Journal of Robotics, Networking and Artificial Life
Vol. **8**(**1**); June (2021), *pp.* 1–5
DOI: https://doi.org/10.2991/jrnal.k.210521.001; ISSN 2405-9021; eISSN 2352-6386
https://www.atlantis-press.com/journals/jrnal

Research Article

Measuring Redundancy Score for Test Suite Evaluation by Using Test Cases Matching Approach

Mochamad Chandra Saputra[1,*], Tetsuro Katayama[1], Yoshihiro Kita[2], Hisaaki Yamaba[1], Kentaro Aburada[1], Naonobu Okazaki[1]

[1]*Interdisciplinary Graduate School of Agriculture and Engineering, University of Miyazaki, 1-1 Gakuen-Kibanadai Nishi, Miyazaki 889-2192, Japan*
[2]*Department of Information Security, Faculty of Information Systems, Siebold Campus, University of Nagasaki, 1-1-1 Manabino, Nagayo-cho, Nishi-Sonogi-gun, Nagasaki 851-2195, Japan*

ARTICLE INFO

Article History

Received 10 November 2019
Accepted 27 January 2021

Keywords

Redundancy score
test suite evaluation
test case matching
redundant test cases

ABSTRACT

Evaluating a test suite that contains redundant test cases is necessary to reduce the cost of testing. The redundant test cases exist when both of the two test cases are executed the same lines of code. This research evaluates the test suite by identifying redundant test cases. Exact match approach is used to investigate the redundant test cases in the test suite. The redundancy score is defined by redundancy formula which is calculated by dividing the number of redundant test cases by numbers of test cases in a test suite. The experiment uses two Java programs. The redundancy scores of the two test suites from each program are 0.37 and 0.67, respectively. It means 37% and 67% redundant test cases are included in the test suites. The redundancy score provides useful information to improve the efficiency of software testing, especially in testing other programs by reusing the same test suite such as regression testing and automated testing.

1. INTRODUCTION

Software contains a series of instructions that execute several tasks to achieve their objective. From a software developer's point of view, instructions are defined as a program. Testing is important for assessing whether the program behaves as users require. Testing helps to examine the source code and to ensure that the release version of the software is stable.

Institute of Electrical and Electronics Engineer has defined a test case as a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement [1]. Executing the test case to the program to ensure individual components are tested and operate correctly [2]. The programmer uses unit testing to test individual program unit, such as procedures, functions, methods, or classes.

Evaluating the test cases in test suite is necessary to reduce the cost of testing [3]. Cost of testing contains the cost of designing, maintaining, and executing test cases [4]. Test suite may contain redundant, ambiguous, vague, and unfit test cases [5]. Evaluating a test suite in this research is to find redundant test cases that should be eliminated to reduce the cost of testing. Testing a software system contains executing various sets of test cases that examine several parts of the source code. The test cases are redundant when the lines of code executed by a test case are similar to another test case in a test suite.

The purpose of this research is to evaluate the test suite by identifying redundant test cases. This research compares the lines of code executed among the test cases. The redundancy score is defined by redundancy formula. This paper is organized on calculating the redundancy score helps you to reduce the cost of testing. The rest of the paper is organized as follows. Section 2 describes the methodology of this research to find redundant test cases using the exact match approach. Section 3 describes the experimental activity and its result. Section 4 describes the result and discussion of the research. Section 5 describes the conclusion and future work of the research.

2. METHODOLOGY

One of the testing techniques is structural testing, known as white-box testing. The white-box testing examines the source code, specifically concerning the internal structure of the source code. One of the objectives of the test case is to gain the code coverage information as a kind of valuable information in the white-box testing [6]. The test case coverage needs to confirm which line of code is executed in testing [7].

The approach in this research is illustrated in Figure 1. The first process in this research is finding the test case coverage information from the test case. The test case coverage information is the

*Corresponding author Email: chandra@earth.cs.miyazaki-u.ac.jp

information of lines of code executed by the test case. The test case coverage information is examined with exact match approach to identify redundant test cases. The redundant test cases exist when both of the two test cases execute the same lines of code. Two test cases are not categorized as redundant test cases if they have different lines of code executed even one line of code.

The exact match approach compares the lines of code executed by the test cases in the test suite. To compare them, flags are introduced. When the value of the flag is 1, it means the line is executed. And, 0 means not executed. Table 1 shows an example of an exact match of test case coverage information. The result is whether all the lines of code executed from test case coverage information identically. The unmatch result is when the lines of code executed from test case have differences, even one line.

Redundant test cases in the test suite may increase the cost of executing the test suite in testing. It is important to identify redundant test cases in the test suite and then to measure the redundancy score to reduce the cost of executing the test suite. The redundancy score is calculated by dividing the number of redundant test cases in the test suite by the number of test cases. The formula for calculating the redundancy score is as follows:

$$\text{Redundancy score} = \frac{\sum \text{Redundant test cases}}{\sum \text{Test cases}} \quad (1)$$

Its range is from 0 to 1. It represents the degree of redundancy as the ratio of redundant test cases in the test suite. The research uses the test case coverage information. Here, Java code coverage library (JaCoCo) is used to generate reports on the lines of code executed by the test cases [8].

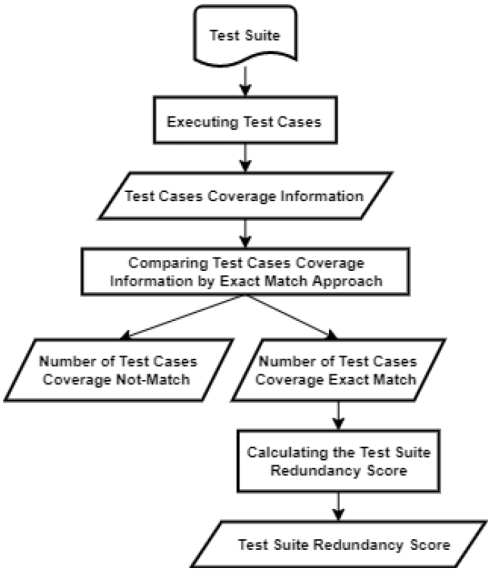

Figure 1 | The flow of test suite redundancy calculation approach.

Table 1 | Example of exact match of test case coverage information

LOC	1	2	3	4	5	6	7	8	9	Result
TC-1	0	1	1	1	1	0	0	1	1	Unmatch
TC-2	0	1	1	0	0	0	0	1	1	
TC-1	0	1	1	1	1	0	0	1	1	Match
TC-3	0	1	1	1	1	0	0	1	1	

3. THE EXPERIMENT

The experiment has two java source codes: *student grades* and *quadratic function* as shown in Tables 2 and 3. And, test suites of each program are prepared. Table 4 shows the sample test suites of them. They consist of eight test cases for *student grades* and nine test cases for *quadratic function*, respectively.

Table 2 | Student grade

LOC	Statement
1	input = new Scanner(System.in);
2	System.out.println(" Please Enter you Score: ");
3	score = input.nextInt();
4	if(score>=90 && score<=100){
5	System.out.println("Your Grade is A");
6	} else if (score<90 && score>=80){
7	System.out.println("Your Grade is B");
8	} else if (score <80 && score >=70){
9	System.out.println("Your Grade is C");
10	} else if (score <70 && score >=60){
11	System.out.println("Your Grade is D");
12	} else {
13	System.out.println("You Faild on this class, try next year");}
14	}

Table 3 | Quadratic function

LOC	Statement
1	Scanner input = new Scanner(System.in);
2	System.out.print("Input a: ");
3	double a = input.nextDouble();
4	System.out.print("Input b: ");
5	double b = input.nextDouble();
6	System.out.print("Input c: ");
7	double c = input.nextDouble();
8	double result = b * b 4.0 * a * c;
9	if (result > 0.0) {
10	double r1 = (b + Math.pow(result, 0.5)) / (2.0 * a);
11	double r2 = (b Math.pow(result, 0.5)) / (2.0 * a);
12	System.out.println("The roots are " + r1 + " and " + r2);
13	} else if (result == 0.0) {
14	double r1 = b / (2.0 * a);
15	System.out.println("The root is " + r1);
16	} else {
17	System.out.println("The equation has no real roots.");
18	}

Table 4 | Sample test suites of student grades and quadratic function suite

Student grades		Quadratic function	
Test case	Input data	Test case	Input data
TC-1	100	TC-1	a:1, b:–5, c:6
TC-2	15	TC-2	a:1, b:–2, c:1
TC-3	90	TC-3	a:1, b:1, c:1
TC-4	70	TC-4	a:1, b:–4, c:4
TC-5	65	TC-5	a:1, b:–6, c:25
TC-6	0	TC-6	a:0;b:2, c:4
TC-7	–1	TC-7	a:1;b:–4, c:-5
TC-8	85	TC-8	a:1;b:–3, c:3
–	–	TC-9	a:50;b:100, c:50

The experiment uses the test case coverage information that defined as lines of code are executed by the test cases. Tables 5 and 6 show the result of the test case coverage information on *student grades* and *quadratic function*, respectively. When the value is 1, it means the line is executed. And, 0 means not executed.

Tables 7 and 8 show the result of the test case matching of *student grades* and for *quadratic function*, respectively. Table 9 shows the result of the exact match for *student grades* and *quadratic function*. The number of redundant test cases is used to calculate the redundancy score by formula (1).

Table 5 | The result of the test case coverage information on student grades

Test case name	Test case coverage information
TC-1	1,1,1,1,1,0,0,0,0,0,0,0,0,1
TC-2	1,1,1,1,0,1,0,1,0,1,0,1,1,1
TC-3	1,1,1,1,1,0,0,0,0,0,0,0,0,1
TC-4	1,1,1,1,0,1,0,1,1,0,0,0,0,1
TC-5	1,1,1,1,0,1,0,1,0,1,1,0,0,1
TC-6	1,1,1,1,0,1,0,1,0,1,0,1,1,1
TC-7	1,1,1,1,0,1,0,1,0,1,0,1,1,1
TC-8	1,1,1,1,0,1,1,0,0,0,0,0,0,1

Table 6 | The result of the test case coverage information on quadratic function

Test case name	Test case coverage information
TC-1	1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,0,0,1
TC-2	1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1,0,1
TC-3	1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,1,1,1
TC-4	1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1,0,1
TC-5	1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1
TC-6	1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,0,0,1
TC-7	1,1,1,1,1,1,1,1,1,1,1,1,1,0,1,0,0,0,1
TC-8	1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,1,1,1
TC-9	1,1,1,1,1,1,1,1,1,1,0,0,0,1,1,1,1,0,1

Table 9 | The result of the exact match

Student grades	Quadratic function
TC-1, TC-3	TC-1,TC-6 and TC-7
TC-2, TC-6, TC7	TC-2, TC-4 and TC-9
–	TC-3, TC-5 and TC-8

4. RESULTS AND DISCUSSION

The redundant test cases are investigated by exact match approach among test cases in the test suite. The result from exact match approach can find the same test case coverage information. In the results of exact match approach between test cases in the test suite, it was found that several test cases were identical, which means that several test cases exactly match to others. By detecting the test cases redundancy, redundant test cases are eliminated and the cost of testing is reduced.

In the experiment, the redundant test cases are found on *student grades* and *quadratic function*. Based on the result of exact match in Table 9, the number of redundant test cases for *student grades* is three test cases. The result of the exact match for its test suite is that (TC-1 and TC-3) and (TC-2, TC-6, and TC7) exactly match. The number of redundant test cases for *quadratic function* is six test cases. The result of the exact match for its test suite is that (TC-1, TC-6, and TC-7), (TC-2, TC-4, and TC-9), and (TC-3, TC-5, and TC-8) exactly match. The test suite only needs one test case for each group of redundant test cases and they should be eliminated.

This research calculates the redundancy score by using formula (1). The redundancy score of the test suite for *student grades* is 0.37, and *quadratic function* is 0.67, respectively. It means 37% and 67% redundant test cases are included in the test suites.

Hence, we can evaluate the test suite by identifying redundant test cases. The redundancy score provides useful information to improve the efficiency of software testing, especially in testing

Table 7 | The result of the test case matching of student grades

	TC-1	TC-2	TC-3	TC-4	TC-5	TC-6	TC-7	TC-8
TC-1	–	Unmatch	Match	Unmatch	Unmatch	Unmatch	Unmatch	Unmatch
TC-2	Unmatch	–	Unmatch	Unmatch	Unmatch	Match	Match	Unmatch
TC-3	Match	Unmatch	–	Unmatch	Unmatch	Unmatch	Unmatch	Unmatch
TC-4	Unmatch	Unmatch	Unmatch	–	Unmatch	Unmatch	Unmatch	Unmatch
TC-5	Unmatch	Unmatch	Unmatch	Unmatch	–	Unmatch	Unmatch	Unmatch
TC-6	Unmatch	Match	Unmatch	Unmatch	Unmatch	–	Unmatch	Unmatch
TC-7	Unmatch	Match	Unmatch	Unmatch	Unmatch	Unmatch	–	Unmatch
TC-8	Unmatch	Unmatch	Unmatch	Unmatch	Unmatch	Unmatch	Unmatch	–

Table 8 | The result of the test case matching of quadratic function

	TC-1	TC-2	TC-3	TC-4	TC-5	TC-6	TC-7	TC-8	TC-9
TC-1	–	Unmatch	Unmatch	Unmatch	Unmatch	Match	Match	Unmatch	Unmatch
TC-2	Unmatch	–	Unmatch	Match	Unmatch	Unmatch	Unmatch	Unmatch	Match
TC-3	Unmatch	Unmatch	–	Unmatch	Match	Unmatch	Unmatch	Match	Unmatch
TC-4	Unmatch	Match	Unmatch	–	Unmatch	Unmatch	Unmatch	Unmatch	Unmatch
TC-5	Unmatch	Unmatch	Match	Unmatch	–	Unmatch	Unmatch	Unmatch	Unmatch
TC-6	Match	Unmatch	Unmatch	Unmatch	Unmatch	–	Unmatch	Unmatch	Unmatch
TC-7	Match	Unmatch	Unmatch	Unmatch	Unmatch	Unmatch	–	Unmatch	Unmatch
TC-8	Unmatch	Unmatch	Match	Unmatch	Unmatch	Unmatch	Unmatch	–	Unmatch
TC-9	Unmatch	Match	Unmatch	Unmatch	Unmatch	Unmatch	Unmatch	Unmatch	–

other programs by reusing the same test suite such as regression testing and automated testing.

5. CONCLUSION

The research confirms that test suites can be evaluated by the redundancy score which is calculated by the result from exact match approach among the test case coverage information in the test suite. The current research identified the redundant test cases on the result from exact match approach.

The experiment has two java source codes: *student grades* and *quadratic function*. And, each test suite is prepared. The redundancy score of *student grades* is 0.37 and of *quadratic function* is 0.67, respectively. It can represent the percentage of redundant test cases in the test suite. It means 37% and 67% redundant test cases are included in the test suites.

Hence, we can evaluate the test suite by identifying redundant test cases. The redundancy score provides useful information to improve the efficiency of software testing, especially in testing other programs by reusing the same test suite such as regression testing and automated testing.

Future research is needed to consider other evaluation methods of a test suite, for example, relation between the test case redundancy and path coverage, reusability of test cases or test suite, and so on.

CONFLICTS OF INTEREST

The authors declare they have no conflicts of interest.

REFERENCES

[1] The Institute of Electrical and Electronics Engineers (IEEE), 610-1990 - IEEE standard computer dictionary: a compilation of IEEE standard computer glossaries, IEEE, 1990.
[2] B.B. Agarwal, S.P. Tayal, M. Gupta, Software engineering & testing: an introduction, Jones and Bartlett Publisher, USA, 2010.
[3] A. Farooq, R.R. Dumke, Evaluation approaches in software testing, Technical Report, Nr.: FIN-05-2008, 2008.
[4] K. Naik, P. Tripathy, Software testing and quality assurance. Theory and practice, vol. 1, John Wiley & Sons, Inc., Hoboken, NJ, USA, 2008.
[5] H. Mohanty, J.R. Mohanty, A. Balakrishnan, Trends in software testing, Springer Singapore, Singapore, 2017.
[6] P. Heed, A. Westrup, Automated platform testing using input generation and code coverage, Technical Report, Department of Computer Science, Lund University, Lund, Sweden, 2009.
[7] M.C. Saputra, T. Katayama, Code coverage visualization on web-based testing tool for java programs, J. Robot. Netw. Artif. Life 2 (2015), 89–93.
[8] "EclEmma - Java Code Coverage for Eclipse." [Online]. Available from: https://www.jacoco.org/ (accessed November 5, 2020).

AUTHORS INTRODUCTION

Mr. Mochamad Chandra Saputra

He received the Master's degree from the University of Miyazaki, Japan, and Brawijaya University, Indonesia on Double Degree Program in 2014. He also worked in Brawijaya University ICT Unit as System Analyst from 2006 to 2014. Since 2015, he has been a lecturer on the Faculty of Computer Science, Brawijaya University. Currently, he is persuing the Doctoral Study at the University of Miyazaki. His research interests include software testing, software quality, and software project management.

Mr. Yoshihiro Kita

He received a PhD degree in systems engineering from the University of Miyazaki, Japan in 2011. He is currently an Associate Professor with the Faculty of Information Systems, University of Nagasaki, Japan. His research interests include software testing and biometrics authentication.

Mr. Tetsuro Katayama

He received the PhD degree in engineering from Kyushu University, Fukuoka, Japan in 1996. From 1996 to 2000, he has been a Research Associate at the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. Since 2000, he has been an Associate Professor at the Faculty of Engineering, Miyazaki University, Japan. He is currently a Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include software testing and quality. He is a member of the IPSJ, IEICE, and JSSST.

Mr. Hisaaki Yamaba

He received the B.S. and M.S. degrees in chemical engineering from the Tokyo Institute of Technology, Japan, in 1988 and 1990, respectively, and the PhD degree in systems engineering from the University of Miyazaki, Japan, in 2011. He is currently an Assistant Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include network security and user authentication. He is a member of SICE and SCEJ.

Mr. Kentaro Aburada

He received the B.S., M.S., and PhD degrees in computer science and system engineering from the University of Miyazaki, Japan, in 2003, 2005, and 2009, respectively. He is currently an Associate Professor with the Faculty of Engineering, University of Miyazaki, Japan. His research interests include computer network and security. He is a member of IPSJ and IEICE.

Mr. Naonobu Okazaki

He received his B.S., M.S., and PhD degrees in electrical and communication engineering from Tohoku University, Japan, in 1986, 1988, and 1992, respectively. He joined the Information Technology Research and Development Center, Mitsubishi Electric Corporation in 1991. Since 2002, he is a Professor with the Faculty of Engineering, University of Miyazaki. His research interests include mobile network and network security. He is a member of IPSJ, IEICE, and IEEE.