Basis Path Testing Implementation for Detecting Software Risk toward Errors and Failures

Andika Elok Amalia
Software Engineering Department
Institut Teknologi Telkom Purwokerto
Purwokerto, Indonesia
andika.amalia@ittelkom-pwt.ac.id

Emi Iryanti
Informatics Department
Institut Teknologi Telkom Purwokerto
Purwokerto, Indonesia
emi_iryanti@ittelkom-pwt.ac.id

***Abstract*— The software testing process is very important to avoid errors when the device is used by the user. Software testing is also an important component of software quality assurance (SQA), and a number of software organizations spend up to 40% of their resources for testing. Some software that has not gone through a testing process that is in accordance with the standard so that it does not have a testing document that results in the system still untested but has been used. In this paper, basis path testing is implemented to a software that never been tested before to evaluate the risk of the software towards errors and failure. From three functionalities that tested, one of them has cyclometic complexity value > 10 that means it has medium risk.**

***Keywords*-whitebox, basis path, errors, failure, software, testing**

I. INTRODUCTION

Software testing is an activity carried out to evaluate the quality of software that will be the basis for improvement if errors are found [1]. The purpose of systematic and gradual testing is to detect various errors with minimum time and also with a lot less effort. Software testing is also an important component of software quality assurance (SQA), and a number of software organizations spend up to 40% of their resources for testing [2].

The software testing process is very important to avoid errors when the device is used by the user. The results of a survey that was conducted, it was found that on average, 24% of the project development budget was allocated for testing. In relation to time resources, an average of 27% of the project implementation time is done for the schedule for testing [3].

In this paper, testing was carried out on a software in the form of an information system for new student admissions that used at Institut Teknologi Telkom Purwokerto. That is because of the software has never been tested but has been used. Also the testing is conducted to identify the risk of the software against errors and failures. The rest of this paper is organized as follows. Section 2 contains literature review about software testing and basis path testing. Section 3 explains the steps of the research that produced this paper. The experiment result and discussion are described in the next section, and the last section presents the conclusion.

II. THEORY

A. Software Testing

Detailed software testing or software testing is a formal process carried out by a special testing team on a software unit, several integrated software units or the entire package of software run a software program and check for errors. Tests are carried out based on the agreed testing procedures and test cases [3]. The main purpose of software testing is to ensure that the software being tested is ready for use.

To get good software testing results, it is necessary to plan testing steps or strategies in their implementation. There are 4 software testing strategies, namely unit testing, integration testing, system testing and acceptance testing. Unit testing is a test performed at the lowest level that tests a module or software component. The unit is the smallest module, the series of smallest lines of code that can be tested. Integration testing is testing when two or more units are combined into a larger structure and then tested, usually this test is carried out at the interface between components. While system testing tests the end-to-end quality of the entire system, it is often based on functional specifications and system requirements. Acceptance testing is a test that is performed when a complete system or software has been submitted to the user. Users will carry out several scenarios and claim to receive or request repairs from the complete software [2].

B. Basis Path Testing

Whitebox testing is an important software testing technique based on internal work analysis and software structure and can reveal implementation errors. The advantage of white box testing is it reveals errors in hidden code, side effects and can help in removing excess code lines [4]. Whitebox testing details and the internal structure of the system is made clear so that it requires input in the form of program code from the application to be tested.

Basis Path Testing is one of the techniques from whitebox testing that was first proposed by Tom McCabe and allows test designers to produce a measure of the logical complexity of procedural design which is then used as an approach to describe

ISBN 978-602-53524-0-9
The Ministry of Research, Technology, & Higher Education (*sponsors).*

the basic set of execution paths of the program. Steps of basis path testing can be described as follows [4].

Step 1. Change the line of code for each functionality into control flow graph (CFG) notation

Step 2. Calculate the Cyclometic Complexity of the CFG that has been created and determine a set of linear independent paths from the program

Step 3. Use the results of the second step as the test case

Step 4. Create graph matrices

III. RESEARCH METHODOLOGY

This this research was carried out through several stages as follows.

A. *Literature Study*

Before the testing is implemented, some literatures about software testing especially basis path testing testing are studied so that the research is conducted well directed.

B. *Collecting Software Requirement and Source Code*

In this step, the information about software requirement and specification is collected by meeting and discussion with software developer.

C. *Basis Path Testing Implementation*

Based on literature study on step A that was explained in section before that basis path testing has four steps as follows.

a) Change the line of code for each functionality into control flow graph (CFG) notation.

CFG is a directed graph consisting of nodes and streams. Node (N) is expressed with a circle labeled (number) representing one or more statements, outcome conditions or program procedures. The control flow is represented by the direction of the arrow which can also be called edge (E) which represents the flow of the flow program. In CFG there is a predicate node (P), which is a node containing conditions and region (R) that is an area directed by nodes and edges [5] [6] while CFG forms based on the program can be seen in Figure 1.

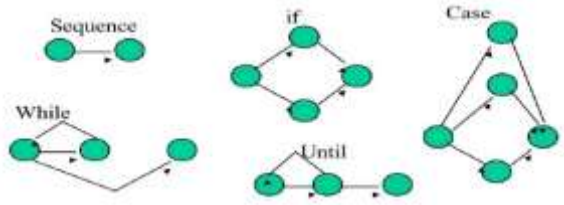

Figure 1. Forms of CFG

b) Calculate the Cyclometic Complexity of the CFG that has been created and determine a set of linear independent paths from the program

The next stage after CFG is the calculation of cyclometic complexity, a software metric that provides a quantitative measure of the fourth logical complexity. On the basis of path testing, the cyclometic complexity defines the number of independent pathways in the basic series of programs and provides an upper limit for the number of tests needed to guarantee the coverage of all program statements [4]. Cyclometic Complexity will be described in the form of flowgraph with the calculation of complexity as follows [4]. Determining the cyclometic complexity (V (G)) is carried out with the following formulas (1) and (2).

$$V(G) = \sum E - \sum N + 2 \ldots\ldots\ldots(1)$$

$$V\ (G) = \sum P + 1 \ldots\ldots\ldots\ldots\ldots\ldots(2)$$

Where E is the number of edges, N is the number of nodes and P is the number of predicate nodes.

c) Use the results of the second step as the test case

All of the path created in first step and second step then used to create test case. On this paper, the result of this step is not presented

.

d) Create graph matrices

Graph matrix is a two-dimensional matrix that has columns and rows equal to the number of nodes in CFG. To distinguish each other each node is represented by several numbers. Each box is provided with several link weights (0-not connected, 1-if there is a connection) [4]. Examples of making it can be seen in Figure 2, Table 1 and Table 2.

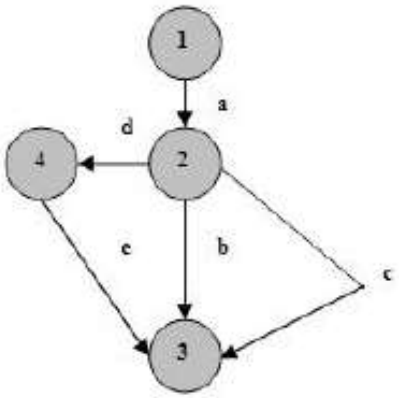

Figure 2. CFG Example

TABLE I. GRAPH MATRIX

Node	1	2	3	4
1		a		
2			b,c	d
3				
4			e	

TABLE II. CONNECTION MATRIX

Node	1	2	3	4	Connections

ISBN 978-602-53524-0-9

1	1			1-1 = 0
2		1,1	1	3-1 = 2
3				0
4		1		1-1 = 0

The column "Connections" in Table II is calculated by subtracting the weight of the link with the number 1 which can then be calculated by cyclometic complexity by summing connections from all rows.

IV. Result and Discussion

A. Software Requirement

Testing can be conducted well if the requirements and functional specifications of the software to be tested are known. Figure 3 is a use case diagram of the PMB module on D'ions for PMB_Participants actors.

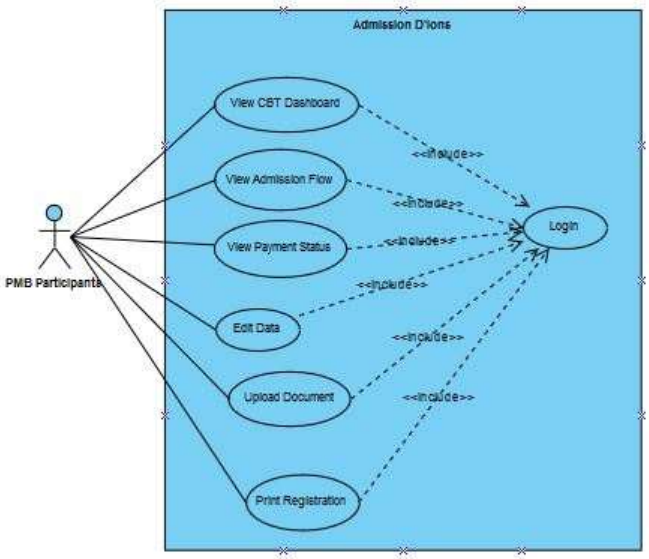

Figure 3. Use Case Diagram for PMB_Participants Actor

B. Source Code

In addition to the requirements described in the use case diagram, whitebox testing also requires a program code that is related to the functionalities that can be done by PMB_Participants. Program codes obtained from software developers can be seen in Table III.

TABLE III. Sour Code List

Directory	Source Code File
module/pmb	pmb.exam.dashboard.php
	pmb.participant.payment.status.php
	pmb.participant.documents.php
	pmb.participant.personal.data.php
	pmb.document.checklist.print.php
libraries	db.class.master.php
	db.class.pmb.payment.php
	db.class.pmb.php
libraries/ajax	ajax.datatables.php
	ajax.pmb.payment.php
	ajax.pmb.php

C. Basis Path Testing Result

Testing whitebox using base path testing is done on the functionality of "View CBT Dashboard", "View Payment Status" and "Print Registration". Each will be tested according to the stages in the Basis Path Testing method.

a) Use Case "View CBT Dashboard"

Figure 4 is control flow graph (CFG) of view CBT Dashboard functional.

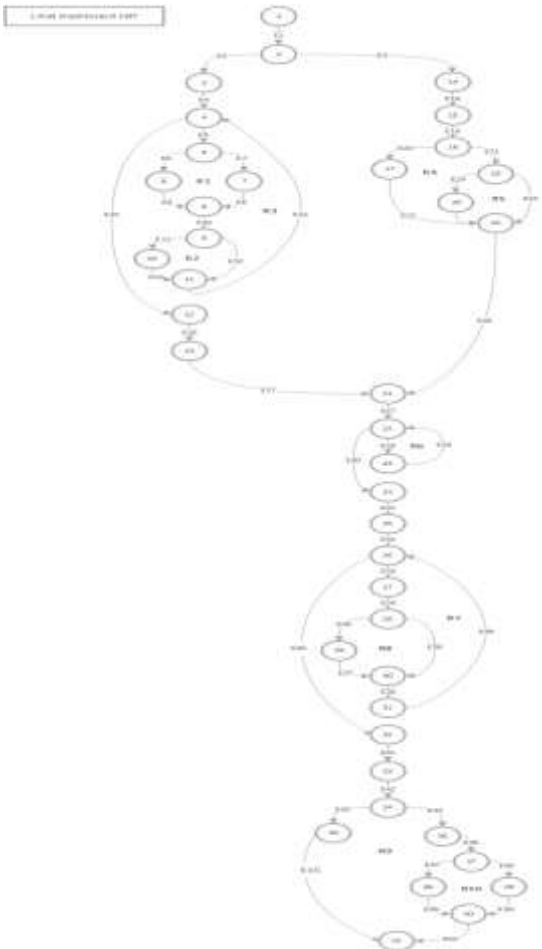

Figure 4. CFG "View CBT Dashboard"

From the CFG above, it is obtained the number of Nodes is 41, the number of Edge is 51 and the number of Predicate Node is 11. So from Formula (1) or (2) if calculated then,

$$V(G) = 51 - 41 + 2 = 12, \text{ or}$$

$$V\ (G) = 11 + 1 = 12$$

After that, graph matrix and connection matrix are created and get the result of V(G) is 11 as follows.

ISBN 978-602-53524-0-9

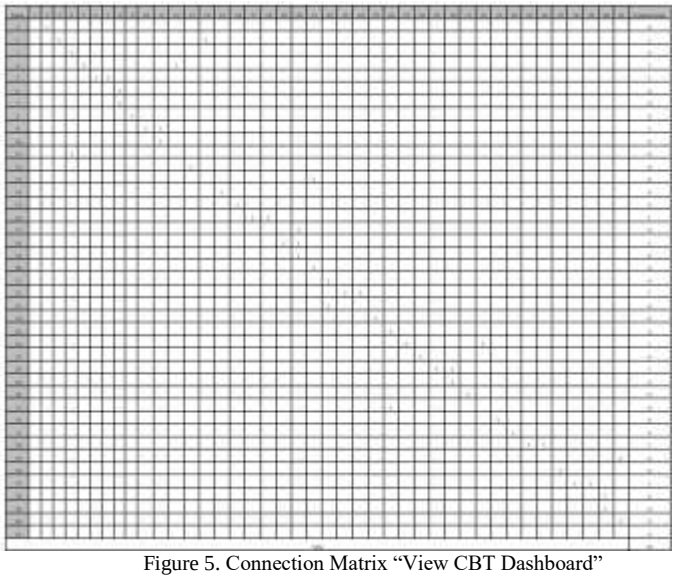

Figure 5. Connection Matrix "View CBT Dashboard"

b) Use Case "View Payment Status"

Figure 6 is control flow graph (CFG) of view View Payment Status functional.

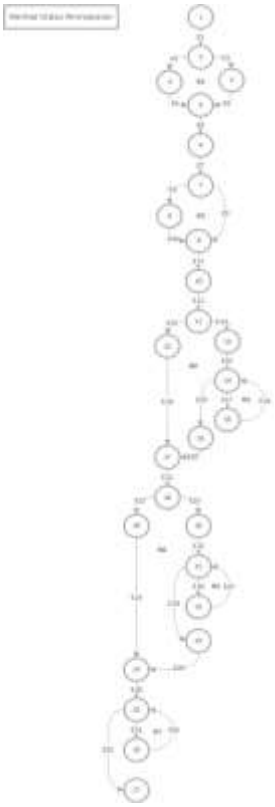

Figure 6. CFG "View Payment Status"

From the CFG above, it is obtained the number of Nodes is 27, the number of Edge is 33 and the number of Predicate Node is 7. So from Formula (1) or (2) if calculated then,

$$V(G) = 33 - 27 + 2 = 8, \text{ or}$$

$$V\ (G) = 7 + 1 = 8$$

After that, graph matrix and connection matrix are created and get the result of V(G) is 7 as follows.

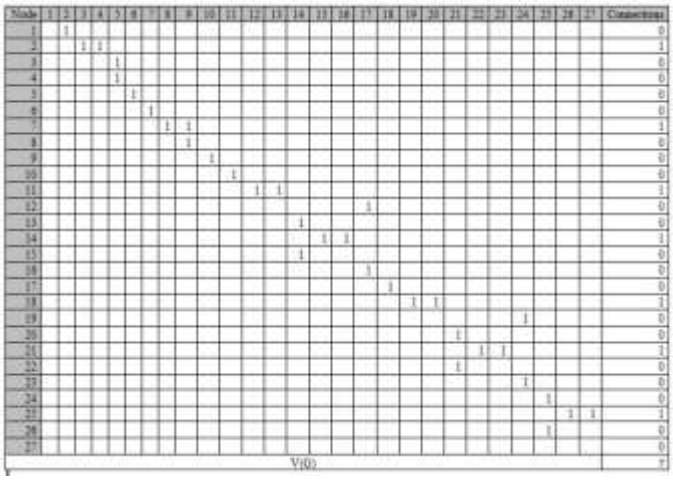

Figure 7. Connection Matrix "View Payment Status"

c) Use Case "Print Registration"

Figure 8 is control flow graph (CFG) of Print Registration.

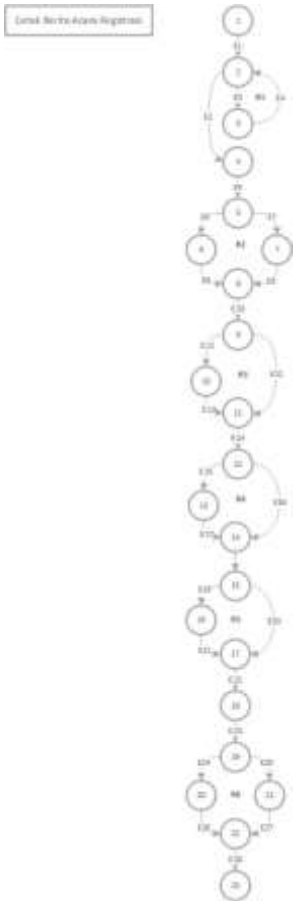

Figure 8. CFG "Print Registration"

From the CFG above, it is obtained the number of Nodes is 23, the number of Edge is 28 and the number of Predicate Node is 6. So from Formula (1) or (2) if calculated then,

$$V(G) = 28 - 23 + 2 = 7, \text{ or}$$

$$V\ (G) = 6 + 1 = 7$$

After that, graph matrix and connection matrix are created and get the result of V(G) is 6 as follows.

ISBN 978-602-53524-0-9

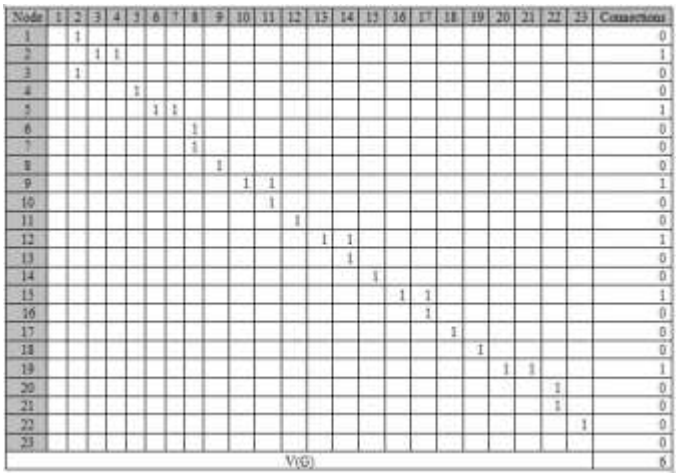

Figure 9. Connection Matrix "Print Registration"

Some facts show that if a program with a cyclometic complexity greater than 10, the program has a higher probability of errors and defects. The Software Engineering Institute classifies risk evaluation according to the cyclometic complexity value as presented in the Table IV [7].

TABLE IV. SOFTWARE RISK EVALUATION

Cyclometic Complexity	Risk Evaluation
1-10	Free from Risk
11-20	Software has medium risk
21-50	Software has high risk
>51	Software has very high risk

So, based on the table above, one functionality in admission module has medium risk toward errors and failure, that is "View CBT Dashboard".

V. CONCLUSION

There is an anomaly on the results of the cyclomatic complexity calculation on white box testing. That is one point difference using formula (1) or (2) by calculating the results of the graph matrix.

In the test "View CBT Dashboard" obtained the fact that V (G)> 10 means that the program code is at medium risk and can still be separated into several program code units.

ACKNOWLEDGMENT

Acknowledgments are delivered to The Ministry of Research, Technology, & Higher Education Indonesia which has funded this research and also Institut Teknologi Telkom Purwokerto especially to Institute of Research and Community Service (LPPM) which has facilitated this research.

.

REFERENCES

[1] IEEE Society, "Guide to the Software Engineering Body of Knowledge Version 3.0", IEEE, 2014.

[2] Chaucan R. Kaur , and Iqbal Singh, "Latest Research and Development on Software Testing Techniques and Tools", *International Journal of Current Engineering and Technology*, Vol 4, No.4, pp 2368 – 2372, 2014.

[3] Galin, Daniel, "Software Quality Assurance", *Pearson Addison Wesley : London*, 2004.

[4] Khan, Mohd. Ehmer, "Different Approach to White Box Testing Techniques for Finding Errors", International Journal of Software Engineering and Its Applications, Vol. 5, No.3, pp 1-14, 2011.

[5] Nidhra, Srinivas, and Jagruthi Dondeti, "Blackbox and Whitebox Testing Techniques – A Literature Review", *International Journal of Embedded System and Applications* (IJESA),Vol.2, No.2, pp 29-50, 2012.

[6] Pranata, Fredy Nendra, Fajar Pradana, and Tri Astoto Kurniawan, "Pengembangan Sistem Perhitungan Kompleksitas Kode Sumber Berdasarkan Metrik Halstead dan Cyclomatic Complexity", *SENTRIN*, pp 27-35, 2016.

[7] Khan, Mohd. Ehmer, "Different Approach to White Box Testing Techniques for Finding Errors", International Journal of Software Engineering and Its Applications, Vol. 5, No.3, pp 1-14, 2011.

AUTHORS PROFILE

Andika Elok Amalia is a lecturer in software engineering department in Institut Teknologi Telkom Purwokerto. She is interested in software process and testing area. She got her Master Degree from Institut Teknologi Bandung

Emi Iryanti is a lecturer in informatics department in Institut Teknologi Telkom Purwokerto. She is interested in usability and multimedia area. She got her Master Degree from Institut Teknologi Bandung

ISBN 978-602-53524-0-9