ISSN: 2321-7782 (Online)
Volume 4, Issue 2, February 2016
International Journal of Advance Research in Computer Science and Management Studies
Research Article / Survey Paper / Case Study
Available online at: www.ijarcsms.com

Analysis of code coverage metrics using eCobertura and EclEmma: A case study for sorting programs

Sarita Pathy[1]
P.G Dept. of Computer Science and Application,
Jyoti Vihar, Sambalpur University
Sambalpur, Odisha - India

Dr. Sarada Baboo[2]
Reader: P.G Dept. of Computer Science and Application,
Jyoti Vihar, Sambalpur University
Sambalpur, Odisha - India

Abstract: ***Quality is the main characteristics or criteria for any software. Software has been developing using different phases of software Development Life cycle (SDLC) in which 50% of effort goes to testing. Code coverage is one of the most significant indicators of software testing. Code coverage analysis helps in evaluating the testing effectiveness. Various techniques are carried out to reduce the testing effort time as well as to accept all the criterion of user requirements, out of which automated testing tools are good examples for generating test cases. A best quantitative and decision making approach of software testing is by considering software metrics. In this paper code coverage metrics is considered using control flow graph and two open source tools that are eCobertura and EclEmma are chosen which are plug- in with eclipse IDE to find the line and branch coverage for a set of sorting programs implemented in java. These sorting programs are considered as a case study.***

Keywords: code coverage, line coverage, branch coverage, code coverage tools, CFG, UML diagram, McCabe's Cyclomatic Complexity.

I. INTRODUCTION

Code coverage analysis evaluate the code structures exercised by a given test or set of tests. Generally code coverage expressed as a percentage and a portion of software that is more thoroughly tested, higher the code coverage percentage, smaller the chance that the software contains defects [1].Several industry standards needs the use of code coverage as just the means of proving the test completeness. Many software companies highly focus on testing of software effectiveness. Testing of software reduces the probability of software failures.

There are two important factors are considered for code writing that is it must be accurate and followed effective testing. It is really difficult to write test cases for every class, every method or every lines of code in a program. It is very difficult to know when we required enough tests. It's extremely difficult to determine the 100% testing of code. So the uses of code coverage tools are introduced and it can avoid those problems. Code coverage plays an important role to provide software effectiveness. Code coverage provides a quantitative measure, which is used as indicator of reliability of software products [2]. So code coverage is adopted by every software development projects.

Code coverage is effectual to facilitate in test cases prioritization and generation, which reduces the effort and cost, increases the number of effective test cases [3]. Testing with the goal of maximizing the code coverage can remove many problems such as race conditions, misplaced codes, data flow errors etc.

Object oriented programming is a popular concept in today's software development environment. So we need to recognize the quality of object oriented programming by testing the codes with appropriate code coverage analysis.

© 2016, IJARCSMS All Rights Reserved

In this paper eCobertura and EclEmma tool are used for code coverage analysis of java programs taking as input and an algorithm is proposed for code coverage analysis for desired percentage. Also find the McCabe's Cyclomatic Complexity from the analysis of code coverage using control flow graph(CFG).Then generate Unified Modelling Language(UML) class diagram for every java program .These two tools are an innovative test coverage tools used to provide an effective and intuitive graphical representation and visually evaluate the quality of the coverage. These automated tools are used to find the coverage of test cases in Java. It can be used to identify which parts of the Java program are missing test coverage.

II. RELATED WORK

Cai and Lyu reported that code coverage is an important indicator for the capability of fault detection on a normal test set. They also found out the effect of code coverage on fault detection varied based on the test .They conclude that the relationship between code coverage and fault coverage was higher in case of structural testing than random testing. Functional test cases are more effective than random test cases in determining the fault detection effectiveness of a test set [4]. K.Ali Alemerien *et al.*, conducted an experiment to investigate the difference among code coverage tools in terms of statement, branch and method metrics and effectiveness of code coverage tools and conclude that eCobertura tools is more useful to calculate the percentage of statement and branch coverage metrics for a program than other tool. The source code are executing through Junit frameworks, this tools show the result of testing coverage process [5].

A.M.R. Vincenzi *et al.*, purposed a technique that based on control flow, data flow based coverage criteria and a software testing tool named JaButi that supports the testing of java programs .Control flow testing is described using all nodes and all edges criteria with the help of the def-use graph representation of a program. Data flow based testing is explained based on all-uses criteria comprising of all p-uses and c-uses criteria [6].

H.singh *et al*., used the control-flow and data-flow criteria to support the testing of Java programs (Java bytecode) aimed at intra-method structural testing of code and its components based on various testing criteria. Used a testing tool, named JaBUTi (Java Bytecode Understanding and Testing), which supports the application of such criteria for testing Java programs and components, which is used for structural testing and code coverage analysis [7]

Kajo-Mece and Tartari perform an experiment that examined two code coverage tools Emma and Clover using java programs for search and sort algorithms [8].

Priya *et al.*, perform an experiment to examine the set of metrics to support testing procedural software. Considered nine small programs and four code coverage tools to calculate the proposed metrics but they did not focus on the inconsistency in values of coverage metrics [9]. Both Yang et al. [10] and Shahid and Ibrahim [11] compared coverage-based testing tools for the following characteristics: programming languages supported, instrumentation levels, code coverage criteria, and report formats and they provided guidelines for researchers to select the appropriate code coverage tool.

From the review point of view, various types of code coverage, the techniques and tools are reviewed from different literatures. It is concluded from summary of code coverage analysis metrics that line and branch code coverage is the simplest way of identifying which area of a program is effected and which is not effected by a set of test cases. Here eCobertura and EclEmma tools are chosen to find the line and branch coverage for a set of sorting programs execute in java.

© 2016, IJARCSMS All Rights Reserved

III. THE PROPOSED METHODOLOGY

a) Tool Suite Used

In this paper the following tools are used for calculating the code coverage of java programs.

1. eCobertura:

eCobertura is a free Eclipse plug-in that calculates the percentage of code coverage before generation of test cases [12]. It can be used to recognize which parts of java program are missing in the test coverage. After executing the source code through JUnit framework, the results of testing coverage process are shown by the eCobertura. eCobertura tool used as a code coverage reporting tool. eCobertura is used to calculate the percentage of line and branch coverage metrics for each package, each class, and for the whole program.

Basing on the coverage mode the source files are appear in different colors. So, the piece of source code that is accessed by test cases is colored in green whereas the untested part is appeared as red color.

Advantages of eCobertura

- The detailed line and branch coverage results are shown in a tree view.
- According to line coverage source code are appeared as different colored.
- The classes and packages which are unnecessary are sifted out.

2. EclEmma:

It is an open source tool for java code coverage. EclEmma is used to calculate the percentage of statement coverage metric. This tool highlights in green for fully covered code, uncovered code are displayed in red color. Percentage of code coverage can be evaluated within the eclipse IDE interactive development environment by selecting the "Cover As" command from the run menu. EclEmma is a free Java code coverage tool for Eclipse, available under the Eclipse Public Licence [13].

Advantages of EclEmma

EclEmma is called in coverage mode and works like run and debug mode.

- It provides the summaries lists of coverage section of java programs.
- It also summarized the different instructions, branches, lines, methods, types or Cyclomatic Complexity.
- It provides the switching between coverage data from multiple sessions.

3. Eclipse 3.8:

Eclipse is an open source Integrated Development Environment (IDE) java application. Eclipse can be installed in different operating systems such as windows, Linux, Mac etc. Eclipse IDE can be used for different programming languages such as Java, C, C++ etc [14].

4. Graphviz:

Graph visualization software (Graphviz) is a package of open source tool used for representing information as graphs. This tool is developed in AT and T Research Laboratories. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks [15].

b) Pseudo-code for code coverage:

The following variables are used for representation of different terms.

© 2016, IJARCSMS All Rights Reserved

BT- branches that evaluated to 'true' at least once.

BF - branches that evaluated to 'false' at least once.

SC -Statement Coverage.

MC-Method entered.

TB-Total number of branches.

TS- Total number of statements.

TM - Total number of methods.

B – Branch

L – Line

D – Descriptor

J-Jump

hit – counter variable

Total Coverage Percentage (TCP) is

(TCP) = (BT+BF+SC+MC)/ (2*TB+TS+TM)*100 %.....(1) [16].

Algorithm in the form of pseudo code is shown below:

1. Generate the Abstract Syntax Tree (AST) of the input java program.
2. Initialize the coverage type B to 0, L to 0, j to 0, TB to 0, BT to 0, BF to 0.
3. Initialize hit to 0, TS to 0 and D to 0.
4. Count the number of lines from AST, line number using hash code.
5. Find the method entered (MC) and set the descriptor D and increment correspondingly.
6. Find the line having branch (B), increment j for each jump to find out number of branches and increment TB.
7. Calculate total number of lines TL.
8. Calculate the statements covered (SC), using run test (test case file generated during execution).
9. Find the branch covered BT and BF.
10. Find TB, TS and TM by calculating the incremented j, hit, TS and D.
11. Calculate the line (LP) in % = SC/TS*100 and branch(LP) in %=(BT+BF)/TB*100
12. Calculate TCP = (BT+BF+SC+MC)/ (2*TB+TS+TM)*100%
13. Return LP, BP and TCP.
14. Generate CFG and UML diagram.

c) Explanation of the Algorithm

In the tool first input the java program then it converts it to the abstract syntax tree. Initialize the branch coverage, line coverage, jump, total number of branches, branches that evaluated to "true" at least once and branches that evaluated to false at least once to zero. Again initialize hit, total number of statements and descriptor to zero. Descriptor increments when the

© 2016, IJARCSMS All Rights Reserved

numbers of methods are covered. Hit is a counter variable, it initialized to zero first then it incremented when a statement is covered. Then count the number of lines from abstract syntax tree and line number using hash code. Hash code used to count the number of lines in a node. Find the methods which have been entered and set the descriptor then increment correspondingly. Find out the line having branch, increment j for each jump to find out the number of branches and increment total number of branches. Calculate total number of lines. Calculate the statements covered using run tests that are generated during execution. Find the branch covered by the branches that evaluated to true and branches that evaluate to false at least once. Then find out the total number of branch, total number of statements and total number of methods by calculating the incremented jump, hit, total statements and descriptor. Calculate the percentage of line coverage by using the formula LP in % = SC/TS *100 and calculate the percentage of branch coverage by using the formula BP in % = (BT+BF)/TB*100.Finally calculate the total coverage of percentage by using the formula TCP = (BT+BF+SC+MC)/ (2*TB+TS+TM)*100.Then returns the final results line of percentage, branch percentage, and total coverage percentages. Accordingly the control flow graph, source code graph and the unified modelling diagrams are generated.

d) Flow Chart

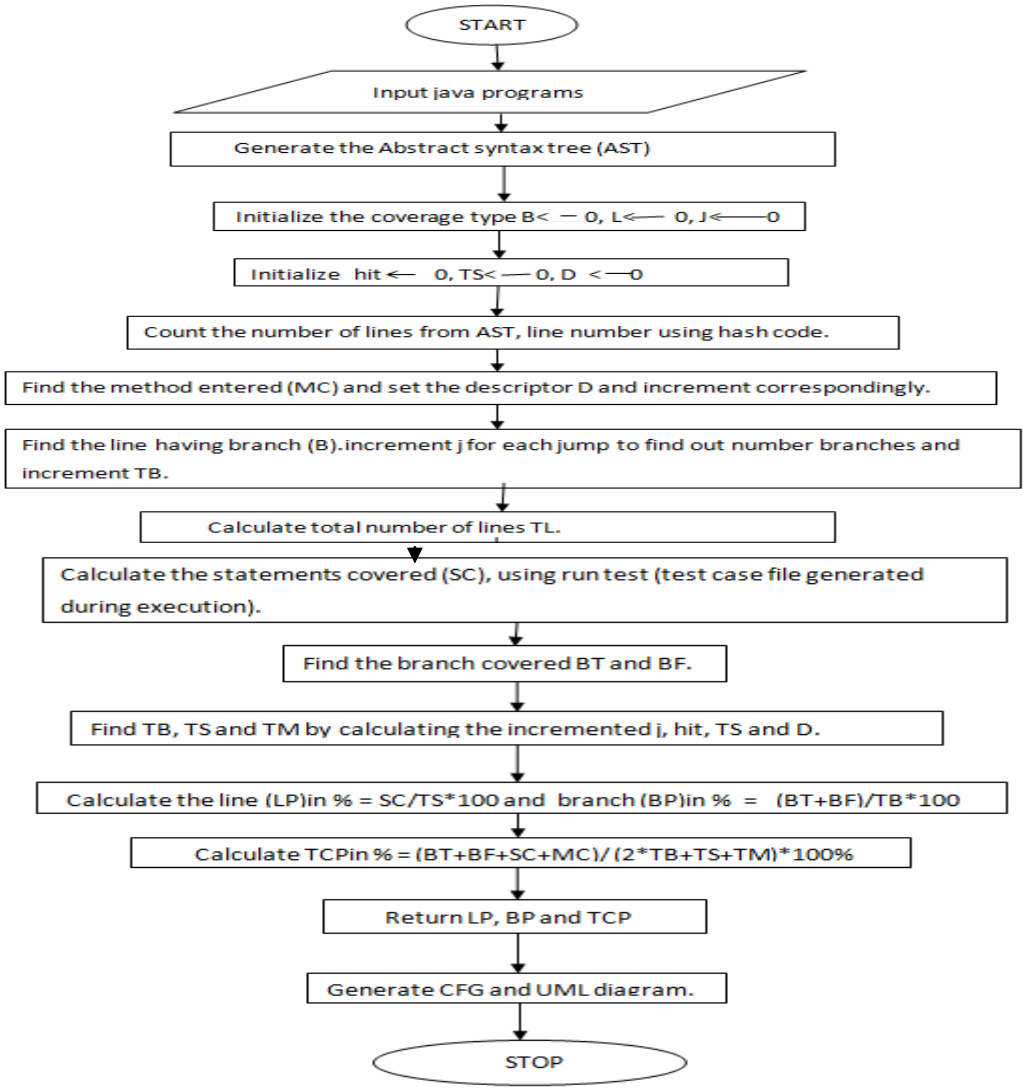
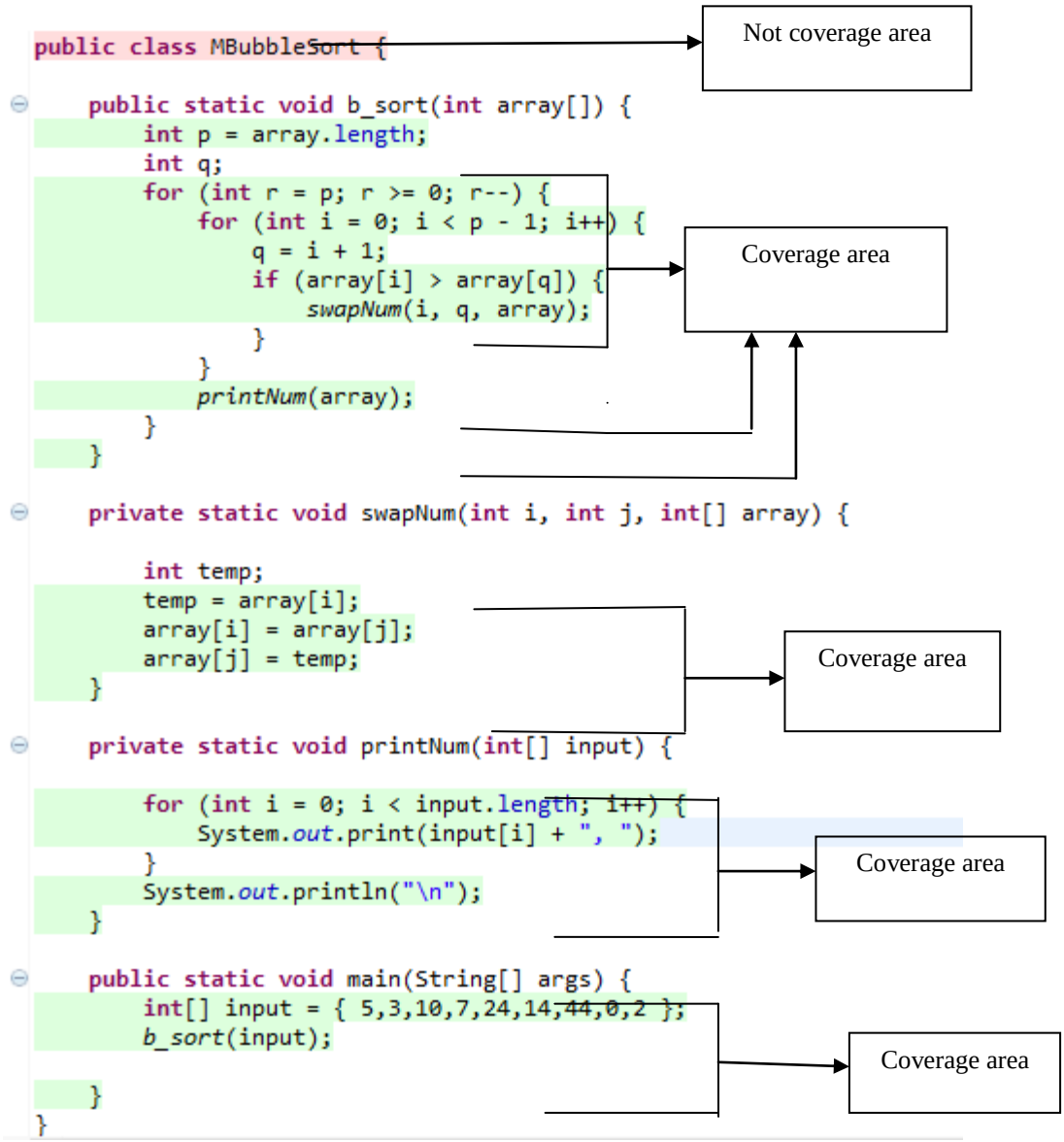


Fig.1 flow chart for Pseudo-code for code coverage

© 2016, IJARCSMS All Rights Reserved

IV. EXPERIMENT AND RESULTS

Program for Bubble Sort




```
public class MBubbleSort {

    public static void b_sort(int array[]) {
        int p = array.length;
        int q;
        for (int r = p; r >= 0; r--) {
            for (int i = 0; i < p - 1; i++) {
                q = i + 1;
                if (array[i] > array[q]) {
                    swapNum(i, q, array);
                }
            }
            printNum(array);
        }
    }

    private static void swapNum(int i, int j, int[] array) {

        int temp;
        temp = array[i];
        array[i] = array[j];
        array[j] = temp;
    }

    private static void printNum(int[] input) {

        for (int i = 0; i < input.length; i++) {
            System.out.print(input[i] + ", ");
        }
        System.out.println("\n");
    }

    public static void main(String[] args) {
        int[] input = { 5,3,10,7,24,14,44,0,2 };
        b_sort(input);

    }
}
```

Fig.2 Snapshots for bubble sort program

```
3, 5, 7, 10, 14, 24, 0, 2, 44,

3, 5, 7, 10, 14, 0, 2, 24, 44,

3, 5, 7, 10, 0, 2, 14, 24, 44,

3, 5, 7, 0, 2, 10, 14, 24, 44,

3, 5, 0, 2, 7, 10, 14, 24, 44,

3, 0, 2, 5, 7, 10, 14, 24, 44,

0, 2, 3, 5, 7, 10, 14, 24, 44,

0, 2, 3, 5, 7, 10, 14, 24, 44,

0, 2, 3, 5, 7, 10, 14, 24, 44,

0, 2, 3, 5, 7, 10, 14, 24, 44,
```

Fig. 3 Output of the program

© 2016, IJARCSMS All Rights Reserved

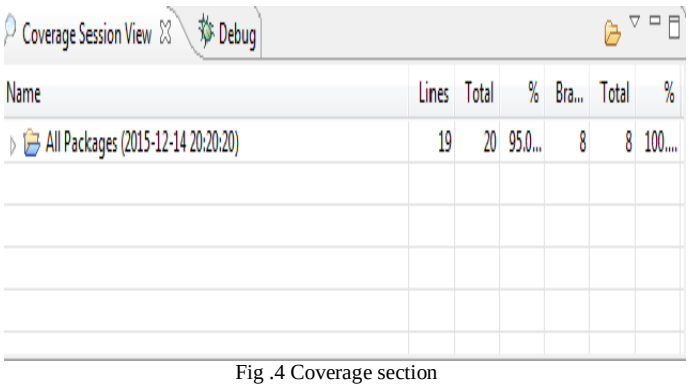


Fig .4 Coverage section

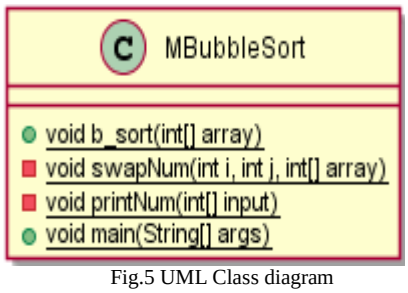


Fig.5 UML Class diagram

Used Methods in Bubble Sort

The Bubble sort program consists of following 4 methods. The details are given below.

1. b_sort (int[]):void

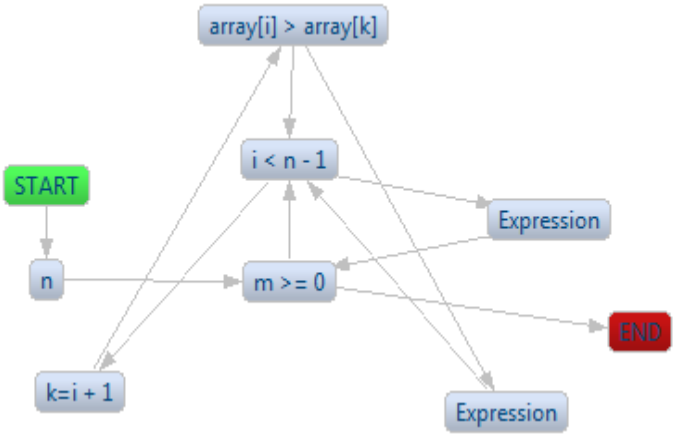


Fig.6 CFG For b_sort(int[]):void

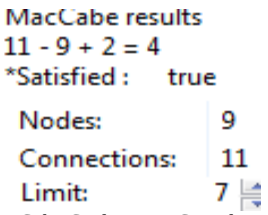


Fig.7 mcCabe Cyclomatic Complexity for b_sort(int[]):void

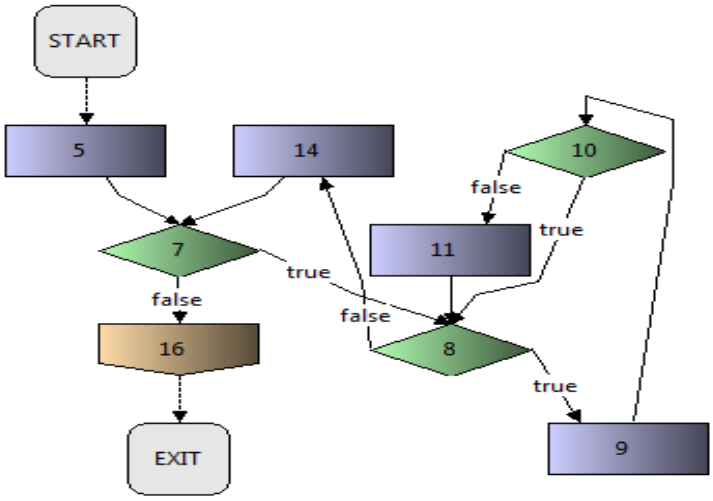


Fig.8 source code Graph For b_sort(int[]):void

© 2016, IJARCSMS All Rights Reserved

2. swapNum(int, int, int[])

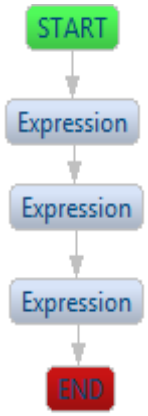


Fig.9 CFG for swapNum(int,int,int[])

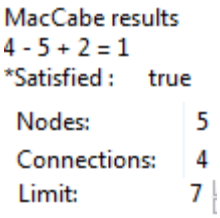


Fig.10 mcCabe Cyclomatic for swapNum(int, int, int[])

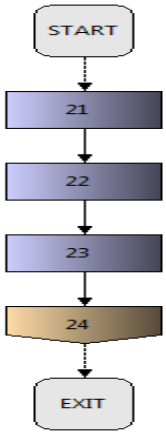


Fig.11 source code graph for swapNum(int,int,int[])

3. printNum(int[])

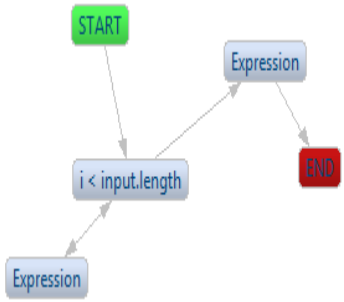


Fig .12 CFG for printNum(int[])

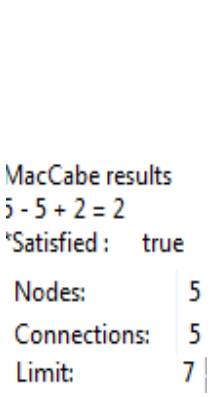


Fig .13 McCabe Cyclomatic Complexity for printNum(int[])

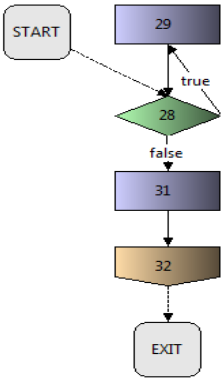


Fig.14 source code Graph for printNum(int[])

4. main (String[])

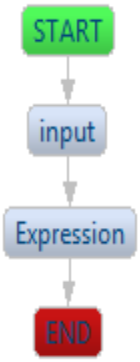


Fig.15 CFG for main(String[])

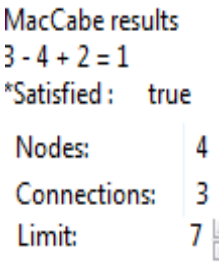


Fig.16 mcCabe Cyclomatic Complexity for main(String[])

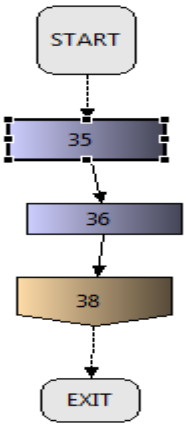


Fig.17 source code Graph for main(String[])

© 2016, IJARCSMS All Rights Reserved

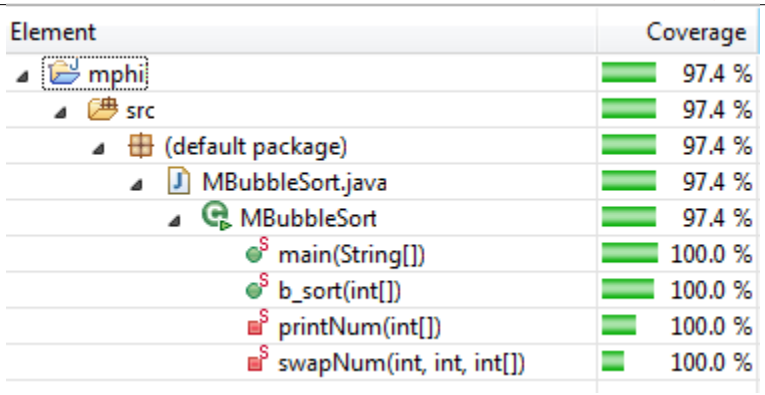


Fig.18 coverage section of the program using EclEmma tool

a) RESULTS OF COMPARISON:

Like bubble sort other sorting programs (Selection sort, Quick sort, Insertion sort, Heap sort, Merge sort) are taken as input as both the tools eCobertura and EclEmma and found the following results.

Table1: comparison of sorting programs basing on coverage metrics using eCobertura tool

programs	Line	Total	% of line coverage	Branch	Total	% of Branch coverage
Bubble sort	19	20	95%	8	8	100%
Selection sort	15	16	93.75%	8	8	100%
Quick sort	34	35	97.14%	15	18	83.33%
Insertion sort	13	14	92.86%	8	8	100%
Heap sort	32	33	96.97%	16	16	100%
Merge sort	37	37	100%	14	14	100%

According to the percentage of line coverage and branch coverage, different types of sorting are arranged in descending order as follows.

According to Line coverage

Merge sort>quick sort>heap sort>bubble sort>selection sort>insertion sort

According to branch coverage

Merge sort=heap sort= bubble sort= selection sort= insertion sort >quick sort

Table: comparison of sorting programs basing on coverage metrics using EcElemma tool

Programs	TCP % of coverage
Bubble sort	97.4%
Selection sort	97.3%
Quick sort	99.4%
Insertion sort	97.2%
Heap sort	98.2%
Merge sort	100%

According to the total percentage of line coverage and branch coverage, different types of sorting are arranged in descending order as follows.

Merge sort>quick sort>heap sort>bubble sort> selection sort> insertion sort

© 2016, IJARCSMS All Rights Reserved

b) RESULTS:

According to the code coverage basing on the metrics line and branch the order of different sorting programs is as follows

Merge sort>quick sort>heap sort>bubble sort >selection sort>insertion sort.

Hence merge sort is the better sorting algorithm according to code coverage analysis.

V. CONCLUSION AND FUTURE WORK

Different categories of testing are carried to generate test cases. But a good automated testing tool is one which generates large number of test cases which covers maximum portion of code. More research is going on to increase test cases which are vital requirement for testing. In current paper two open source tools are used that are eCobertura and EclEmma for finding efficient code coverage.

From the experiment of this paper it is concluded that merge sort is a better algorithm for sorting using code coverage. It is also concluded that both the tools are useful for easy identification and which part of code is useful to generate test cases. In future the research work can be extended to cover other types of code coverage metrics like path, block and the Modified Condition/Decision Coverage (MC/DC) which are needed for multithreading programming in java.

References

1. www.mil-embedded.com/guest-bolgopost.
2. B. Beizer, “Software testing techniques,” 2nd edition. New York: Van Nostrand Reinhold, 1990.
3. J.J. Li, D.Weiss, and H. Yee, “Code-coverage guided prioritized test generation,” Inf. Softw. Technol., 48, 2006, pp. 1187-1198.
4. M. H. Chen, M. R. Lyu, and W. E. Wong, "An empirical study of the correlation between code coverage and reliability estimation," In Proceedings of the 3rd International Software Metrics Symposium, 25-26 Mar. pp. 133-141,1996.
5. Khalid Alemerien Et Al “Examining The Effectiveness Of Testing Coverage Tools: An Empirical Study” International Journal Of Software Engineering And Its Applications Vol.8, No.5 ,2014 .
6. A.M.R. Vincenzia,et al.” Coverage testing of Java programs and components”Science of Computer Programming 56, 2005.
7. H.singh et al. “Code Coverage Analysis of Object-Oriented Programming” Bachelor of Technology thesis, Department of Computer Science and Engineering National Institute of Technology Rourkela,2011.
8. E. Kajo-Mece and M. Tartari, “An Evaluation of Java Code Coverage Testing Tools”, Local Proceedings © 2012 Faculty of Sciences, University of Novi Sad, BCI’12, Novi Sad, Serbia, (2012), September 16-20.
9. L. S. Priya, A. Askarunisa and N. Ramaraj, “Measuring the Effectiveness of Open Coverage Based Testing Tools”, Journal of Theoretical and Applied Information Technology, vol. 5, no. 5, (2005), pp. 499-514.
10. Q. Yang, J. J. Li and D. Weiss, “A survey of coverage based testing tools”, First International Workshop on Automation of Software Testing, AST’06, ACM, (2006), pp. 99-103.
11. M. Shahid and S. Ibrahim, “An Evaluation of Test Coverage Tools in Software Testing”, Proceedings of International Conference on Telecommunication Technology and Applications (CSIT), vol. 5, (2011).
12. J. Hofer, “Ecobertura” http://ecobertura.johoop.de/, retrieved on February 2013.
13. EclEmma. Available at: http://www.eclemma.org/index.html.
14. http://www.eclipse.org/eclipse/development/readme_eclipse_3.8.php.
15. http://www.graphviz.org/content/how-create-autometic-graph.
16. http://conflence.atlassian.com/pages/viewpage.action?pageId=79986990.

© 2016, IJARCSMS All Rights Reserved