Proposal of a Method to Measure Test Suite Quality Attributes for White-Box Testing

Mochamad Chandra Saputra[1], Tetsuro Katayama[2]
Dept. Materials and Informatics, Interdisciplinary Graduate School of Agriculture and Engineering,
University of Miyazaki, Miyazaki, Japan

***Abstract*—As an important asset in software testing, measuring quality attributes of the test suite is important to describe the quality of software. This research proposes a method to measure the test suite quality attributes for white-box testing. The attributes are usability, efficiency, reliability, functionality, portability, and maintainability that are selected from 28 attributes in software quality. By using the proposed method, the test suite quality attributes are calculated with various results of level of quality. The result of test suite quality attribute measurement then proves the validity of its result by the reliability analysis. It is used Cohen's kappa coefficient to validating the result of test suite quality attributes measurement based on the level of agreement between the result of measurement and expert assessment. Reliability analysis on test suite quality attribute finds the attribute that strongly related based on the minimum percentage of level of agreement value are usability, reliability and functionality. Hence, our proposed method is useful to measure test suite quality attributes.**

***Keywords*—Test case; test suite quality attributes; white-box testing; reliability analysis; software quality**

I. Introduction

The quality of software is confirmed by systematically exercising the software in carefully controlled circumstances especially in testing phase[1]. During the testing, test suite which contains several test cases play a very important role to check various aspects of the software such as actual program structure and the software functions as per the specification[2]. The test cases are usually developed by a set of inputs, execution preconditions, and expected outcomes for a specific objective. Testing in software development is one of the ways to ensure quality of the software.

The main activity of software testing is verification and validation[3]. In software development life cycle, verification and validation aim is to help the software development build software with good quality. Verification ensures the specific function of the software is correctly implemented. Validation ensures the software are suitable to customer requirement. One of the software testing approaches is white-box. The white-box testing approach aims to ensure that the program is successfully tested based on the internal structures of the software[4].

Software quality is defined as the whole of features and characteristics of a product or service that able to satisfy stated or implied user needs[1]. As an important asset in software testing, measuring the quality of a test suite is important to describe quality of the software. Software testing is one of the quality approaches to control the program before its delivery or installation at the user with an acceptable level of quality. Various software quality attributes have been used on software quality models to define the degree of quality. Software quality attributes are multipurpose attributes that mean any area of software development process can use the attributes. Examining code programs by using the test suite is one of the methods to assure their quality. Test suite quality measurement is necessary to gain information on the test suite performance.

The big problem with quality attributes is uncertainty attributes and their measurement for informing the degree of test suite quality. Currently, measuring the attributes of test suite quality is one of the interesting problems in software testing. The aim of test suite quality attributes measurement to provide useful information about the degree of test suite quality.

The objective of this research is to find and propose the test suite quality attributes measurement and then validate its measurement by the reliability analysis. The research concern with quality attributes for test suite in white-box testing. The research provides a questionnaire for the expert to assess the test suite quality attribute based on their experience. The reliability analysis uses Cohen's kappa coefficient approach. Cohen's kappa coefficient is used to analyze the reliability of test suite quality from test suite quality attributes measurement result and expert assessment. The level of agreement is presenting the reliability of the test suite quality attributes measurement with the expert assessment. This research uses only the test suite for white-box testing. The results of the reliability analysis are the test suite quality attributes that have strongly agreed to the quality of the test suite.

The rest of the paper is organized as follows. Section 2 describe the highlight work done by others that somehow ties in with this research. Section 3 describes the principle and formula to measure test suite quality attributes. Section 4 describes the reliability analysis of test suite quality attribute by using Cohen's kappa coefficient. Section 5 describes the research methodology to validate the test suite quality attributes by using the level of agreement between the result of the measurement and expert assessment. Section 6 describes for experimental activity and its result. Section 7 explains the result of the questionnaire to test suite quality attributes measurement. Section 8 describes the conclusion and future work of the research.

II. Related Works

Test suite consists of a set of test scripts or test procedures known as test case to be executed in a specific test run[2]. Test script in test suite is related to the test case that consists of expected results based on the inputs. The difficulties in software testing quality especially in white-box testing approach vary depending on the size and complexity of the program being tested[5]. It was a great idea to measuring the degree of test suite quality by using the attributes from software quality. Several studies have been reported in the scope of quality attributes of test case that focus on increasing the testing effectiveness consider to mutation testing[6]. The usability especially identification error with effective and efficient is important to enhance software quality[7][8]. Efficiency of test suite is related to number of redundant test cases in the test suite and reducing redundant test cases possible to improve the efficiency in testing[9]. Reliability is considered to number of mutants because the result on mutants coverage could be used to find the true reliability of a program[10][11]. Functionality in the testing approach is to ensure the method in the program satisfies functional requirements and assesses the quality itself[12]. The study on test suite reusability is related to portability has been reported that in the test suite reuse effective at discovering and repairing bugs inserted during pragmatic reuse[13]. Reducing number of test cases and ability of the test cases reused to examine another object should be considered to improve the maintainability[14][15].

With respect to previous work, this research analyzes the quality attributes for test suite to ensure the quality of the test suite. As we already introduced, this proposal a method to measure the test suite quality attributes is adopted the quality attributes from software quality which related to test suite, especially in white-box testing approach.

III. Test Suite Quality Attributes

Software quality defines as the degree of software, component, or process to establish the customer requirement under specific conditions[2]. Successful software testing activity is achieved by collaborative activity between testing activity and quality assurance activities[16]. One of the important assets in testing activity is test suite. Software quality has many approaches such as McCall’s Model (1977), Boehm’s Quality Model (1978), ISO 9126 Standard Quality Model (1986), FURPS (1987), FURPS+ (2000), Capability Maturity Model (CMM 1991), Ghezzi Model (1991), IEEE Model (1993), Dromey’s Quality Model (1995), SATC’s Quality Model (1996), Bansiya’s QMOOD Model (2002), Aspect-Oriented Software Quality Model (2006), Component-based Software development Quality Model (2008), DEQUALITE Model (2009), Sehra S. K Model (2011) and SQuaRE’s Model (2011)[17].

Software quality attributes are multipurpose attributes that mean any area of software development process can use the attributes. Examining code programs by using the test suite is one of the methods to assure their quality. The most used attributes on the software quality model are usability, efficiency, reliability, functionality, portability, and maintainability that selected from 28 attributes. The principle of test suite quality attributes on white-box testing in this research is related to the software quality principle. The research proposes the following formula and definition for test suite quality attributes. To simplify the formula, the research uses the following notation.

• SRTC	: Successful Reused Test Cases
• DCC	: Distinct Code Coverage
• OT	: Objects Tested
• NOLOC	: Number of Original Line of code
• NOTC	: Number of Test Cases
• NOMut	: Number of Mutants
• NOMutK	: Number of Mutants Killed
• NOR	: Number of Redundant Test Cases
• NOMet	: Number of Method
• NOMetExec	: Number of Method Executed

The parameters for measuring the test suite quality attributes are gathering from the test suite examination that enhances the accuracy of test suite quality measurement. The result of test suite quality attribute measurement is numerical which ranges from 0 to 1. The experiment assumes that the test suite quality attribute has three levels of quality such as low, medium, and high. The result of test suite quality attributes divided into those three levels which begin from 0 – 0.33 for low quality, 0.34 – 0.66 for medium quality, and 0.67 – 1 for high quality. One of the criteria of good test cases in the test suite related to white box testing is that the test cases can achieve 100% code coverage. The test suite quality attribute measurement additionally considers code coverage on its measurement.

1) Usability as test suite quality attribute: Usability defines as the degree of a program able to be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use[18]. The research assumes that the test suite usability should consider for the effectiveness and efficiency related to previous definition. In other words, test suite usability defines as the extent to test suite is successfully examine program by the software tester to guarantee that all statements have been exercised at least once and validate the internal data structure with effectiveness and efficiency. By using the notation, the formula for test suite usability as follows.

$$Test\ Suite\ Usability = \frac{(1-(\mathrm{NOR}/\mathrm{NOTC}))+(DCC/NOLOC)}{2} \qquad (1)$$

2) Efficiency as test suite quality attribute: Efficiency defines as the capability of the software product to provide appropriate performance, relative to the number of resources used under stated conditions[2]. Number of resources in the case of test suite is related to number of test cases. Redundant test cases in the test suite are one of the problems that can reduce efficiency value. The previous research conducted the identification and elimination process of redundant test cases in the test suite[19]. The efficiency in this research is related to the degree of redundancy of test cases on the test suite. The test suite efficiency defines as the level of test suite redundancy to complete a certain task. The redundant test cases exist when both of the two test cases are executed in the same lines of code. By using the notation, the formula for test suite efficiency as follows.

$$Test\ Suite\ Efficiency = \left(1 - (NOR/NOTC)\right) \tag{2}$$

3) Reliability as test suite quality attribute: Reliability defines as the ability of the software to operating required functions in specific conditions and time, or number of operations[2]. One of the causes of the inability of the software product to perform a required function is mutant. Mutants define as changed/mutated statements of the source code. The capability of the test cases kills the mutants to ensure the quality of test cases in terms of reliability. Test suite reliability is defined as the probability of test cases in the test suite killed the mutants in testing that consider to its coverage. By using the notation, the formula for test suite reliability as follows.

$$Test\ Suite\ Reliability = \frac{NOMut}{NOMutK} \tag{3}$$

4) Functionality as test suite quality attribute: The functionality defines as the capability of the software to perform functions that are related to user requirements with specific conditions[2]. The research analyzes the terms function in Java program related to the white-box testing is a method that also considers the coverage of the test suite. The test suite functionality is defined as the capability of test cases on the test suite to performs the behavior of the program. The test suite has performed the behavior of the java program when a high number of the method examines by the test suite. The formula for test suite functionality measurement as follows.

$$Test\ Suite\ Functionality = \frac{NOMetExec}{NOMet} \tag{4}$$

5) Portability as Test Suite Quality Attribute: Portability defines as the capability of software that can be reused from one hardware or software environment to another[2]. Test suite portability is defined as the capability of the test cases in the test suite to run on a new program without change. Portability is related to the degree of reusability of test suite. The test suite reusability defines as the capability of test cases in the test suite to examine several or all paths of method that should be tested on diverse objects.

This research uses the clones of Banker's Algorithm with code clones type 1, 2, 3, and 4 [20][21]. The test suite portability measurement is applied on code clones because the portability of the test cases in the test suite needs to use the same characteristic of input for the program. Code clone type 1 (exact clones) are identical clones with no differences with original code. Code clone type 2 which the differences from the original code are renamed identifiers, literals, types, layout, and comments but the structurally and syntactically are similar. Code clones type 3 are modified the statement such as statement insertions/deletions in addition to changes in identifiers, literals, types, and layouts. Code clone type 4 has been modified on code fragments to perform the same objective but different syntactic variants. By using the notation, the formula for test suite reusability as follows.

$$Test\ Suite\ Portability = \frac{\sum SRTC + \sum DCC}{(\sum OT \times \sum TC) + \sum NOLOC} \tag{5}$$

6) Maintainability as test suite quality attribute: Maintainability defines as the capability of software to be modified for correct defects, meet new requirements, make future maintenance easier, or adapted to a changing environment with less effort to maintain[2]. Test suite maintainability is related to the capability of the test suite that suitable to test another program with less effort to maintain by avoiding redundant test cases. The maintainability considers to reusability and efficiency(non-redundant test cases) of the test suite. By using the notation, the formula for test suite maintainability as follows.

$$Test\ Suite\ Maintainability = \frac{1 - (NOR/NOTC) + \left(\frac{\sum SRTC + \sum DCC}{(\sum OT \times \sum TC) + \sum NOLOC}\right)}{2} \tag{6}$$

IV. Reliability Analysis

Reliability analysis in this research is to validate the formulas of test suite quality attributes measurement. Validation of those formulas is to observe the result with the real condition based on the expert assessment. Reliability analysis objective is to validate the level of agreement from the result of measurement to the expert [22]. The validation refers more specifically to the consistency of measurement that involves the expert.

Cohen's kappa coefficient is generally for assessing agreement between raters. Cohen defined the coefficient as "the proportion of chance-expected disagreements which do not occur, or the proportion of agreement after chance agreement is removed from consideration"[23]. Cohen's Kappa has used a quantitative measurement of reliability for two raters that are rating the same thing, corrected for how often that the raters may agree by chance. The formula for Cohen's kappa coefficient as follows.

$$k = \frac{p_0 - p_c}{1 - p_c} \tag{7}$$

where,

p_0 = the proportion of units for which the judges agreed (relative observed agreement among raters)

p_c = the proportion of units for which agreement is expected by chance (chance-expected agreement)

TABLE I. COHEN'S KAPPA CONTINGENCY MATRIX

		Rater 1	
		Category 1	Category 2
Rater 2	Category 1	a: number of agreements on category 1 P(a) = a/N	b: number of disagreements (judge 1 and category 2, and judge 2 and category 1) P(b) = b/N
	Category 2	c: number of disagreements (judge 1 and category 1, and judge 2 and category 2) P(c) = c/N	d: number of agreements on category 2 P(d) = d/N

TABLE II. COHEN'S KAPPA INTERPRETATION

Kappa Statistic	**Strength of Agreement**
<0.00	Poor
0.00 – 0.20	Slight
0.21 – 0.40	Fair
0.41 – 0.60	Moderate
0.61 – 0.80	Substantial
0.81 – 1.0	Almost Perfect

Distribution of the frequency for two raters on Cohen's kappa coefficient is represented by the contingency matrix as shown in Table I [23]. Based on Cohen's kappa contingency matrix, p_o and p_c are calculated as follows:

$p_o=P(a)+P(d)$ $\quad P_{category1}=(P(a)+P(c))*(P(a)+P(b))$

$p_c=P_{category1}+P_{category2}$ $\quad P_{category2}=(P(b)+P(d))*(P(c)+P(d))$

The value of Cohen's kappa coefficient is positive when the value greater-than-chance agreement and negative when less-than-chance agreement. The maximum value for Cohen's kappa coefficient is +1.0. Its value related to the strength of agreement as shown in Table II.

V. RESEARCH METHODOLOGY

This section explains the research methodology to measure the test suite quality attributes and then validate the result by using the reliability analysis. The reliability analysis is used Cohen's kappa coefficient to validate the result of test suite quality measurement based on the level of agreement from the result of measurement and expert assessment. Fig. 1 shows the test suite quality attributes measurement and validation activity that consists of two main activities such as test suite profiling for test suite quality measurement and expert assessment for validating the result of measurement.

A. *Proposed Test Suite Quality Measurement Activity*

The objective of test suite profiling is to collect the important and useful information of the test cases in the test suite for the test suite quality attributes measurement in white-box testing. Test suite profiling uses the Java program and given test suites then running the test suite which has been implemented on Junit to test the Java program and the result is test suite information. The test suite information contains such as follow.

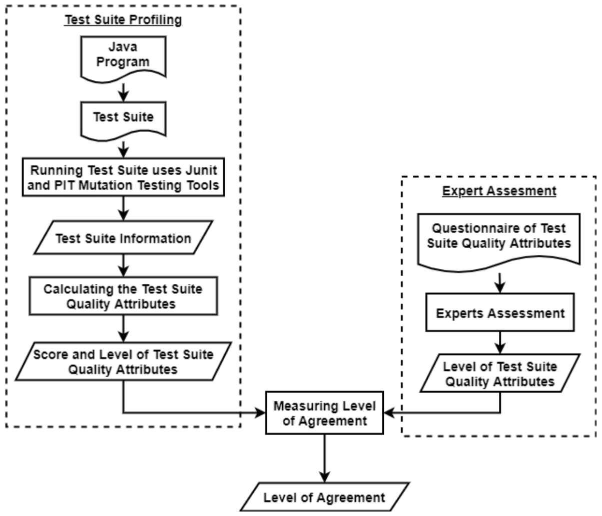
Test Suite Profiling
Java Program
Test Suite
Running Test Suite uses Junit and PIT Mutation Testing Tools
Test Suite Information
Calculating the Test Suite Quality Attributes
Score and Level of Test Suite Quality Attributes
Expert Assesment
Questionnaire of Test Suite Quality Attributes
Experts Assessment
Level of Test Suite Quality Attributes
Measuring Level of Agreement
Level of Agreement

Fig. 1. Test Suite Quality Attributes Measurement and Validation Activity.

a) Number lines of code,

b) Number lines code executed,

c) Number of test cases,

d) Distinct lines of code executed,

e) Number of mutants

f) Number of mutants killed by the test suite

g) Number of methods

h) Number of method executed.

Test suite information uses to calculate test suite quality attributes. Test suite quality attributes calculated in this research is usability, efficiency, reliability, functionality, portability, and maintainability. The result of the calculation is the score and level for test suite quality attributes.

B. *Validation*

Validation aims to prove the validity of test suite quality attributes measurement results with expert assessment. The expert assessment activity objective is to assess the quality of the test suite based on the experience from the expert. The questionnaire contains information and question that should answer by the expert such as how long the experience in software engineering and his work. The questionnaire provides information such as the explanation for test suite quality attributes, Java program and given test suites, and the result of test suite examination. The expert answers the question of test suite quality attributes assessment based on their experience, and knowledge. The expert assesses the quality of the test suite by choosing the level of quality such as low, medium, and high quality.

The level of test suite quality attribute from the test suite quality attributes measurement and expert assessment then analyze the reliability by using Cohen's kappa coefficient. The profiling and expert assessment for test suite quality attributes are used the same dataset. Cohen's Kappa coefficient and percentage of agreement are used to measure the level of agreement for test suite quality attributes. The result of measurement is the value of Cohen's kappa coefficient and percentage of agreement from test suite quality attributes measurement and expert assessment.

VI. The Experiment

The purpose of the experiment is to collect the test suite information for test suite quality attributes measurement for white-box testing. The result of measurement then validates the level of agreement from the result of measurement by using the result of expert assessment.

A. Dataset

The experiment uses Banker’s Algorithm Java program as shown in Fig. 2 and given two test suites for the dataset as shown in Table III that contains the number of test case (TC), input data, and expected output. Banker’s Algorithm is developed by Dijkstra for resource allocation and deadlock avoidance algorithm[11], [24]. The number of test cases is seven for each test suite. The test suites are implemented on Junit to examine the Banker’s Algorithm. The test suite mutants information gains by using PIT mutation testing tool[25] and Junit for code coverage information.

B. Experiment Works

The Banker’s Algorithm java program is examined by the given test suites in which test cases implemented in Junit. The test suite examination result is code coverage information and mutation coverage for profiling the test suite. The code coverage information for each test suite is collected by executing the Junit in Eclipse IDE that presents the information of lines of code executed by the test case in the test suite and percentage of code coverage.

The test suite mutants information is collected by using PIT mutation testing tool. PIT mutation testing tool are greatly manipulated bytecode to generates mutants and examine the test suite or test case to know the capability of the test suite to kill the mutants[25]. The mutants are killed by the test case in the test suite by showing different behaviour, and live when they are not. The ratio of mutants in the test suite is calculated in PIT mutation testing tool. The ratio of mutants is calculated by the mutants killed over the total number of mutants. The mutation score is used in test suite reliability measurement by combining with code coverage. Test suite quality attributes such as usability, efficiency, reliability, functionality, portability, and maintainability are calculated by using the formula that explains in Section 2.

The experiment uses the questionnaire to gathering the test suite quality attributes information from the expert assessment. The questionnaire contains the introduction and aims of the test suite quality attribute assessment, dataset, result of the test suites execution, and summary of the test suites examination. The question is focused on assessing an aspect of the level of test suite quality attributes from an expert view with seven questions. They are designed from the definition of test suite quality attributes and the result of all test suites examination. An example of the question is as follows.

```
public class Banker {
private final int need[][], allocate[][], max[][], avail[][], np, nr;
public Banker(int[][] need, int[][] allocate, int[][] max, int[][] avail, int np, int nr) {
 if (need.length != np || allocate.length != np || max.length != np || avail.length != 1|| need[0].length != nr ||
allocate[0].length != nr || max[0].length != nr || avail[0].length != nr) {
 throw new IllegalArgumentException("The matrices should have \"np\" rows and \"nr\" columns. \"avail\" should have only one
row.");}
 this.need = need;
 this.allocate = allocate;
 this.max = max;
 this.avail = avail;
 this.np = np;
 this.nr = nr;}
 private int[][] calc_need() {
  for (int i = 0; i < np; i++) {
   for (int j = 0; j < nr; j++) {
  need[i][j] = max[i][j] - allocate[i][j];}
 }
 return need;}
 private boolean check(int i) {
 for (int j = 0; j < nr; j++) {
  if (avail[0][j] < need[i][j]) {
  return false;}
 }
 return true;}
 public boolean isSafe() {
  calc_need();
        boolean done[] = new boolean[np];
        int j = 0;
        while (j < np) {
         boolean allocated = false;
         for (int i = 0; i < np; i++) {
         if (!done[i] && check(i)) {
          for (int k = 0; k < nr; k++) {
avail[0][k] = avail[0][k] - need[i][k] + max[i][k];}
 System.out.println("Allocated process : " + i);
 allocated = done[i] = true;
  j++;}
}
  if (!allocated) {
   break;}
}
return j == np;}
}
```

Fig. 2. Java Program for Banker’s Algorithm.

TABLE III. TEST SUITES OF BANKER'S ALGORITHM

Test Suite-1	**TC-1**	Input data	int[][] intArray0 = new int[1][1]; int[][] intArray1 = new int[1][1]; int[] intArray2 = new int[1]; intArray2[0] = 1; intArray1[0] = intArray2;	**Test Suite-2**	**TC-1**	Input data	int[][] intArray0 = new int[1][1]; int[][] intArray1 = new int[1][6];
		Expected Output	True or false for method isSafe()			Expected Output	fail("Expecting exception: IllegalArgumentException")
	TC-2	Input data	int[][] intArray0 = new int[1][1]; int[] intArray1 = new int[1]; intArray1[0] = 1; intArray0[0] = intArray1;		**TC-2**	Input data	int[][] intArray0 = new int[1][1]; int[][] intArray1 = new int[1][1]; int[] intArray2 = new int[9]; intArray1[0] = intArray2;
		Expected Output	True or false for method isSafe()			Expected Output	fail("Expecting exception: IllegalArgumentException")
	TC-3	Input data	int[][] intArray0 = new int[1][6];		**TC-3**	Input data	int[][] intArray0 = new int[0][6];
		Expected Output	True or false for method isSafe()			Expected Output	fail("Expecting exception: IllegalArgumentException")
	TC-4	Input data	Banker banker0 = null;		**TC-4**	Input data	int[][] intArray0 = new int[1][6]; int[][] intArray1 = new int[3][0];
		Expected Output	True or false for method isSafe()			Expected Output	fail("Expecting exception: IllegalArgumentException")
	TC-5	Input data	int[][] intArray0 = new int[0][6]; int[][] intArray1 = new int[1][0];		**TC-5**	Input data	int[][] intArray0 = new int[0][6]; int[][] intArray1 = new int[1][0]; int[][] intArray1 = new int[1][0];
		Expected Output	True or false for method isSafe()			Expected Output	fail("Expecting exception: IllegalArgumentException")
	TC-6	Input data	int[][] intArray0 = new int[1][1]; int[][] intArray1 = new int[1][1]; int[] intArray2 = new int[1]; intArray2[0] = 1; intArray0[0] = intArray2;		**TC-6**	Input data	int[][] intArray0 = new int[1][6];
		Expected Output	True or false for method isSafe()			Expected Output	fail("Expecting exception: IllegalArgumentException")
	TC-7	Input data	int[][] intArray0 = new int[1][1]; int[][] intArray1 = new int[1][1]; int[] intArray2 = new int[9]; intArray1[0] = intArray2;		**TC-7**	Input data	int[][] intArray0 = new int[11][6];
		Expected Output	fail("Expecting exception: IllegalArgumentException")			Expected Output	fail("Expecting exception: IllegalArgumentException")

"The efficiency of test suite usability in this research is related to the degree of redundancy of test cases on the test suite. The test suite efficiency defines as the level of test suite redundancy to complete a certain task. The redundant test cases exist when both of the two test cases are executed in the same lines of code.

Based on the definition and the summary of test suites examination, what is the degree of test suite efficiency for test suite 1 and test suite 2?".

The expert will choose one answer such as low, medium, high for each test suite. The questionnaire collects the personal information of the expert like name, kind of experience in software engineering. Result of the questionnaire is presenting the level of each test suite quality attributes of every test suite.

TABLE IV. RESULT OF TEST SUITE INFORMATION

Test Suite Information	Test Suite-1	Test Suite-2
Number lines of code	33	33
Distinct lines of code executed	33	3
Number of test cases	7	7
Number of mutants	41	41
Number of mutants killed	32	3
Number of methods	4	4
Number of methods executed	4	1
Number of object reused test suite(test cases)	4	4
Number of successful reused test cases for all object tested	28	28
Number of redundant test cases	3	6

TABLE V. RESULT OF TEST SUITE QUALITY ATTRIBUTES MEASUREMENT

Test Suite Quality Attributes	**Test Suite-1**		**Test Suite-2**	
	Score	Test Suite Quality Attribute Level	Score	Test Suite Quality Attributes Level
Usability	0.79	High	0.12	Low
Efficiency	0.57	Medium	0.14	Low
Reliability	0.78	High	0.07	Low
Functionality	1.00	High	0.25	Low
Portability	1.00	High	0.51	Medium
Maintainability	0.79	High	0.33	Low

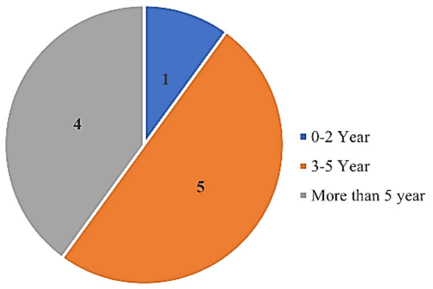

Fig. 3. Expert Time Experiences in Software Engineering.

The result of level test suite quality attributes and expert assessment is used to calculate the level of agreement by using Cohen's kappa coefficient and percentage of agreement. The contingency matrix for Cohen's kappa coefficient contains several conditions. True positive condition is the total number of instances that both raters said correct. False positive condition is the total number of instances that the result of test suite quality attribute measurement said incorrect, but experts said correct. False negative condition is the total number of instances that the result of test suite quality attribute measurement said correct, but the experts said incorrect. True negative condition is the total number of instances that both the result of test suite quality attribute measurement and the experts said incorrect. The value in the contingency matrix is used to calculate Cohen's kappa.

The percentage of agreement is calculated based on comparison data from the result of the level of test suite quality attributes measurement and expert assessment. Criteria of the match agreement if the result from the expert is the same as the measurement. The result from Cohen's kappa coefficient and percentage of agreement is to enrich the analysis of reliability in test suite quality attributes.

C. *Experiment Result*

The dataset from Banker's Algorithm and given test suites then examines by using Junit and PIT mutation testing tool, the result of test suite information is shown in Table IV. The test suite information consists of the number of lines of code, number of lines executed by the test case, distinct number of lines executed, number of methods, and number of method executed. Table V shows the result of test suite quality attributes measurement for usability, efficiency, reliability, functionality, portability, and maintainability by using the formula in Section 2 and their quality level.

The level of agreement is analyzing by Cohen's kappa coefficient and percentage of agreement. The Cohen's kappa coefficient in this research is to present the degree of agreement. The percentage of agreement is to enhance detailed information about the percentage of test suite quality attributes agreement. The number of experts is ten with different year experiences as shown in Fig. 3. Most of the experts have experience 3-5 years in software engineering. Time experience in software engineering from the expert helps to answer the question.

Calculation of Cohen's kappa coefficient uses the value of true positive, false positive, false negative, and true negative on contingency matrix from the result of the questionnaire as shown in Table VI. The percentage of agreement is measured with the proportion of experts who respond that identical or similar, lower or higher than the result of test suite quality attributes measurement as shown in Table VII. Similar terms in Table VII means that the result from the expert is the same as the measurement result, lower means that the result from the expert is lower than the measurement result, and higher means that the result from the expert is higher than the measurement result.

TABLE VI. CONTINGENCY MATRIX FOR ALL TEST SUITE AND EACH TEST SUITE

Cohen's Kappa Coefficient											
All Test Suite				Test Suite -1				Test Suite-2			
	Expert				Expert				Expert		
Measurement	Yes	No	Total	**Measurement**	Yes	No	Total	**Measurement**	Yes	No	Total
Yes	31	40	71	**Yes**	28	4	32	**Yes**	4	37	41
No	29	20	49	**No**	20	6	26	**No**	1	19	20
Total	60	60	120	**Total**	48	10	58	**Total**	5	56	61
Po	0.425			**Po**	0.586			**Po**	0.377		
Pc	0.5			**Pc**	0.533			**Pc**	0.356		
Kappa	-0.15	Less than Chance Agreement		**Kappa**	0.112	Slight Agreement		**Kappa**	0.032	Slight Agreement	

TABLE VII. RESULT OF PERCENTAGE OF AGREEMENT FOR ALL TEST SUITE AND EACH TEST SUITE

Quality Attributes	Similar		Lower		Higher	
	Total	Percentage	Total	Percentage	Total	Percentage
Usability	11	55%	6	30%	3	15%
Efficiency	6	30%	1	5%	7	35%
Reliability	10	50%	3	15%	7	35%
Functionality	13	65%	2	10%	5	25%
Portability	8	40%	7	35%	9	45%
Maintainability	4	20%	7	35%	9	45%
All Attributes	52	43%	26	22%	40	33%

VII. DISCUSSION

This section provides implications from the result of the questionnaire to test suite quality attributes measurement. The Cohen's kappa coefficient result for all test suites is – 0.15 which means less than chance agreement or no agreement as shown in Table VI. Negative value of Cohen's kappa coefficient is represented great disagreement between measurement and experts. Disagreement means that the observed agreement is less than chance agreement. There is no strict lower value for the kappa coefficient and its meaning. The weakness for a negative value of kappa has no fixed threshold for a lower value that difficult to have suitable interpretation from the assessment especially on the level of agreement.

The kappa coefficient is measured for each test suite. Table VI shows the result of kappa coefficient is 0.112 for test suite 1 and 0.032 for test suite 2 which has similar meaning is slight agreement. The slight agreement means that condition is needed to consider the result from the experts and measurements. Interpretation of the kappa coefficient for indicated the good agreement is not easy because in the terms of accuracy from a single kappa analysis itself.

The research analyzes that level of agreement by kappa coefficient is incompleted to represent the level of agreement between test suite quality attributes measurement and expert assessment. This research uses the percentage of agreement to complete the analysis. The percentage of agreement has improved the result of positive value of kappa coefficient. Table VII shows the percentage of agreement for all and each test suite quality attribute.

The result from all quality attributes measurement and the expert shows that 46% are similar to the test suite quality attributes measurement which means that the expert 46% agree with the principle of test suite quality attributes measurement. The highest attribute which similar is reliability with a percentage of 70%. The lowest attribute which similar is maintainability with a percentage of 20%.

The result of test suite quality attributes measurement for each test suite and expert assessment are confirmed for slight agreement. This result is approved by the result of the percentage of agreement. The research assumes that the level of agreement is strongly agreed when the percentage of the agreement greater than or equal to 50%. The result shows that usability with 55%, reliability with 50%, and functionality 65% which means that strongly agreed.

VIII. CONCLUSION AND FUTURE WORK

This research investigated the quality attributes for test suite based on the attributes of software quality. The attributes are usability, efficiency, reliability, functionality, portability, and maintainability that are selected from 28 attributes in software quality. The test suite quality attributes measurement uses the result of test suite examination as parameters and input. The experiment uses the Banker's Algorithm by using given two test suites. The result of test suite quality attributes measurement is presented by score and level of quality as the degree of test suite quality.

The experiment uses reliability analysis to prove the validity of test suite quality attributes measurement. The reliability analysis uses Cohen's kappa coefficient and percentage of agreement. Cohen's kappa coefficient analyzes the reliability of test suite quality based on test suite quality attributes measurement result and expert assessment. The result of Cohen's kappa coefficient measurement is – 0.15 for all test suites, 0.112 for test suite 1, and 0.032 for test suite 2. The result of test suite quality attributes measurement for each test suite is confirmed for slight agreement. This result is approved by using the result of the percentage of agreement. The research assumes that the level of agreement is strongly agreed when the percentage of the agreement greater than or equal to 50%. The result shows that usability with 55%, reliability with 50%, and functionality 65% which means that strongly agreed. Hence, our proposed method is useful to measure test suite quality attributes.

Our approach is a method to measure test suite quality attributes for white box testing which is specifically contained usability, efficiency, reliability, functionality, portability, and maintainability as attributes. In the future, it's important to consider the weight of each test suite quality attribute to define a formula that can measure test suite quality more accurately.

REFERENCES

[1] Lovely Professional University, Software Testing and Quality Assurance. New Delhi: Excel Books Private Limited, 2012.

[2] International Software Testing Qualifications Board (ISTQB), "Standard Glossary of Terms used in Software Testing Version 3.5," 2020.

[3] P. D. Roger S. Pressman, Software Engineering: A Practitioner's Approach, 7th ed. New York: McGraw-Hill, 2009.

[4] I. Sommerville, "Software Engineering 9th Edition., USA: Addison-Wesley Publishing Company, 133–170. 2010.

[5] Z. Nayyar, N. Rafique, N. Hashmi, N. Rashid, and S. Awan, "Analyzing Test Case Quality with Mutation Testing Approach," Proc. 2015 Sci. Inf. Conf. SAI 2015, pp. 902–905, 2015.

[6] G. Grano, “A new dimension of test quality: Assessing and Generating Higher Quality Unit Test Cases,” Proc. 28th ACM SIGSOFT Int. Symp. Softw. Test. Anal., pp. 419–423, 2019.

[7] M. M. Ali-Shahid and S. Sulaiman, “A Case Study on Reliability and Usability Testing Of A Web Portal,” in 2015 9th Malaysian Software Engineering Conference (MySEC), 2015, no. June, pp. 31–36.

[8] F. A. Muqtadiroh, H. M. Astuti, E. W. T. Darmaningrat, and F. R. Aprilian, “Usability Evaluation to Enhance Software Quality of Cultural Conservation System Based on Nielsen Model (WikiBudaya),” Procedia Comput. Sci., vol. 124, pp. 513–521, 2017.

[9] R. Ibrahim, M. Ahmed, R. Nayak, and S. Jamel, “Reducing Redundancy of Test Cases Generation Using Code Smell Detection and Refactoring,” J. King Saud Univ. - Comput. Inf. Sci., vol. 32, no. 3, pp. 367–374, 2020.

[10] A. Dimov, S. K. Chandran, S. Punnekkat, A. Nasir, and N. Azam, “Mutation Testing Framework for Software Reliability Model Analysis and Reliability Estimation,” 6th Central and Eastern European Software Engineering Conference (CEE-SECR), pp. 163–169. 2010.

[11] G. Guizzo, F. Sarro, and M. Harman, “Cost Measures Matter for Mutation Testing Study Validity,” Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1127–1139. 2020.

[12] Z. Q. Zhou, S. Xiang, and T. Y. Chen, “Metamorphic Testing for Software Quality Assessment: A Study of Search Engines,” IEEE Trans. Softw. Eng., vol. 42, no. 3, pp. 264–284. 2016.

[13] S. Makady and R. J. Walker, “Debugging and Maintaining Pragmatically Reused Test Suites,” Inf. Softw. Technol., vol. 102, no. March 2017, pp. 6–29, 2018.

[14] H. Ghandorh, A. Noorwali, A. B. Nassif, L. F. Capretz, and R. Eagleson, “A Systematic Literature Review for Software Portability Measurement,” Proceedings of the 2020 9th International Conference on Software and Computer Applications, pp. 152–157. 2020.

[15] D. Pfluger et al., “The Scalability-Efficiency/Maintainability-Portability Trade-Off in Simulation Software Engineering: Examples and a Preliminary Systematic Literature Review,” in 2016 Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCSE), pp. 26–34. 2016.

[16] I. C. Society, Guide to the Software Engineering Body of Knowledge Version 3.0 (SWEBOK Guide V3.0).

[17] Suman and W. Manoj, “A Comparative Study of Software Quality Models,” Int. J. Comput. Sci. Inf. Technol., vol. 5, no. 4, pp. 1-8. 2014.

[18] ISO-Comitte, International Standard - ISO 9241-210. 2010.

[19] M. C. Saputra, T. Katayama, Y. Kita, H. Yamaba, K. Aburada, and N. Okazaki, “Test Cases Redundant Elimination on Code Coverage Uses Distance and Correlation Measurement Method,” Proc. Int. Conf. Artif. Life Robot., vol. 25, pp. 755–758. 2020.

[20] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, “Comparison and Evaluation of Clone Detection Tools,” IEEE Trans. Softw. Eng., vol. 33, no. 9, pp. 577–591, Sep. 2007.

[21] C. K. Roy and J. R. Cordy, “Survey on Software Clone Detection Research,” in Technical Report No. 2007-541, 2007.

[22] R. T. Lange, “Inter-rater Reliability,” in Encyclopedia of Clinical Neuropsychology, J. S. Kreutzer, J. DeLuca, and B. Caplan, Eds. New York, NY: Springer New York, pp. 1348–1348. 2011.

[23] J. Pérez, J. Díaz, J. Garcia-Martin, and B. Tabuenca, “Systematic Literature Reviews in Software Engineering Enhancement of The Study Selection Process Using Cohen’s Kappa Statistic,” J. Syst. Softw., vol. 168, p. 110657. 2020.

[24] “GitHub - iguit0/BankersAlgorithm: Dijkstra’s famous algorithm.” [Online]. Available: https://github.com/iguit0/BankersAlgorithm. [Accessed: 08-Apr-2021].

[25] H. Coles, T. Laurent, C. Henard, M. Papadakis, and A. Ventresque, “Demo: PIT A Practical Mutation Testing Tool for Java (demo),” in Proceedings of the 25th International Symposium on Software Testing and Analysis, 2016, pp. 449–452.