ISSN (Online) : 0974-5645
ISSN (Print) : 0974-6846
DOI: 10.17485/ijst/2015/v8iS1/59426

Indian Journal of Science and Technology, *Vol 8(S1), 475–480, January 2015*

Design and Implementation of Java Dynamic Testing Tool using Instrumentation

Sun-Myung Hwang[1] and Jihyun Lee[2*]

[1]Computer Engineering Department, Daejeon University, Daejeon, 300-716, SouthKorea; sunhwang@dju.kr
[2]College of Liberal Arts, Daejeon University, Daejeon, 300-716, South Korea; jihyun30@dju.kr

Abstract

This work reports the design of Java Dynamic Testing tool by using Instrumentation, which is the statement inserted into the program to count the number of iteration process and conditional statement process in Java Programming by must not affecting the program behavior. In addition, it focuses on Java Source Instrumentation to count the number process of Iteration and conditional statement process by using three main approaches for designing this Dynamic Java Testing tool such as Java controller, Java Iteration and Iteration Counting Process. The Java Dynamic Testing by using Instrumentation is to give the engineer manager the report of counting iteration process which is the main key indicator of knowing the software cost and time as well as the input for estimating project effort which use to calculate productivity and other measurements.

Keywords: Instrumentation, Iteration, Java Parsing, Java Dynamic Testing Tool

1. Introduction

Obviously, there have been considerable researches of testing tool of software for different kind of purposes. Most of the software testing tool is an investigation conducted to provide stakeholders product quality information by using their own technique of testing tools[1–7].

Moreover, previous studies centered on using instrumentation by inserting parameters into java byte code and java source code file. However they used three techniques which specialised only on thread, safety and data-flow structure which detects only static java parallel loops[8–12]. Yet, it does not research on Dynamic Java which using instrumentation on java source file to count the number of processes of iteration and conditional statement process number.

The purpose of the research is to count the number of iteration process and to find out the difficulty and easiness of using Instrumentation, by inserting java source code file parameter. Therefore, this java tool will compile and display the result of process counting of iteration and conditional statement process. As shown in Figure 1, the Java Dynamic Testing tool by using Instrumentation consists of three main approach such as Java controller, Java Iteration which the parser to detect the iteration for inserting the parameter, and Iteration Counting Process. In addition, the objective of this research is to find out the similarity and difference from most of the previous testing tools in terms of speed and algorithm. Therefore, this study will provide an important role of further research on testing tool by using Instrumentation to inserting parameter into java source code file to count the number process of iteration and conditional statement. The scope and limitation of this research will be illustrated as follows:

- The instrumentation will use only the standard of java source code file
- Java source code file has to ensure without any errors before inserting parameters for creating instrumented file.
- This research will only focus on iteration
- Count the process of iteration
- Count the process of conditional statement

*Author for correspondence

Figure 1. Result of counting process iteration and conditional statement of bubble sort sour code file.

2. Methodology

During the running time, JVM (Java Virtual Machine) will be automatically run the instrumented file which will demonstrate to display the result of java source file and the counting number process of iteration and conditional statement[13,14]. As it will be show in Figure 1, which is the result of instrumented file after executing. Parser has role as the scanning the input of java source code file to detect and check which all the input provided. In this paper, Java Dynamic testing tool class IterationDetection is the parser to detect iteration of Java source file programming language.

In Figure 2, three main approaches in general concept of testing tool are illustrated, and for Java Dynamic testing tool in this paper, the three approaches will be included such as Java controller, Java Iteration and Iteration counting process[15–17]. Table 1 show the result the Java input source code file and the instrumented file after applying all main three approaches:

- Parser has a role as the scanning the input of java source code file to detect and check which all the input provided. In this paper, Java Dynamic testing tool class
- IterationDetection has the role as the parser to detect iteration as well as the conditional statement from Java source file programming language
- Instrumentor is the file of instrumented file after inserting the parameter into source code file for waiting the process of execution to output the result of report the number process Iteration of this Java Dynamic testing tool as it is shown in Table 1
- Test Report for Java Dynamic testing tool in this paper will display the result of java source file and importantly the result of the iteration process count and the process of conditional statement for the computer manager to figure out the fastness and slowness of the project.

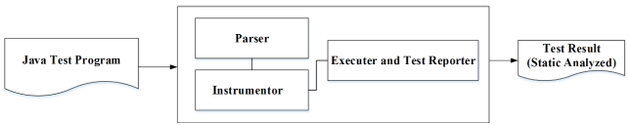

Figure 2. Java dynamic testing tool by using instrument method general process.

There are three main approaches of the Java Dynamic Testing tool such as Java controller, Java Iteration and Iteration Counting Process.

- JReadWrite class has two significant roles as the Java Controller to open Java programming source code file to create the instrumentation file and to back up the original source code file into the new folder called backup folder. Also, JReadWrite class will read line by line to call class IterationDection for scanning and detecting loops in Java Source code file.
- IterationDetection class functions as a parser to detect three types of iteration process such as for loop, do while loop and while loop as it is shown in Figure 2-3. The method of detecting iteration in class ItearationDetection.
- NumProcess class shows the analysis code that is invoked when the number process is encountered. The special technique method of NumProcess class is to count iteration process; however, by counting the process is not simple to count because it is needed to know whether it is the same loop name or not.

In case, there is the same loop name, the new object will not be created but it will increase the previous index value.Therefore, in case that the loop does not exit, it will create a new loop name and start to increment the values of index as it is shown in Figure 3, the method of NumberProcess for counting process in both cases. Also, for the conditional statement in case the conditional statement will be count the process if the conditional is true and if the conditional is false, instrumented also will be inserted and count the process as well.

The result of instrumentation tabulated as in Table 1, and it displays the input Java source code file which is one of the examples of java source code file of bubble sort

Table 1. Input Java source file and instrumented file

Input Java Source code	Instrumented file

import java.util.Scanner;
class BubbleSort {
……
for (c = 0; c < n; c++)
array[c] = in.nextInt();
for (c = 0; c < (n - 1); c++) {
for (d = 0; d < n - c - 1; d++) {
if (array[d] > array[d+1]) {
swap = array[d];
array[d] = array[d+1];
array[d+1] = swap;}}}
System.out.println("Sorted list of numbers");
for (k = 0; k < n; k++)
System.out.println(array[k]);
}
}
	import java.util.Scanner;
	class BubbleSort {
	…….
	for (c = 0; c < n; c++)
	{NumProcess.setNumprocess("for (c = 0; c < n; c++)"); //
	Instrumenation
	array[c] = in.nextInt();}
	for (c = 0; c < (n - 1); c++) {
	NumProcess.setNumprocess("for (c = 0; c < (n - 1); c++) {");
	//Instrumentation
	for (d = 0; d < n - c - 1; d++) {
	NumProcess.setNumprocess("for (d = 0; d < n - c - 1; d++) {");
	//Instrumentation
	if (array[d] > array[d+1]) /* For descending order use < */
	{
	NumProcess.setNumprocess("if (array[d] > array[d+1]) /* For
	descending order use < */"); //Instrumenation
	…….
	}}}
	System.out.println("Sorted list of numbers");
	for (k = 0; k < n; k++)
	{NumProcess.setNumprocess("for (k = 0; k < n; k++)"); //
	Instrumenation
	System.out.println(array[k]);
	}}}

and the instrumented file of it, while Table 1 contains the result of counting both process of source code file and the instrumented file.

ReadWrite class has an important function as the scanning by reading line by line from the java source code and it has an object to point for asking IterationDetection whether it has iteration or conditional statement for counting the process. Then, for the IterationDetection has two major function to detect and how to count iteration and conditional statement.

For the Iteration counting process, it has three type of loop such as *For loop*, *While loop*, and *Do while loop*.

In case of *For loop* and *While loop* the return type has 1, 2, and 3. The meaning of each return type of *For loop* and *While loop* is as follows:

- Return 1: means the open brace "{" is at the same line with *For loop* or *While loop*.
- Return 2: mean the open brace "{"is not the same line with *For loop* or *While loop*.
- Return 3: mean there is no open and close brace, therefore the open and close brace must be created before inserting the instrumented program into the Java source code.

The special case for *Do while loop* it has only two type of return number such as 1 and 3 which means that there is no such a case of *Do while loop* statement that do not have open and close brace. It means that *Do while loop* must have open and close brace by referring to the standard of java program language code.

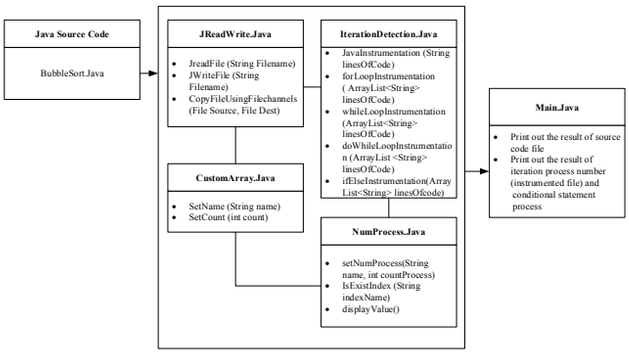

Figure 3. Class diagram of java dynamic testing tool by using instrumentation.

For the conditional statement counting process, it has three type for detection such as *If, Else if,* and *Else*.

Actually, the return type of 1 is if statement and return of 3 is else if statement and last but not least for the else statement it returns 2.

In case of return 2 means that if/else if/ else statement within only one statement, thus it needs to add open and close brace "{", "}" then insert the instrumented program.

CustomArray is the class which for set name and get name for the array one dimensional of counting the process iteration and conditional statement.

NumProcess class is the class for counting the number of process iteration and conditional statement. In case if it has the same name of previous iteration or conditional statement, the values of the array location will be increased. On the other hand, if it does not have the same name of previous iteration or conditional statement, the new location of array will be increased and start to count the new value of that array.

Finally, for the main class JExecute, it has function to let the programmer or user to choose the standard of Java programming language source code to test. Firstly, it needed to compile and after the executed, the Java Dynamic testing tool will execute the instrumented file which already created from the source code file by display the result of Java source code file and the number process of iteration and conditional statement. The new folder also has been created for storing the original file which gives a facility and usefulness for the user to see the different between instrumented file and Java Programming source code file.

3. Implementation and Evaluation Result

In order to get into the result the reporting of this Dynamic testing tool, some requirement of software and version of software are required to implement and exit which is illustrated in the Table 2.

To make it easy in implementation and evaluate the java source code instrumentation, we develop a tool called Java Instrumentation Tool as shown in Figure 4.

In this tool, user can do instrumentation with a java file or a collection of java files in a project. In addition, user also can count the number of line of code and functions in Java Source code at the same time.

Table 2. Testing tool requirement

Software Requirement Name	Version of Software Requirement
Windows XP, Vista, 7, 8	• Windows XP Starter • Windows XP Home • Windows XP Professional • Windows XP Professional x64 • Windows XP 64-bit Edition • Windows 7 Vista Home Premium • Windows 7 Vista Business • Windows 7 Vista Enterprise • Windows 7 Vista Ultimate • Window 8 • Window 8 Pro • Window 8 Enterprise
JCreator	• JCreatorXinox Software version 4.50.010
Java Source Code	• Follow the Java standard rule
JDK	• JDK 1.6.0 • JDK 1.4.2 • JDK 1.2.2 • JDK 1.0.2

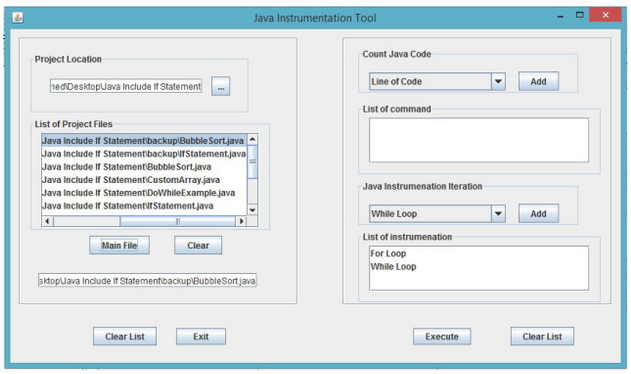

Figure 4. Java instrumentation tool.

To evaluate our Java Instrumentation testing tool, we have randomly selected 10 Java projects as shown in the Table 3 for testing.

4. Conclusion

The main purpose of this Java Dynamic Testing is to utilize Instrumentation to provide the engineer manager the report of iteration process estimation, one of the

Table 3. Java source code example testing

Project Name	Project Description	Line of Code
Insertion Sort	This is a Java implementation of the "Insertion Sort" algorithm using "integer arrays" that sorts elements of arrays progressively by inserting each one of them in the right position.	56 lines
SelectionSort	The "Selection Sort Algorithm" is one of the most intuitive algorithm that was invented for sorting a list of elements in ascending or descending order. This algorithm is in general quicker than the "Bubble Sort Algorithm".	61 lines
BinarySearch	Using Binary Search algorithm to find a search value in a array	43 lines
Day of the Week	Using loop, switches and validation for defining the day of the weeks	203 lines
Excel demo	To read the excel file sheet name	77 lines
Linear Search Recursion	To search a value by using linear search algorithm	41 lines
Multi Array Search	To search a value in an array two dimension to find row and column number	35 lines
Bubble Search Number	Using bubble search algorithm to find input number into the list of number input by user	33 lines
Token Example	Takes a pre-defined string and breaks it into and array of tokens (single words). The array of tokens is then searched for the keyword and a response is given depending on its presence.	62 lines
Bubble Sort	To sort value input by user by using bubble sort algorithm	49 lines

main key indicators of software cost and time. In this paper, three main approaches—Java controller, Java Iteration and Iteration Counting Process were used for counting iteration process and conditional statement in Java Source Code; also, this Dynamic Java Testing Tool using Instrumentation is useful to apply the Java source code Instrumentation to insert into source code of Java programming to count the number process of iteration and conditional statement by using Java controller, Java Iteration and Iteration Counting Process. The benefit of analysing the number process of the Java Dynamic Testing tool by using source code instrumentation is the main key point for the computer manager to estimate the project effort as well as to calculate the productivity; since the number process of iteration and conditional statement which is the main report for the slowness and fastness of one programming source code file. To extend the research, the following works will be carried out:

- To count the coverage of Java source code such as branch coverage, statement coverage and function call coverage.
- To be able to be used with other source code file to input beside Java Source code file.
- To have interface for more facility for the user to display the report of counting process of iteration and conditional statement.

5. Acknowledgement

This work was supported by Business for Cooperative R&D between Industry, Academy, and Research Institute funded Korea Small and Medium Business Administration in 2014.

6. References

1. Korel B. A dynamic approach of test data generation. In: Proceedings of IEEE Conference on Software Maintenance; 1990. p. 311–7.
2. Von Praun C, Gross TR. Object race detection. In: Proceedings of the 16thACM SIGPLAN conference on Object-oriented Programming, Systems, Languages, and Applications, OOPSLA'01; 2001. p. 70–82.
3. Dig D, Tarce M, Radoi C, Minea M, Johnson R. Relooper: refactoring for loop parallelism in java. In: Proceedings of the 24thACM SIGPLAN conference companion on Object oriented programming systems languages and applications, OOPSLA'09; 2009. p. 793–4.

4. Hwang G, Ta, K. Hunag T. Reachability testing: an approach to testing concurrent software. Int J Software Eng Knowl Eng. 1995. 5(4):493–510.
5. Lee HB. BIT: A tool for instrumenting java bytecodes. In: Proceedings of USENIX Symposium on Internet Technologies and Systems; 1997..p. 73–82.
6. Sommerville I. Software engineering. 6th ed. Addison Wesley; 2000.
7. Bull JM, Smith LA, Westhead MD, Henty DS, Davey RA. A benchmark suite for high performance java. In: Proceedings of Java Grande. 1999. p. 81–8.
8. Nethercote N, Seward J. Valgrind: A framework for heavy-weight dynamic binary instrumentation. In: Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation, PLDI'07; 2007.
9. Pratikakis P, Foster JS, Hicks M. LOCKSMITH: practical static race detection for C. ACM Trans Program Lang Syst. 2011; 33(1):1–55.
10. Hastings R, Joyce B. Purify: fast detection of memory leaks and access errors. In: Proceedings of the Winter USENIX Conference; 1992. p. 125–36.
11. Hwang SM. A construction of software quality assurance tool with complexity metrics based on regular expression. 1987.
12. Zheng Y, Ansaloni D, Marek L, Sewe A, Binder W, Villazon, et al. DiSL: Partial evaluation for high-level bytecode instrumentation. In: Objects, Models, Components, Patterns. Lecture Notes in Computer Science. 2012; 7304:353–68
13. Thomas J, Young M, Brown K, Glover A. Java testing patterns. John Wiley and Sons; 2004.
14. Arnold K, Gosling J. The java programming language. Addison-Wesley; 1996.
15. Myers J. The Art of software testing. New York, USA: 1979.
16. Ammann P, Offutt J. Introduction to software testing. Cambridge; UK: 2008.
17. Patton R. Software testing. 2nd ed. SAMS; Indiana, USA: 2005.