EATOOS-Testing Tool for Unit Testing of Object Oriented Software

S.Suguna Mallika

CVR College of Engineering,
Department of CSE
Ibrahimpatnam, R.R. District, A.P., India

ABSTRACT

With the advent of Object Oriented programming, most of the software being built is using object oriented programming languages. The major challenge lies with testing the software and it is a known fact that testing consumes around 40% of the time in the total software development process. If more number of errors are uncovered in the unit testing phase itself then the probability of propagation of errors to other components or phases gets reduced drastically. Similarly object oriented software also needs to be tested thoroughly at unit level. Unit level testing in OO software refers to the individual classes or group of classes which need to be tested. In the current work an attempt has been made to come up with an automated testing tool for unit testing of object oriented classes developed in java. It offers a single click automation solution for unit testing of OO software by providing the flexibility for the tester to choose the methods of his choice to test.

General Terms

Software Testing, Automated Testing Tools, Automated Object Oriented Testing.

Key Words

unit testing, automated testing, testing tool, class testing.

1. INTRODUCTION

Verifying implementation of a class means verifying the specification for that class. If so, each of the instances should behave properly. When it comes to testing of a class, it is analogous to testing the methods of the class. This paper presents a new testing execution tool for testing object oriented software. A testing framework EATOOS permits completely automated unit testing for classes of objects oriented software. The tool highlights the execution of test cases specifically method by method and especially the desired method which is present in all of the methods listed in the class.

2. LITERATURE SURVEY

Several unit testing tools like DART and JTest have been examined and used for unit testing. But some of the predominant drawbacks found with respect to these tools were that, the tester was supposed to sit and write a particular piece of test case in the proposed format of the tool and then execute the test case [2]. Some of the major challenges are

- The tester finds it difficult and tedious to generate the input and the corresponding oracles simultaneously.
- The tester would just have to go through the laborious process of sitting and coding every test case.[4].
- The tester cannot select a particular method that he was interested in testing with respect to a class.

3. IMPLEMENTATION

The software was divided into 4 different modules for the convenience of construction. The four modules are:

3.1 Test Data Generation

For the tester to execute test cases on a given source code file, he needs to primarily generate some data for executing the test cases [2]. For this, the tester needs to write a small script in line with the source code class file to be tested. By running this script, it would provide the tester with an interface to enter the inputs as test data for each particular method defined as part of the class. The test data entered would be stored in an xml format with the name of the class appended by the method name appended by either input or output. The inputs which were to be executed on the source code are stored as input xml file. And the expected outputs are stored as an output xml file.

3.2 Compilation and Method Retrieval

The Source code file i.e a class written in java when given as input to the tool is compiled and the class file corresponding to the java file is generated and the methods pertaining to the class are retrieved and displayed for the tester. The tester is then given a choice to select the methods which he would particularly like to test.

3.3 Dispatching Methods

Once the tester makes a selection of the methods to be tested, the corresponding input xml files if have been generated by the tester prior to choosing the methods are invoked and taken into the tool. The tool utilizes the data present in the input xml files and executes the test cases on the source code.

3.4 Report Generation and Logging

Once the test cases are run, the results obtained are compared with the expected results which have been stored as output xml files. If the actual result and the expected output are the same, then the tool displays a test case pass message otherwise a fail message. After showing the output to the tester, the results are stored in a file as log files.

4. CASE STUDY

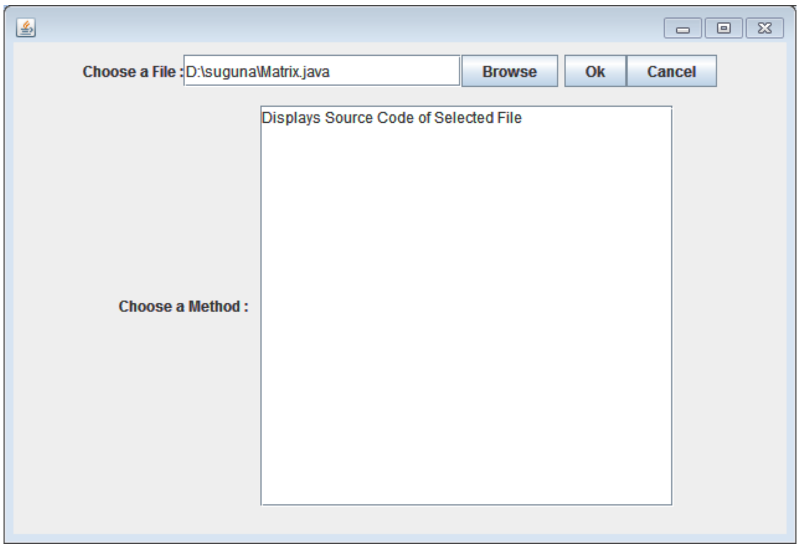
A sample case study of matrix multiplication class has been taken and code for the matrix class is as follows:

```
import java.util.Scanner;
public class Matrix {
private int[][] matrixElements;
private int rows;
private int cols;
public Matrix() {
rows = 2;
cols = 2;
matrixElements = new int[2][2];
}
public Matrix(int rows, int cols) {
this.rows = rows;
this.cols = cols;
this.matrixElements = new int[rows][cols];
}

public boolean equals(Matrix matrix){
boolean equalFlag = true;
if(this.rows != matrix.rows && this.cols != matrix.cols){
return false;
}
for(int rowIndex = 0;rowIndex<matrix.rows;
++rowIndex){
for(int colIndex =0;colIndex < matrix.cols;++colIndex){
if(this.matrixElements[rowIndex][colIndex] !=
matrix.matrixElements[rowIndex][colIndex]){
equalFlag = false;
return false;
}
}
}
return equalFlag;
}

public Matrix(int rows, int cols, int[][] matrix) {
this.rows = rows;
this.cols = cols;
this.matrixElements = new int[rows][cols];
for(int rowIndex=0; rowIndex <rows; ++rowIndex)
for(int colIndex=0; colIndex <cols; ++colIndex)
this.matrixElements[rowIndex][colIndex] =
matrix[rowIndex][colIndex];
}

public Matrix add(Matrix matrix1, Matrix matrix2) {
Matrix matrix3;
matrix3 = new Matrix(matrix1.rows, matrix2.cols);
for(int row=0; row < matrix1.rows; ++row)
for(int col=0; col < matrix1.cols; ++col)
matrix3.matrixElements[row][col] +=
matrix1.matrixElements[row][col] +
matrix2.matrixElements[row][col];
System.out.println(matrix3);
return matrix3;
}

public Matrix multiply(Matrix matrix1, Matrix matrix2)
{
if(matrix1.cols != matrix2.rows) {
System.out.println("\nMatrix Multiplication Not
possible\n");
System.exit(1);
}
Matrix matrix3;
matrix3= new Matrix(matrix1.rows, matrix2.cols);
for(int rowIndex=0; rowIndex < matrix1.rows;
++rowIndex)
for(int colIndex = 0; colIndex < matrix2.cols;
++colIndex)
for(int tempIndex=0; tempIndex < matrix1.cols;
++tempIndex)
matrix3.matrixElements[rowIndex][colIndex] +=
matrix1.matrixElements[rowIndex][tempIndex]
matrix2.matrixElements[tempIndex][colIndex];
return matrix3;
}

public Matrix transpose(Matrix matrix) {
Matrix transposedMatrix = new Matrix(matrix.cols,
matrix.rows);
for(int rowIndex=0; rowIndex < matrix.rows;
++rowIndex)
for(int colIndex = 0; colIndex < matrix.cols; ++colIndex)
transposedMatrix.matrixElements[colIndex][rowIndex] =
matrix.matrixElements[rowIndex][colIndex];
return transposedMatrix;
}
public String toString() {
String matrixResultString = "\n";
for(int row = 0; row < rows; ++row) {
1.  for(int col = 0; col < cols; ++col)
2.  matrixResultString += matrixElements[row][col] +
    "  ";
3.  matrixResultString += "\n\n";
4.  }
5.  return matrixResultString;
6.  }
}
```

5.RESULTS

The screenshots of the tool are as follows:

Step1: Select the required java class file to be tested as shown in fig. 1.



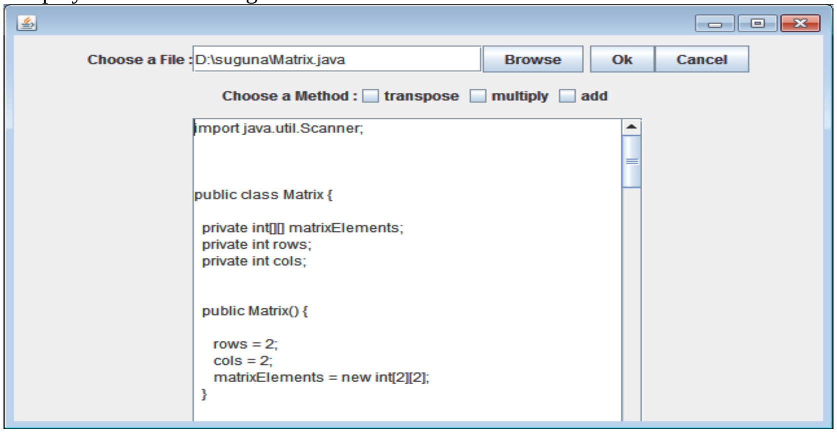
Choose a File : D:\suguna\Matrix.java

Displays Source Code of Selected File

Choose a Method :

Fig 1: GUI showing Import File Feature

Step2: After browsing the appropriate file and clicking on 'Ok' button, the methods of the Class with checkboxes and the code of the class are displayed as shown in fig. 2.



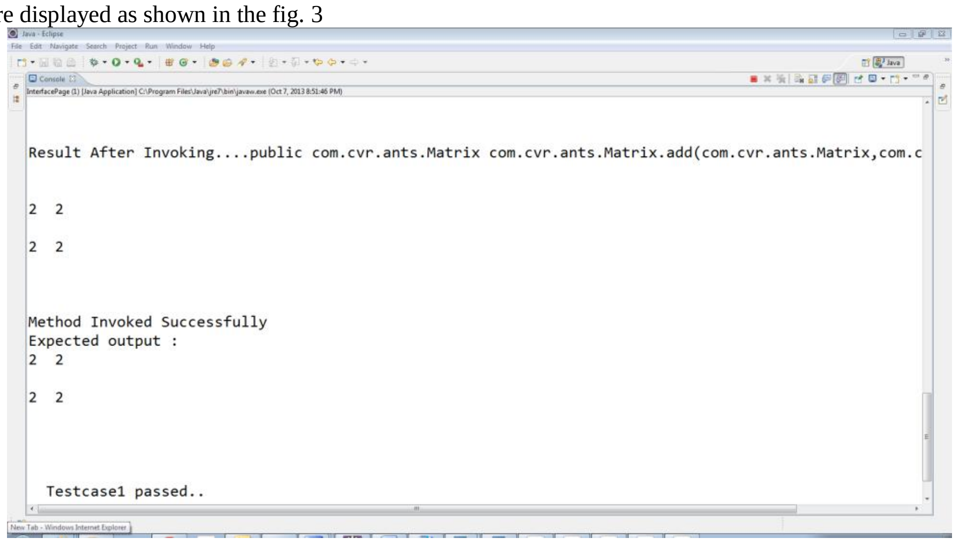
Choose a File : D:\suguna\Matrix.java

Choose a Method : ☐ transpose ☐ multiply ☐ add

```
import java.util.Scanner;


public class Matrix {

 private int[][] matrixElements;
 private int rows;
 private int cols;


 public Matrix() {

    rows = 2;
    cols = 2;
    matrixElements = new int[2][2];
 }
```

Fig 2: GUI displaying the code and methods

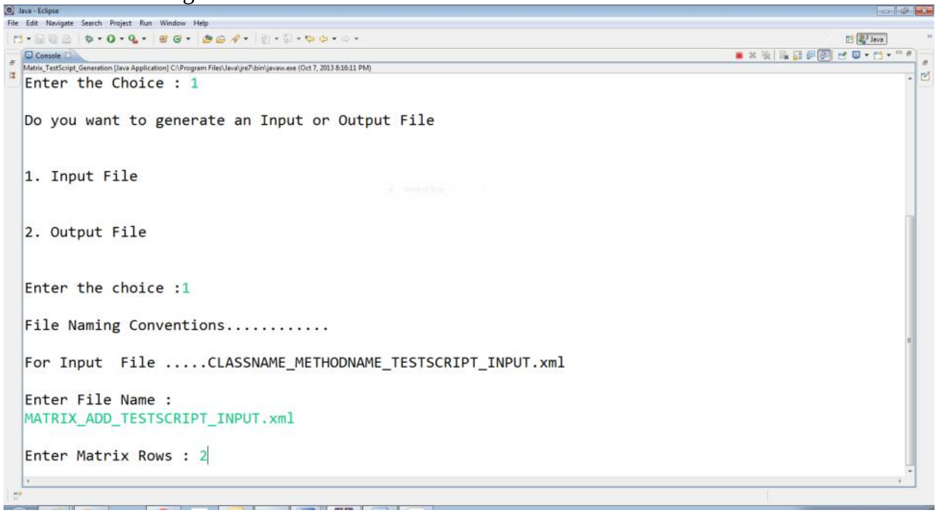
Step3: After selecting the appropriate methods by the tester, the test cases corresponding to that method are executed and the results are displayed as shown in the fig. 3



```
InterfacePage (1) [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Oct 7, 2013 8:51:46 PM)


Result After Invoking....public com.cvr.ants.Matrix com.cvr.ants.Matrix.add(com.cvr.ants.Matrix,com.c


2  2

2  2



Method Invoked Successfully
Expected output :
2  2

2  2




  Testcase1 passed..
```

Fig. 3: GUI displaying results of test cases

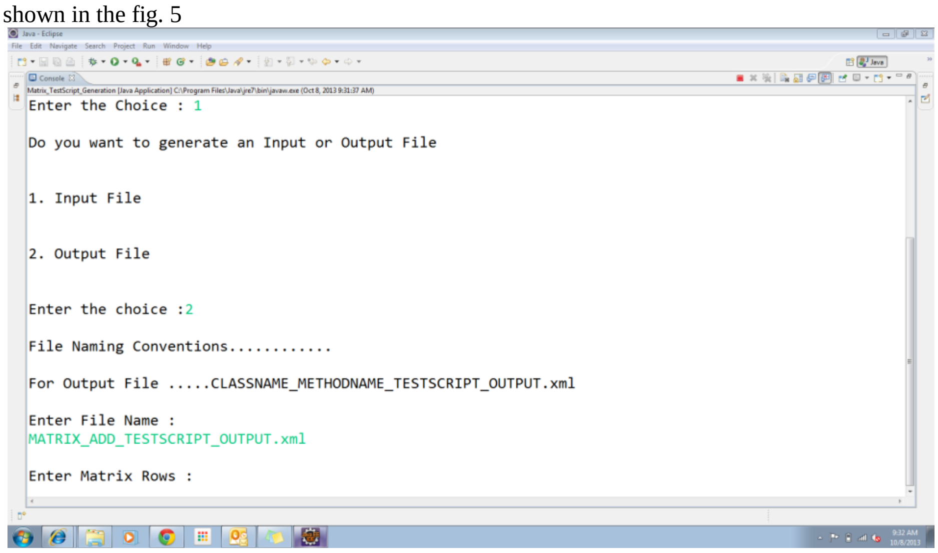
Step4: In order for the tester to run the test cases for a particular method, it is mandatory to generate the corresponding input test script file by using the console provided for the tester as shown in fig. 4.



```
Enter the Choice : 1

Do you want to generate an Input or Output File


1. Input File


2. Output File


Enter the choice :1

File Naming Conventions...........

For Input  File .....CLASSNAME_METHODNAME_TESTSCRIPT_INPUT.xml

Enter File Name :
MATRIX_ADD_TESTSCRIPT_INPUT.xml

Enter Matrix Rows : 2
```

Fig. 4: GUI for generating InputTestScript File

Step5: The corresponding Expected Output file for each method also has to be generated by the tester by using the output console as shown in the fig. 5



```
Enter the Choice : 1

Do you want to generate an Input or Output File


1. Input File


2. Output File


Enter the choice :2

File Naming Conventions...........

For Output File .....CLASSNAME_METHODNAME_TESTSCRIPT_OUTPUT.xml

Enter File Name :
MATRIX_ADD_TESTSCRIPT_OUTPUT.xml

Enter Matrix Rows :
```

Fig 5: GUI for generating the ExpectedOutputScript File

6. CONCLUSIONS

- ➢ EATOOS is certainly a powerful tool overcoming some of the most important shortcomings of the existing tools like ARTOO, JUNIT et. al. [3].
- ➢ It has the capability of extracting the methods defined in the class file automatically.
- ➢ It gives the flexibility to the tester to select only those methods of his choice for running the test cases.
- ➢ The tool is competent enough to automatically retrieve and place objects in the input and the output xml files.

7. LIMITATIONS AND FUTURE WORK

- ➢ The current tool has certain limitations in terms of its inability to automatically generate an automated test data generation script for the source code file given as input to the tool.
- ➢ Future Work should include developing a completely generic framework which would automatically generate the test cases as well.
- ➢ Generate the test generation script by parsing the input source code file.
- ➢ If the methods are not having parameters which can be passed then the generation of input script file is difficult. This limitation has to be rectified in the future work.

8. ACKNOWLEDGEMENTS
My sincere thanks to Mr.K. Narendar Reddy, Professor, Department of IT, CVR College of Engineering, for his kind support and encouragement in the development of this paper. My heartfelt gratitude to Mr. J. Vamshi Vijaya Krishna, Assistant Professor, Department of IT, for his support in the development of this tool.

9. REFERENCES

[1] C. Boyapati, S. Khurshid and D. Marinov: Korat: Automated Testing Based on Java Predicates, in 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2002), Rome, Italy, 2002.

[2] Robert V. Binder: Testing Object-Oriented Systems: Models, Patterns and Tools, Addison-Wesley, 1999.

[3] TY Chen, H. Leung and I. Mak: Adaptive random testing, in M. J. Maher (ed.), Advances in Computer Science – ASIAN 2004: Higher-Level Decision Making, 9th Asian Computing Science Conference, Springer-Verlag, 2004.

[4] Yoonsik Cheon and Gary T. Leavens: A Simple and Practical Approach to Unit Testing: The JML and JUnit Way, in ECOOP 2002 (Proceedings of European Conference on Object-Oriented Programming, Malaga, 2002), ed. Boris Magnusson, Lecture Notes in Computer Science 2374, Springer Verlag , 2002, pages 231-255.

[5] Yoonsik Cheon and Gary T. Leavens: The JML and JUnit Way of Unit Testing and its Implementation, Technical Report 04-02, Computer Science Department, Iowa State University,atarchives.cs.iastate.edu/documents/disk0/00/00/03/27/00000327-00/TR.pdf.

[6] Bertrand Meyer, Ilinca Ciupa, Andreas Leitner and Lisa (Ling) Liu, Automatic Testing of Object-Oriented Software, in SOFSEM 2007 (Current Trends in Theory and Practice of Computer Science, Harrachov, Czech Republic, 20-26 January 2007), ed. Jan van Leeuwen, to appear in Lecture Notes in Computer Science, Springer-Verlag, 2007.

[7] Jitendra S.Kushwah, Mahendra S. Yadav, TESTING FOR OBJECT ORIENTED SOFTWARE, Indian Journal of Computer Science and Engineering (IJCSE), Volume2, No.1.

[8] Automatic Test Factoring for Java – David Saff, Shay Artzi, Jeff H.Perkins, Michael D. Ernst

[9] Selective Capture and Replay of Program Executions – Alessandro Orso and Bryan Kennedy

[10] Eclat: Automatic Generation and Classification of Test Inputs – Carlos Pacheco, Michael D. Ernst

[11] Orstra: Augmenting Automatically Generated Unit-Test Suites with Regression Oracle Checking

[12] Substra: A Framework for Automatic Generation of Integration Tests – Hai Yuan, Tao Xie

[13] Symstra: A Framework for Generating Object-Oriented Unit Tests Using Symbolic Execution – Tao Xie, Darko Marinov, Wolfram Schulte, David Notkin