate students of Computer Engineering and Information Technology disciplines in different Engineering Institutes, was obtained her Bachelors degree in Engineering from Manonmaniam Sundarnar University, Tirunelveli, Tamilnadu during the year 1999. Also, she had worked as a Software Engineer in different firms in Chennai and Mumbai for 4 years. Her research interest is more into Software testing, Service Oriented Architecture (SOA) and Web Engineering.

Ms Dhanamma Jagli,Lecturer ,V.E.S Institute of Technology, having total 8 year of teaching Ex perience for Post graduate and under graduate students of Master of Computer applications, computer Engineering, Inforamtion Technology and Electronics& Telecommunication disciplines in different Engineering institutes, was obtained Master Degree in Information Technology from Jawaharlal Nehru Technological University, Hyderabad, Andhra Pradesh during the year 2004.her research interest is more into Software Engineering, Data base Systems, Data mining and embedded Real time systems.