Journal of Robotics, Networking and Artificial Life, Vol. 2, No. 2 (September 2015), 89-93

Code Coverage Visualization on Web-Based Testing Tool for Java Programs

Mochamad Chandra Saputra[*], Tetsuro Katayama[†]
[*]*Universitas Brawijaya, Jl. Veteran, Malang 65145, Indonesia*
[†]*University of Miyazaki, 1-1 Gakuen-kibanadai nishi, Miyazaki, 889-2192 Japan*
E-mail: andra@ub.ac.id, kat@cs.miyazaki-u.ac.jp

Abstract

The visualization of statement coverage (C0) and branch coverage (C1) measurement output can be used in several ways to improve the verification and validation process. The result displays are percentage of a successful tested code and visual information with highlighted in bright green as information of executed lines, bright yellow for statement coverage and dark green The web-based testing tool significantly reduces the time for testing the code and help user to understand the behavior of the tested code.

Keywords: Visualization on software testing, Code coverage, Web-based testing tool, Java

1. Introduction

There are many ways to measure software development incorrectly. Testing can be the process of validating and verifying the software product to ensure the business and technical requirements to work as expected.[1]

A common way to evaluate tests is to measure code coverage. Code coverage helps software engineers to understanding which portion of code has been executed, measure the percentage of source code executed during the run and also the software engineering using a given test suite throughout the software testing process. [2]

Since software testing is a long and complex process with probably huge result data collection, visual information will provide testers with a quick and general perspective, which leads to a better understanding of a system's software behavior.[3] Implementing software testing as a web application for visualizing the result of testing is one of the solution to easily understand the behavior of a software code.

The main advantages of adopting the web applications are (1) no installation costs, (2) automatic upgrade with new features for all users, (3) universal access from any machine connected to the Internet, and (4) independence from the operating system.[4]

To display the testing process and to understand the behavior of a code, this research has implemented a code coverage visualization on web-based testing tool for java programs. The testing process is shown the executed each line of tested code and calculation of the lines that executed several times using statement coverage.

2. Specifications and Implementation Policies For The Tool

2.1 Specifications

This research uses statement coverage (C0) and branch coverage (C1).[5]

The testing tool has three parts: uploader code, java service testing, and insertion of temporary database. The java service testing has four sub-parts: analyzer, C0 and C1 instrument code generator, testing part, and random data generator as shown the design system in Fig.1.

To implement this model, several steps are followed.

Published by Atlantis Press

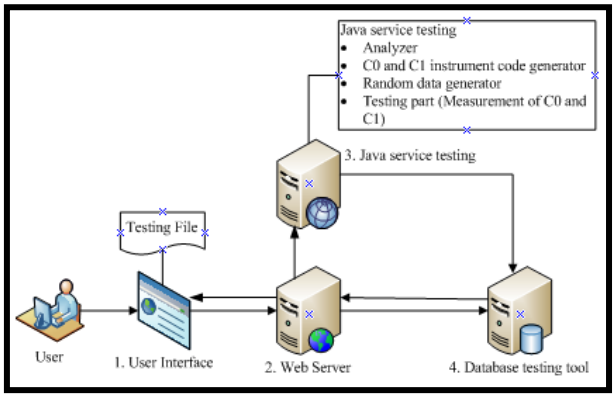

Fig. 1. Design of the testing tool.

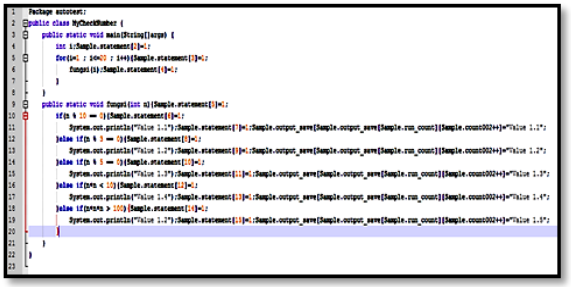

Fig. 2. Example of the C0 and C1 instrumented Code CheckNumber tested code.

1. Uploader code is an input of testing tool.

The input is tested code from a user into the server that used for the java service testing. Tested code is a java program.

2. Java service testing

a) Analyzer loads the original tested code and then the original tested code used by the C0 and C1 instrument code generator, testing part, and random data test generator. The testing tool will execute the java service testing to analyze and read the original code based on the information on the specified file, then testing the code and inserting it into a temporary database for javascript visualization.

b) The C0 and C1 instrumented code generator generates a C0 and C1 instrumented code. It is inserted or rewritten instrument code at each line of the original code, and it is used for calculating the number of executions of C0 and C1.

c) Testing part views the covering status of statements and branches by inputting random data during the background process. Java service testing finds the class name of the original code by pattern matching. The class name is used when the C0 and C1 instrumented code generator generates the instrumented code.

The testing method process will insert the data execution line by line into the database. Data stored in the database are the line number, number of executions of each line, and tested code.

When the java service testing executes a testing, the service assigns 1 to an element of the array that corresponds to the executed statement. When all elements of the array C0 are assigned 1, the java service testing judges C0 satisfies 100% and also for C1.

d) Random data generator generates random test data. Users of the testing tool do not need to describe the test data. The random data generator starts after generating the C0 and C1 instrumented code. The testing part executes the C0 and C1 instrumented code. The random data generator inputs random data into the C0 and C1 instrumented code on behalf of the users inputting data per standard input instructions.

After each execution of the C0 and C1 instrumented code by the testing part, the testing tool obtains the covering status of statements and measures C0 and C1. The testing tool visualizes the covering status of the statements by highlighting the original code that is displayed and animated as the sequence process executes the tested code.

2.2 Implementation

This research implements the web-based software testing tool of an automatic unit testing tool using random testing for java programs. This testing tool can automatically test a program based on statement coverage (C0) and branch coverage (C1), without preparing test data by user. As an example of the tested code is Class CheckNumber. Fig.2 shows the tested code with C0 and C1 instrumented Code. To test the code, the following steps are used to generate the code:

- Insert a package before the first line of Fig.2, to generate the C0 and C1 instrumented code.

Fig. 3. Static display for investigation of the CheckNumber tested code

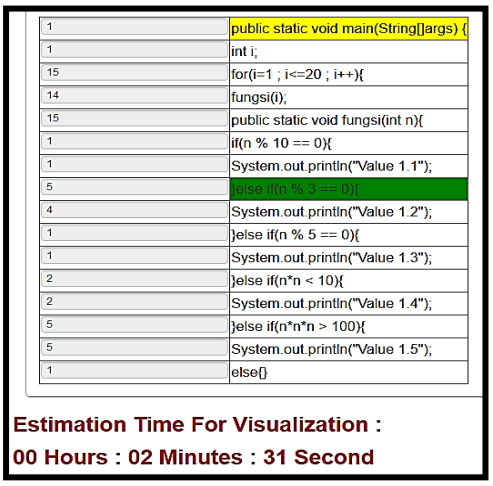

Fig. 4. Dynamic display for investigation of the CheckNumber tested code

- Rewrite an original class name as a class name "MyCheckNumber" specified in advance by the testing part.
- Insert an assignment statement after all statements to gain the covering status of the statements.
- Insert an assignment statement to store outputs after standard input instruction "System.out.println".
- Random data are used for data test the code and then automatically tested for the CheckNumber2 using C0 and C1 instrumented code to verify the branch condition. Each data testing process are insert into the database using for visual information.

Code instrumentation in this research consists of inserting some additional codes to measure coverage results. Instrumentation can be done at the source level in a separate pre-processing phase with pattern matching or at runtime by measure of coverage result. Data gathering consists of storing coverage data collected during test runtime.

Random data tests provide the application under testing with input data generated at random. Typically, testers pay no attention to expected data types.[6] The type of random data used in the testing tool is integer only.

The testing tool has two result displays. The first is a static display as shown in Fig.3. The testing tool displays the static result of testing as the number of each line execution, measurement percentage of success from the statement and branch coverage, and time execution for testing as how many times the line was executed by the java service testing.

The second result display is dynamic with Ajax. The result display visualizes the behavior of the tested code as shown in Fig.4. Certain coverage analysis tools also depict coverage visually, often by highlighting portions of code that are unexecuted by a test suite.[7] In this research, the visual information resets every time a tester select a new code and then tests the code. The testing tool performs new visualizations to know the behavior of the code, and that it does not accumulate with each successive test run before the testing.

3. Discussion

The testing tool can show the correlation between visual information and software testing. This correlation means results collection and a better perspective of software testing. The testing tool shows the correlation as visual information, and it allows a better understanding of the behavior of the tested code.

Visual information describes the behavior of the tested code as a sequence of the line executed by the testing tools. Visual information helps to understand the behavior of the tested code. The result displays for visual information have been highlighted in bright green

Table 1. Time comparison between using the testing tools and manual testing (by human)

Testing Tool	No.	Manual Testing
716 ms	1	3' 54 ″
	2	2' 54 ″
	3	5' 40 ″
	Average	4' 15 ″

as information on executed lines, bright yellow for statement coverage and dark green for branch coverage. Visual information describes the behavior of the tested code as a sequence of the line being executed.

Code coverage visualizations are supposed to improve developer efficiency and knowledge and promote more productive testing strategies. The research for the visualization leads developers toward a better standard of test effectiveness.

The testing tool uses java file CheckNumber that inputs 19 lines and then to measure statement coverage, branch coverage, number of runs and the input of each program to the end of testing by C0 and C1 to reach 100%. The testing measured at the web server with CentOS release 5.9 (Final), Apache/2.2.3, Intel(R) Xeon(R) CPU 3050 @2.13GHz, PHP Version 5.3.3.

The times execution for testing the class CheckNumber is 716 ms and if we test manually (by humans), the average time is 4 minutes 15 second as shown in Table 1. The testing tool can reduce time to describe a tested code and execute unit testing in a shorter time.

Visualization concerns the graphical representation of information to assist human comprehension of and reasoning about that information.[8] The testing tool result makes possible distribution of the software testing scalability problem, making certain key choices instead a technical distribution of responsibilities.

4. Conclusion

To improve the efficiency of testing in software development, this research has implemented a web-based software testing tool with java service testing of an automatic unit testing tool for java programs with random testing. The implemented testing tool generates the C0 and C1 instrumented code from the original code. The testing tool uses java service testing to automatically test a program by inputting random data into the C0 and C1 instrumented code. After testing, the obtained result is output as a static html page and dynamic display for visual information with Ajax.

The testing tool can show the correlation between visual information and software testing as a result collection and perspective of software testing as a sequence of the line executed by the testing tools. The testing tool can reduce the time needed to describe a tested code and execute unit testing. The time execution needed to test CheckNumber was 716 ms.

Future issues are as follows:

- Expand the type of a data test can input data tests other than type int.
- Measurement with key performance indicators for software development like resources and cost, product size and stability, product quality, process performance, technology effectiveness.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Number 24500044.

References

1. Sivakumar.N.,Vivekanandan.K. "Comparing the Testing Approaches of Traditional, Object-Oriented and Agent-Oriented Software System", *International Journal of Computer Science & Engineering Technology (IJCSET)*, **3**(10) (2012) 498-504.
2. Heed. Per., Westrup. Alexander., Runeson.Per. "Automated Platform Testing Using Input Generation and Code Coverage". Master Thesis Document. Lund University, Faculty of Engineering (2009).
3. Wang.Huansong., Zhang.Xiang., Zhou.Mingqi . "MaVis: Feature-based Defects Visualization in Software Testing", *Engineering and Technology (S-CET) 2012 Spring Congress on* (2012) 1-4.
4. Dogana.Serdar., Betin-Cana.Aysu., Garousia.Vahid. "Web application testing: A systematic literature review", *The Journal of Systems and Software (ELSEVIER)*, (2014) 174-201.
5. ISTBQ. "How to Calculate Statement, Branch/Decision and Path Coverage for ISTQB Exam Purpose". http://www.ajoysingha.info

6. Agarwal,B. B., Tayal, S. P., Gupta, M. “Software Engineering & Testing An Introduction”, *Jones and Bartlett Publishers*, (2010) 161-179.
7. Lawrance, Joseph., Clarke, Steven., Burnett, Margaret., Rothermel, Gregg. “How Well Do Professional Developers Test with Code Coverage Visualizations? An Empirical Study”, *IEEE Computer Society*, (2005) 53-60.
8. Petre, Marian.,Quincey, Ed de. “A gentle Overview of Software Visualisation”, *Computer Society of India Communications*, (2006) 1- 10.