Software Quality Metrics Calculations for Java Programming Learning Assistant System

Khin Khin Zaw
Department of Computer Engineereing and Information Technology
Yangon Technological University
Yangon, Myanmar
thihakhinkhin85@gmail.com

Hsu Wai Hnin
Department of Computer Engineereing and Information Technology
Yangon Technological University
Yangon, Myanmar
hsuwaihnin007@gmail.com

Nobuo Funabiki
Department of Electrical and Communication Engineering
Okayama University
Okayama, Japan
funabiki@okayama-u.ac.jp

Khin Yadanar Kyaw
Department of Computer Engineering and Information Tetchnology
Yangon Technological University
Yangon, Myanamar
kk.yadanar@gamil.com

Abstract

A Web-based Java Programming Learning Assistant System (JPLAS) has been proposed to assist Java programming educations in universities. In the code writing problem, the correctness of an answer code from a student is verified by running the test code on JUnit. Besides, their quality should be measured using the metrics to assess them. The currently using plugin could only be measured on eclipse for offline answering in JPLAS. To calculate the metrics and implement in web-based JPLAS, there are several equations that have been reported. In this paper, we find the proper equations to calculate the metrics that provide the same results as from Eclipse plugin. The application results for 45 source codes showed that the adopted metrics equations provide the same results as the plugin.

Keywords: *JPLAS, JUnit, test code, code writing problem*

I. INTRODUCTION

Java is a secure, portable, and platform independent programming language. Java engineers have been demand. Thus, Java is taught in many universities and colleges.

To assist Java educations, *Java programming learning assistant system (JPLAS)* has been developed. In JPLAS, the student can solve the exercise problems both on online and offline using *Eclipse*. JPLAS provides different types of Java programming exercises to cover the different learning levels. The *code writing problem* asks a student to write a whole source code that passes the given *test code* on *JUnit*. Besides, *software metrics* should be measured to assess the quality of the answer code [1].

Software metrics can evaluate the software product under developments, which gives a vision on the quality to make a whole process more successful. These metrics are called *software quality metrics* or *product metrics*. In the code writing problem, seven quality metrics can be measured for assessment of the code. They include NOC - number of classes, NOM – number of methods, CC - cylomatic complexity, NBD – nested block depth, LCOM- the lack of cohesion, MLOC - method lines of code and TLOC – total lines of code [2]. They can be used for the size estimation, the complexity evaluation, and the maintenance of the code.

The currently using plugin can be used on Eclipsed to measure the metrics. Eclipse is used when the students solve the problems on offline. To measure the software metrics of a code, the equations to calculate them must be implementation in the JPLAS server. Unfortunately, there are several different equations to calculate some quality metrics. We need to find proper equation to calculate the metrics that provide the same results from Eclipse plugin.

In this paper, first, equations to calculate each metric are surveyed. Then, the proper one is selected by comparing the results with Eclipse plugin for 45

Identify applicable funding agency here. If none, delete this text box.

answer codes. The results showed that the calculated metrics by the selected equations are equal to those of Eclipse plugin.

II. REVIEW OF JPLAS

In this section, we review JPLAS.

A. JPLAS

In JPLAS, *Ubuntu* is used as the operating system that is running on *VMware*. For web server, *Tomcat* is used to run HTML, JSP and servlets. HTML- hypertext Markup Language is the language to create the webpages. JSP is a script combined with Java code and it is embedded to HTML. Servlet is a small Java program that runs on web server. For database, *MySQL* is used for managing the data.

JPLAS implements supporting functions for teachers and students. In the teacher function, the teacher can show the problems to students by uploading them to the server. In the student function, the student can answer the problems from a Web browser and submit them to the server on online.

Besides, they can answer the problems using Eclipse on offline after receiving the problems by e-mail or downloading the problems from JPLAS, and submit the answer via email.

B. Exercies Problems in JPLAS

In JPLAS, four types of problems are provided. The three problems are the types of fill-in-blank problems to understand the grammar and reading studies. The last problem is for the code writing study. In this paper, the code writing problem is focused for the study of software metrics.

C. Fill-in-blank Problem

In this problem, the students need to fill the correct answers in blanks for a given Java code. The answer is marked by comparing with their original elements in the code. Thus, the original element must be unique correct answer for each blank. The blanks elements are identifiers, variable, reserved words and control symbols.

D. Value Trace Problem

This problem is another type of element fill-in-blank problem that keeps the process of filling and marking the answer. It questions a student about actual values of important variables in the code. The students need to fill the correct values of variables in blanks. In this problem, the output data of variables from the code execution is blanked. The output data may contain one or more values. It blanks the output data line by line.

E. Statement Fill-in-blank Problem

In this problem, the students need to fill the whole statement in a blank for a give Java code. The answer is marked by using the test code on *JUni*t as the code writing problem.

F. Code Writing Problem

In this problem, the students need to write the whole code as the information given in test code. Then, the answer code is tested through the test code on *JUnit* as *test driven development method* (TDD). Besides, their quality is accessed by measuring the seven-quality metrics on *Eclipse plugin*.

G. JUnit

It is an open-source Java framework for unit testing on Java programming language and adopted in JPLAS. It is important in development of TDD - test driven development. Test code is implemented on *JUnit*. Although, test code is a test code programming language, it is rather simple for the Java programmers. This reason is that *JUnit* has been designed for Java programming language. In *JUnit*, one test can be performed by using one method in the *JUnit* library. In the code writing problem, the method whose name starts with 'assert' is used to check the execution results by comparing with the expected method [3].

H. TDD Method

In the TDD method, the following process can be done.

*1)*The source code must be prepared first to write test code from them. Thus, the test code includes the information on model source code, it will be tested in later.

*2)*Then, the answer code is written and tested it on *JUnit* through test code.

*3)*The source code can be re-factored until passing through the test code. Thus, the re-factoring process of a source code becomes easy, because the modified code can be tested instantly.

I. Metric Plugin

Metric plugin for Eclipse is commonly used open source software plugin for metrics calculation. This plugin can measure the various metrics on source code and their results are shown by number in metric view. This plugin is used to measure software metrics to assess the quality of the code for the code writing problem [4][5].

III. Software Metrics

Software metrics are the measurement and prediction of software products, which are essential resources for a project and products relevant for software evolutions. Measurements can be used throughout the software project for quality control by comparing the current measurements with past measurements for similar projects.

A. Overview of Metrics

There are many metrics that can be categorized into process and product metrics. Besides, due to the great interest in the use of object-oriented languages, many object-oriented design metrics has been proposed. *Product* metrics measure size, complexity, quality, and reliability of software product. *Process* metrics measure the various characteristics of the software development process. *Object-oriented* metrics measure the different aspects of object-oriented design, including complexity, cohesion, and coupling. Among them, *product* metrics and *object-oriented* metrics measure the quality of the code.

B. Product Metrics

Product metrics are known as quality metrics. They help improving the quality of the different system components, and comparisons between existing systems. Various kinds of product metrics have been proposed. They include reliability metrics, functionality metrics, performance metrics, usability metrics, cost metrics, size metrics, complexity metrics and style metrics. They are used to measure the properties of the software. Among them, some quality criteria can be used to predict a certain quality of the software [6] and they are as follows:

- NOC - number of classes and DIT - depth of inheritance are measured to assess maintainability and reusability of the program.
- LOC- lines of code is measured to assess the size of the code.
- CC - cyclomatic complexity is measured to assess reliability of the program.

C. Object- Oriented Metrics

Object-oriented designs are more beneficial in software development environment. Object-oriented metrics are used to measure properties of object-oriented designs. The object-oriented metrics measure on class and its design viz; localization, encapsulation, inheritance, polymorphism, and object abstraction techniques, which make the class unique. The object-oriented metrics are defined as follows:

- WMC - Weighted Methods Per Class
- DIT - Depth of Inheritance Tree
- NOC - Number of Children
- CBO - Coupling between Objects
- RFC - Response for a Class
- LCOM - Lack of Cohesion in Methods

IV. Calculation of Seven Metrics

Seven quality metrics are adopted for the code writing problem in JPLAS. They are as the followings:

- NOC - number of classes
- NOM - number of methods
- CC - cyclomatic complexity
- LCOM - lack of cohesion in method
- NBD - nested block depth
- MLOC - method lines of code
- TLOC- total lines of code

The equations of CC and LCOM have many variations, whereas other five metrics have a unique one. In this paper, firstly, all the equations are surveyed. Then, proper equations are selected to calculate them.

A. Number of Classes (NOC)

NOC measures the number of classes within the application package. It is a measure of how many subclasses are going to inherit the methods in the parent class. If a class has many subclasses, it is regarded as the bad design. The lower value of NOC helps maintainability and complexity of codes.

B. Number of Methods (NOM)

NOM measures the number of methods within classes. The number of methods that are local to the class and only those methods can be measured.

C. Cyclomatic Complexity (CC)

CC measures the structural complexity of a procedure by counting the number of independent paths in a method. The paths represent the number of decision points in the code, which include if, while, do-while, for, switch-case-defaults, try-catch finally. The goal of CC is to evaluate the testability and maintainability of a software module [8].

The original complexity is calculated as follows:

$$CC = E - N + 2 \quad (1)$$

Where: CC = cyclomatic complexity

E = the number of edges of the graph

N = the number of nodes of the graph

Then, the improve complexity is defined as the followings [9]:

1) If the source codes contain no decision points, their complexity would be 1 since there is only a single path through the code.

2) If the code has a single IF statement containing a single condition, there would be two paths through the code, one path for TRUE and one path for FALSE.

In above conditions, CC is calculated as follows:

$$CC = E - N + 2P \quad (2)$$

where:

P = the number of connected components

3) An alternate function is used when the cyclomatic complexity is applied to several subprograms at the same time.

$$CC = E - N + P \quad (3)$$

The following example code contains a single IF statement. Thus, it contains the two paths to evaluate the path as TRUE of FALSE.

```
1.  public class Circle{
2.  public static int minFunction(int n1,int n2){
3.              int min;
4.              if(n1>n2)
5.                min=n2;
6.              else
7.                min=n1;
8.              return min;
9.        }
10. }
```

Figure 1. Example code single *If* Statement

Firstly, the statements are transformed into a graph, where every piece of a statement is represented as a node and their flows (sequence of execution of statements) are represented as the edges. For the single program, P is always equal to 1 since it has a single exit point. The cyclomatic complexity may be applied to several subprograms at the same time, where P will be equal to the number of programs. Figure 2 shows the flow chart of the source code containing single IF statements.

In this example, there are seven nodes, seven edges and one connected component. Then, CC= 7-7+2×1=2 is calculated by equation (2).

D. Lack of Cohesion in Methods (LCOM)

LCOM measures the cohesiveness of a class. It represents the difference between two methods whose similarity is zero or not. LCOM can judge the cohesiveness among the class methods. There are several LCOM metrics. The LCOM takes its values in the range 0 to 1.

- If the two methods share at least one field, Q is increased by one. Otherwise, Q is increased by one. It is noted that P and Q are initialized by 0. LCOM is calculated on each pair of metrics as follows [10]:

$$LCOM = (P > Q)? (P - Q):0 \quad (4)$$

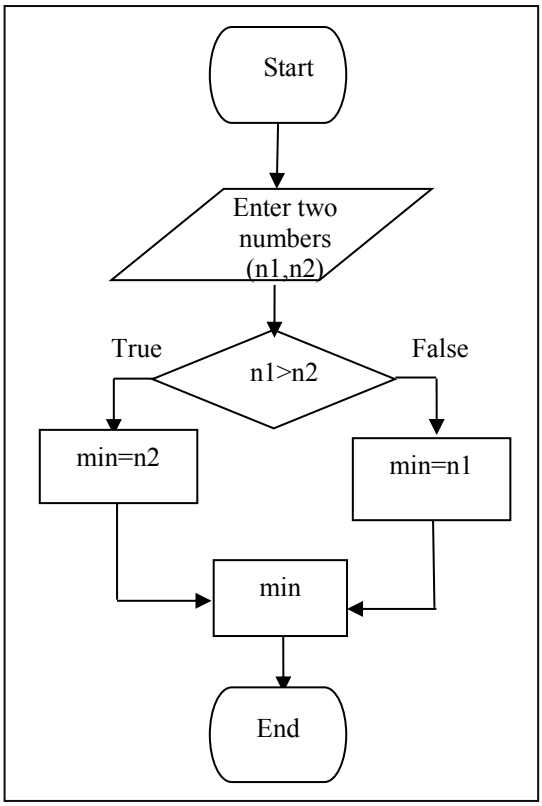

Figure 2. Data flow diagram for source code with single IF statement

- LCOM 1: A low value indicates the high coupling between methods. This also indicates the potentially high reusability and good class design. A high LCOM indicates that a class shall be considered for good design. LCOM = 0 is not a strong evidence that a class enjoys cohesiveness.
- LCOM 2: This is an improved version of LCOM 1.

$$LCOM = 1 - \frac{sum(mA)}{m*a} \quad (5)$$

Where: m = number of methods in class
a = number of attributes in class
mA = number of accessing times of attributes among the methods

- LCOM 3: A completely new expression for cohesion is proposed by Herderson-Sellers, that is called LCOM* [11].

$$\text{LCOM} = \frac{\left(\frac{1}{a}\Sigma_{j=1}^{a}\mu(Aj)\right) - m}{1 - m} \quad (6)$$

Where: $\mu(Aj)$= number of accessing times of attributes among the methods.

The following example code shows the accessing of attributes (member variables) among the methods. It contains two-member variables: *radius* and *colour*, and three methods: *getRadius*, *getColor* and *getData*. In this example, *radius* is accessed three times by *getRadius* and *getData*. *color* is accessed one time by *getColor*.

By equation (4), LCOM is calculated on the number of accessing times of attributes among the methods. The result is 0 because the value of Q is greater than P where P=0 and Q=4. Then, by equation (5), LCOM is calculated by using the number of methods, attributes and the number of accessing times of attributes. The result is 0.4 where m=3, a=2 and mA=4.

Fig.4 shows the diagram for the methods that accessing the attributes. By equation (6), LCOM is calculated by using the number of methods, attributes and the number of accessing times of attributes. LCOM is 0.5 where m=3, a=2 and $\mu(Aj)$ =4 respectively by equation (6).

According to the equation (4), LCOM is only 0 or 1. According to equation and (5) and (6), LCOM decreases and close to 0, when the accessing times of attributes are more. On the other hand, LCOM increases and close to 1 when the accessing times of attributes are less. The declared variables should be accessed among the methods In this time, equation (6) is selected by the calculation result that is same with plugin in Section V.

```
1. public class Circle{
2.    private double radius;
3.    private String color;
4.    public  Circle (){
5.          radius = 1.0;
6.          color = red;
7.     }
8.     public double getRadius () {
9.          return radius;
10.   }
11.    public String getColor (){
12.         return color;
13.    }
14.    public double getArea (){
15.         return 3.14*radius*radius;
16.    }
17. }
```

Figure 3. Example code for method accessing attributes

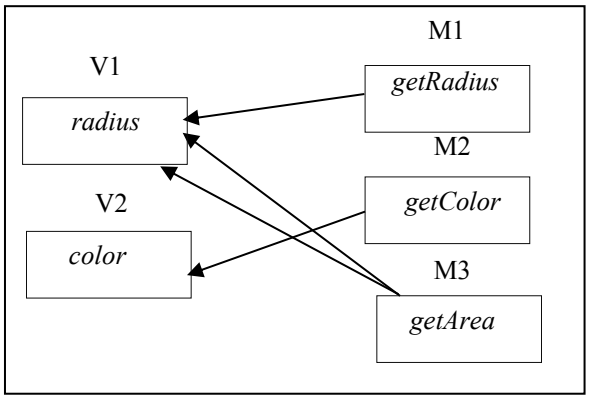

Figure 4. Block diagram of code for methods accessing attributes

E. Nested Block Depth (NBD)

NBD represents the maximum nest depth in a method. The nest depth is given by the number of statements of blocks that are nested due to the use of control structure (branches and loops).

F. Total lines of Code (TLOC)

TLOC measures the total number of lines in the source code. It is calculated by counting on the executable lines, comment and empty lines.

G. Methodl lines of Code (MLOC)

MLOC represents the total number of method lines in method. It is calculated by counting on comments and empty lines.

V. COMPARISON OF METRIC RESULTS

In this section, we evaluate the seven-quality metrics for the 45 source codes for five assignments from nine students in [11]. The first assignments ask the concepts of encapsulation, inheritance and polymorphism. The last two assignments ask the algorithms using those concepts for implementation it.

A. Calculated Metrics

Among the seven metrics, CC and LCOM have several equations to calculate them. For CC, the three equations (1), (2) and (3) are used. For LCOM, equation (6) is used.

B. Comparison of Metrics Values

The metrics values of CC and LCOM for each code are compared between the results by Eclipse plugin and those by the adopted equations to find the proper equations that give the same results. Other values represent the number of classes, methods, branches, and lines of code, which are unique. There is no equation for other metrics values. It can easily be calculated by counting the number of classes, methods, branches and lines of code as described in section IV- A, B, E, F, G. In this paper, the plugin result is represented as *T1* and the equation result is represented as *T2*.

Tables 1-5 show the calculated metrics values of seven metrics for each code. NOC, NOM, NBD, TLOC and MLOC are calculated by counting the number of classes, methods, branches and lines in a code manually. CC is calculated by using Equation (1) (2) and (3). LCOM is calculated by Equation (6) for each code. Then, the values are compared between the plugin and calculated results.

In each assignment, the values NOC, NOM, NBD, TLOC and MLOC are same between T1 and T2. Besides for CC are the same values between them. The values of LCOM are slightly varied between T1 and T2 because of the difference of significant digits. Thus, the adopted equations and the counted number for them are the same as those in Eclipse plugin. In assignment 1, 2 and 3, they don't need the conditions or branches (loops) to implement the concepts of OOP: encapsulation, inheritance and polymorphism. Thus, their CC and NBD are always 1 by both T1 and T2. In assignment 4 and 5, it needs the conditions or branches (loops) to implement the algorithms using the OOP concepts. In assignment 4, its CCs are 2-3 and NBD is 1-3. In assignment 4, its CCs are 2-4 and NBD is 1-3. The values are varied among the codes depending on the student's implementation on code.

TABLE I. COMPARISON OF METRIC VALUES FOR ASSIGNMENT 1

	Assignment 1							
	NOC		*NOM*		*MLOC*		*TLOC*	
	T1	*T2*	*T1*	*T2*	*T1*	*T2*	*T1*	*T2*
S1	1	1	6	6	8	8	26	26
S2	1	1	6	6	8	8	26	26
S3	1	1	6	6	8	8	26	26
S4	1	1	6	6	8	8	26	26
S5	1	1	6	6	8	8	27	27
S6	1	1	6	6	8	8	26	26
S7	1	1	6	6	8	8	26	26
S8	1	1	6	6	8	8	26	26
S9	1	1	6	6	8	8	26	26
	NBD		*CC*		*LCOM*			
	T1	*T2*	*T1*	*T2*	*T1*	*T2*		
S1	1	1	1	1	0.5	0.5		
S2	1	1	1	1	0.5	0.533		
S3	1	1	1	1	0.5	0.533		
S4	1	1	1	1	0.5	0.533		
S5	1	1	1	1	0.7	0.667		
S6	1	1	1	1	0.5	0.533		
S7	1	1	1	1	0.7	0.667		
S8	1	1	1	1	0.6	0.6		
S9	1	1	1	1	0.7	0.667		

TABLE II. COMPARISON OF METRIC VALUES FOR ASSIGNMENT 2

	Assignment 2							
	NOC		*NOM*		*MLOC*		*TLOC*	
	T1	*T2*	*T1*	*T2*	*T1*	*T2*	*T1*	*T2*
S1	2	2	5	5	5	5	28	28
S2	2	2	5	5	5	5	24	24
S3	2	2	5	5	5	5	24	24
S4	2	2	5	5	5	5	24	24
S5	2	2	12	12	15	15	48	48
S6	2	2	6	6	9	9	28	28
S7	2	2	5	5	5	5	24	24
S8	2	2	5	5	5	5	22	22
S9	2	2	9	9	11	11	40	40
	NBD		*CC*		*LCOM*			
	T1	*T2*	*T1*	*T2*	*T1*	*T2*		
S1	1	1	1	1	0.5	0.5		
S2	1	1	1	1	0.5	0.5		
S3	1	1	1	1	0.5	0.5		
S4	1	1	1	1	0.5	0.5		
S5	1	1	1	1	0.7	0.68		
S6	1	1	1	1	0.5	0.5		
S7	1	1	1	1	0.7	0.667		
S8	1	1	1	1	0.5	0.5		
S9	1	1	1	1	0.7	0.667		

TABLE III. COMPARISON OF METRIC VALUES FOR ASSIGNMENT 3

	Assignment 3							
	NOC		*NOM*		*MLOC*		*TLOC*	
	T1	*T2*	*T1*	*T2*	*T1*	*T2*	*T1*	*T2*
S1	1	1	3	3	3	3	14	14
S2	1	1	3	3	3	3	12	12
S3	1	1	3	3	3	3	12	12
S4	1	1	3	3	3	3	12	12
S5	1	1	3	3	9	9	21	21
S6	1	1	3	3	3	3	12	12
S7	1	1	3	3	3	3	12	12
S8	1	1	3	3	6	6	15	15

S9	1	1	3	3	3	3	12	12
	NBD			***CC***			***LCOM***	
	T1		***T2***	***T1***		***T2***	***T1***	***T2***
S1	1		1	1		1	0	0
S2	1		1	1		1	0	0
S3	1		1	1		1	0	0
S4	1		1	1		1	0	0
S5	1		1	1		1	0	0
S6	1		1	1		1	0	0
S7	1		1	1		1	0	0
S8	1		1	1		1	0	0
S9	1		1	1		1	0	0

TABLE IV. COMPARISON OF METRIC VALUES FOR ASSIGNMENT 4

	Assignment 4							
	NOC		***NOM***		***MLOC***		***TLOC***	
	T1	***T2***	***T1***	***T2***	***T1***	***T2***	***T1***	***T2***
S1	1	1	5	5	29	29	50	50
S2	1	1	4	4	37	37	53	53
S3	1	1	4	4	37	37	53	53
S4	1	1	3	3	7	7	19	19
S5	1	1	4	4	20	20	36	36
S6	2	2	9	9	26	26	55	55
S7	1	1	5	5	26	26	44	44
S8	1	1	4	4	24	24	40	40
S9	1	1	5	5	39	39	60	60
	NBD			***CC***			***LCOM***	
	T1		***T2***	***T1***		***T2***	***T1***	***T2***
S1	3		3	4		4	0.4	0.4
S2	3		3	3		3	0.6	0.667
S3	3		3	3		3	0.6	0.667
S4	1		1	2		2	0.5	0.5
S5	2		2	2		2	0.8	0.8
S6	2		2	2		2	0.45	0.45
S7	2		2	2		2	0.2	0.2
S8	2		2	3		3	0.6	0.6
S9	3		3	3		3	0.5	0.5

TABLE V. COMPARISON OF METRIC VALUES FOR ASSIGNMENT 5

	Assignment 5							
	NOC		***NOM***		***TLOC***		***MLOC***	
	T1	***T2***	***T1***	***T2***	***T1***	***T2***	***T1***	***T2***
S1	3	3	5	5	10	10	34	34
S2	2	2	5	5	28	28	49	49
S3	2	2	5	5	28	28	49	49
S4	2	2	4	4	9	9	26	26
S5	2	2	5	5	17	17	39	39
S6	2	2	9	9	26	26	54	54
S7	2	2	6	6	36	36	59	59
S8	2	2	5	5	33	33	55	55
S9	2	2	13	13	45	45	87	87
	NBD			***CC***			***LCOM***	
	T1		***T2***	***T1***		***T2***	***T1***	***T2***
S1	2		2	2		2	0.5	0.5
S2	2		2	2		2	0.8	0.8
S3	2		2	2		2	0.8	0.8
S4	1		1	2		2	0.5	0.5
S5	2		2	2		2	0.9	0.9
S6	2		2	2		2	0	0
S7	2		2	3		3	0.1	0.1
S8	2		2	3		3	0.6	0.6
S9	3		3	2		2	0.5	0.5

VI. CONCLUSIONS

In this paper, we surveyed equations for calculating software quality metrics, verified through comparison between T1 and T2 and found the equations that provide the same values as the Eclipse plugin through applications to 45 source codes. In future works, we will implement the equations in JPLAS and evaluate the quality of source codes from students on real time.

REFERENCES

[1] N. Funabiki, Y. Matsushima, T. Nakanishi and N. Amano, "A Java programming learning assistant system (JPLAS) using test-driven development method," IAENG Int. J. Computer Science, vol. 40, no. 1, pp 38-46, 2013.

[2] K. K. Zaw and N. Funabiki, "A design-aware test code approach for code writing problem in Java programming learning assistant system," Int. J. Spaced-based and Situated Computing, vol. 7, no.3, pp.145-154, 2017.

[3] K. Beck, Test-driven development: by example, Addison-Wesley, 2002.

[4] Metric Plugin, http:// metrics.sourceforge.net.

[5] T. G. S. Filo and M. A. S. Bigonha, "A catalogue of thresholds for object-oriented software metrics," in *Proc. Softeng*, pp. 48-55. 2015.

[6] S. M. Jamali, "Object oriented metrics," Software.
Assurance Technology Center (SATC), 2006.

[7] R. D. Neal "The measurement theory validation of proposed object-oriented software metrics," Dissertation, Virginaia Commonwealth University, 2008.

[8] Cyclomatic Complexity, http://www.projectcodemeter.com/cost_estimation/help/GL_cyclomatic.htm.

[9] K. K. Zaw and N. Funabiki, "A design-aware test code approach for code writing problem in Java programming learning assistant system," Int. J. Space-Based and Situated Computing, vol. 7, no.3, 2017.

[10] Lack of cohesion, http://www.tusharma.in/technical.

[11] K. K. Zaw, W.Zaw, N. Funabiki, Wen-Chung Kao, "An informative test code approach in code writing problem for three object-oriented programming concepts in Java programming learning assistant system", IAENG International Journal of Computer Science, vol.46, no.3, pp.445-453, 2019.