JURNAL RISET INFORMATIKA **P-ISSN: 2656-1743 |E-ISSN: 2656-1735**
Vol. 3, No. 2 March 2021 DOI: https://doi.org/10.34288/jri.v3i2.59

THE BLACK BOX TESTING AND LOC METHOD APPROACH IN TESTING AND STREAMLINING THE PATIENT REGISTRATION PROGRAM

Joosten

Information System
STMIK Mikroskil
https://www. mikroskil.ac.id
joosten.ng@mikroskil.ac.id

Abstrak

Software yang baik dapat digunakan jika ada pengujian yang tepat. Tahap pengujian cukup penting karena software perlu diuji sebelum digunakan oleh pengguna akhir. Pada pembuatan software untuk rumah sakit hewan belum adanya validasi dan verifikasi sehingga diperlukan pengujian. Penelitian ini menggunakan informasi bagian pendaftaran pasien rumah sakit hewan dan diuji dengan tiga metode Black Box Testing, yaitu Equivalence Class Partitioning (ECP), Boundary Value Analysis (BVA), dan Decision Table ditambah pendekatan LOC. Hasil pengujian dari ketiga metode tersebut adalah persentase ECP yang tidak valid lebih besar dari yang valid, sehingga perlu diubah lagi batas nilai inputnya. Kemudian untuk pengujian BVA, persentase yang valid lebih tinggi daripada tidak valid. Dalam tabel keputusan dibuat aturan pemendekan antara layanan operasi dan layanan lainnya sehingga menghasilkan status rawat inap dan besaran uang muka secara otomatis tanpa harus memilih lagi dan diuji kembali oleh decision table dengan cara mencocokkan hasil estimasi dari kedua layanan tersebut.

Kata kunci: BVA, Decision Table, ECP, Pengujian, Validasi, Verifikasi

Abstract

Good software can be used if there is proper testing. The testing phase is quite important because the software needs to be tested before it is used by end users. In making software for animal hospitals there is no validation and verification so testing is needed. This study used information on the registration section of veterinary hospital patients and was tested by three Black Box Testing methods, namely Equivalence Class Partitioning (ECP), Boundary Value Analysis (BVA), and Decision Table plus the LOC approach. The test results of the three methods are that the percentage of invalid ECPs is greater than the valid ones, so the input value limit needs to be changed again. Then for BVA testing, the percentage of valid is higher than invalid. In the decision table, a shortening rule is made between operating services and other services so that it produces inpatient status and down payment automatically without choose it again and is tested again by the decision table by matching the estimation results of the two services.

Keywords: BVA, Decision Table, ECP, Testing, Validation, Verification

INTRODUCTION

Today technological developments are increasingly developing in this modern world. One of them is software development or what is called software. The widespread software applications of the Internet and mobile computing have significantly increased the dependence on enabled software systems (Xu et al., 2015). One of the stages in software development is software testing. The role of testing activities is very important because testing is one of the activities that must be carried out on software developed before the software is applied by users or end-users. There are three main reasons for the importance of software testing, namely errors or deficiencies in software can occur, the application must be the best, and end-user satisfaction is everything (Solution, 2018). Testing is the process of checking software, both internally and externally. From the internal side, testing is carried out to see the statements that have been tested. While on the outer side, testing is directed to find errors that arise from the software and ensure that the limited inputs can produce the desired actual results according to the expected results.

The increasing demand for software makes the testing aspect always neglected. Testing is an important part of the software development process (Hierons & Member, 2015). The complexity of the software programming logic flow that developers make is one of the problems when entering the testing phase. Developers often find it difficult to get

errors in the software they make due to algorithms that are too complex. Users sometimes don't care if the software has an error or not. Ignoring testing activities in the software results in the results (output) that is displayed sometimes not as expected. Ordinary users will tell the developer to change the program they make in order to produce maximum results and satisfactory program performance.

Joel Spolsky (Spolsky, 2020), a software developer in New York City, wrote an article discussing the top 5 reasons (wrong) that a tester is not needed. Joel is applying for a job at a company led by Mr. Gleick. Joel asked Mr. Gleick about the Pipeline that Gleick created. Joel sees that Gleick's creation has problems such as not using an error correction protocol so that it can damage or crash. But Gleick denies that Pipeline has no bugs and is just as bad as Ms. Word from Microsoft. Both reasons from Gleick kept Joel from applying to Gleick's company.

From Joel Spolsky's case, it can be concluded that the average person ignores the validation, verification, and testing of the software that is made. Neglect of software testing makes the software unable to work optimally and many errors occur. This study discusses several methods that can be used to test software so that the software can work optimally.

The use of excessive costs in software testing causes the quality of the software to be poor. Every year, poor software quality losses exceed $ 500 billion (Shahbazi & Miller, 2016). The National Institute of Standards of Technology (NIST) (Rep, 2002) conducted a study in 2012 that explained the United States economy calculates a loss of $ 59.5 billion every year due to bugs in software created. NIST explains about one-third of these losses can be avoided if the developers do software testing better. More than half the cost of fixing bugs given to software users is given to developers and vendors (Wong et al., 2016).

Besides the use of large costs, another thing that assumes that software testing is not needed is the time spent. In the 1980s, one of the most notorious cases of software development failure was the Ariane rocket (Lynch, 2017). The rocket exploded due to software failure. As a result of the explosion of the rocket, the researchers over the years focusing their efforts on looking for problems or bugs in the software on the rocket.

Rashad Khalid (Khalid, 2017) conducted a study that focused on the efficiency side of software testing by combining two methods, namely the black box testing method with the white box testing method. Khalid developed an automated analysis and testing technique consisting of two main steps: 1) the first step was to take the software under test (sut) and identify all files, input and output variables, and functions. 2) in the next step, the user selects the desired module part or function to test and selects the required tests such as dead-code, assertion based tests and exception tests. The programming language khalid tested was the c ++ programming language. Khalid claims the tools he recommends can perform various tests such as static analysis, unit testing, dead-code testing, exception testing and assertion based testing. But the program code that khalid tested was only a simple program and did not tell from the efficiency side whether the recommended tool could handle the limited testing time or not so that the tool did not guarantee the success of the test when the program code was quite complex.

Christopher Dimas Satrio et al (Satrio et al., 2018), conducted a study comparing a test case generation with two genetic algorithm approaches, namely mutual analysis and sampling. The two approaches will be compared from the reduction in test cases that occur. Apart from reducing the test case, the number of iterations, the total number of individuals, the number of fitness evaluations, and the size of the test suites will also be a comparison between the two approaches. From the two approaches, it was found that the genetic algorithm mutual analysis approach was better than the genetic algorithm sampling in terms of the number of test case reductions and all the comparative variables applied. However, in terms of execution time, it is still assumed that the genetic algorithm mutual analysis is faster, so there has been no decision that the genetic algorithm mutual analysis approach is faster when executing larger sample data. Researchers also concluded that to increase the effectiveness and efficiency in software testing by shortening the time that must be allocated for the testing carried out. But the researchers did not explain how long the time should be allocated, so they had to see how complex the software to be tested was.

Marcel Bohme and Soumya Paul (Bohme & Paul, 2016), conducted a study that analyzed the efficiency between random testing (r) and systematic testing (S_0). They built a general model for software testing by specifying a sampling strategy on random testing and systematic testing associated with cost and sampling time. They considered two such strategies whereby random testing was unaware of partition based errors and systematic testing that sampled each partition exactly once. They perform calculations on these two strategies to calculate the relative efficiency value. In the end they conducted 24000 simulation experiments with various parameters. However, in

P-ISSN: 2656-1743 |E-ISSN: 2656-1735
DOI: https://doi.org/10.34288/jri.v3i2.59

this study there are recommendations from researchers to compare systematic testing techniques to random techniques with a given time limit practically because they assume random techniques must have the same "power" compared to systematic techniques so that there may be other testing techniques that can overcome the time required. Limited compared to using random techniques.

RESEARCH METHODS

This study uses methods to verify, validate, and test the hospital information system program. The methods used are Equivalence Partitioning (EP), Boundary Value Analysis (BVA), and Decision Table added with the Line of Code (LOC) approach to count the number of rows before using the Decision Table and after using the Decision Table. These three methods are part of the Black Box Testing method. Black Box Testing is a software testing technique that focuses on the functional specifications of the software being developed (Jaya, 2018). The way Black Box Testing works is only to check the output value based on the software input value without knowing what program code is used.

Equivalence Partitioning (EP) or what is often called Equivalence Class Partitioning (ECP) is a technique or method that produces test data from several system requirements by grouping or dividing input data and testing the data in order to get several understandable and feasible categories (Jahanbin & Zamani, 2018). From these methods, there are several combinations that can occur in Equivalence Partitioning, including:

a. Valid and invalid input values
b. A numeric value that is negative, positive, or zero
c. An empty, non-empty string

Boundary Value Analysis (BVA) (Ardana, 2019) is a technique of the Black Box Testing method that focuses on the input process by testing the values at the upper and lower limits. There are three principles that underlie the Boundary Value Analysis (BVA) method are:

a. Many errors occur with input errors
b. Select a test case that tests the limits of input values
c. BVA is a part of Equivalence Partitioning that selects elements in the equivalence class at the value limit.

Decision Table is one of the techniques of the Black Box Testing method that uses tables to perform testing and can also be used to shorten the logic flow of software programming. Decision Table Testing (Joosten et al., 2020) is a software testing technique used to test software on different input combinations by combining different input and output values and summarizing them into a table. Decision Table is also often referred to as a cause and effect table because of the several causes and effects used to create a Decision Table.

The reason the decision table is quite important is as follows (Joosten et al., 2020):

1. Very helpful in test design techniques.
2. Help testers to look for the effect of the combination of various inputs and the status of other software that must implement business rules properly.
3. Provide a regular way to express complex business rules, which is beneficial for both developers and testers.
4. Assist in the development process with a developer to do a better job. Testing with all combinations might not be practical.
5. This method is basically a technique used in testing and managing requirements.
6. This method is a structured exercise to prepare the requirements when dealing with complex business rules.
7. It can also be used in complex logic models.

Line of code or what is often referred to as the source line of code (SLOC) is a software metric that is often used to measure the size and complexity of a software project. There are two main types of SLOC calculations, namely Physical SLOC (P-SLOC) and Logical SLOC (L-SLOC). The definition of P-SLOC is the line count in the source code text of the program as well as comment lines and blank lines.

Meanwhile, L-SLOC is responsible for measuring the number of expressions that can be executed. The intended expressions are operators, functions, etc. So if there is one or more statements which are followed by the end-of-line comment is a line of code and is counted into L-SLOC. Meanwhile, comment lines and blank lines will not be counted into L-SLOC. In addition to Physical SLOC, Logical SLOC, comment line of code (CLOC), blank line of code (BLOC), there are also code & comment source line of code (C & SLOC) and comment words (CWORD). To find out the calculation of the number of lines in a source code such as P-SLOC, L-SLOC, and others, below is an example of source code:

```
#include <iostream>
#include <conio.h>
#include <windows.h>
using namespace std;
int main ()
{
char lagi;
int name;
int option;
int number;
int paid;
```

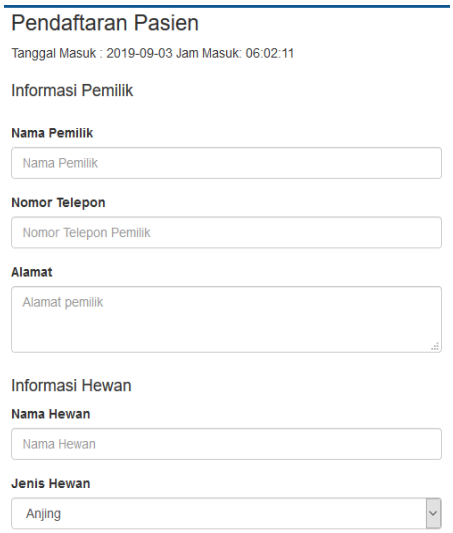
```
int price;
int total;
int code;
// create menu list
// cout << endl ends list;
early:
system("cls");
cout<<"======================"<<endl;
cout<<"+++++++      Keday      kopi      balog
+++++++"<<endl;
cout<<"====Menu========Harga=="<<endl;
cout<<"1. Black coffee Rp.10000"<<endl;
cout<<"2. Coffee mocca  Rp.15000"<<endl;
cout<<"3. Coffee milk   Rp.9000"<<endl;
cout<<"4. Love coffee   Rp.20000"<<endl;
cout<<"5. Pineapple juiceRp.10000"<<endl;
cout<<"6. Avocado juice Rp.12000"<<endl;
cout<<"7. Orange juice Rp.10000"<<endl;
cout<<endl; //end

cout<<"Enter Your Selection (1-7) =";
cin>>code;
```

From the sample source code above, the number of P-SLOC is 29, the number of L-SLOC is 24, the number of BLOC is 1, the number of CLOC is 2. In the source code above, there are C & SLOC and CWORD. The part which is C & SLOC is 'cout<<endl;//selesai'. whereas for CWORD it is 'create', 'list', 'menu', 'cout<<endl', 'ends', 'list;', and 'end'

Research Material

This study uses html and php files from a medical information system. The medical record program will be tested first by checking the patient number on the medical record. After being tested properly, the medical record program can be used in the hospital. The contents of the html and php files are part of the patient registration form which is part of the medical information system. The initial display of the registration form on the medical information system is shown in Figure 1.



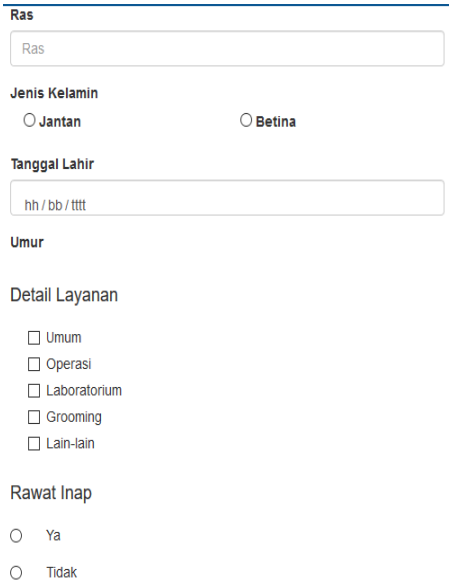


Figure. 1. Patient registration section

In this method, the equivalence class partitioning will test the input value of the registration form to determine whether the form input is valid or not. Then the Boundary Value Analysis method tests the input value limits on the registration form. Then, in the detail section, services and inpatient services are processed using the decision table and LOC approaches to calculate the estimated number of rows before using the decision table and after using the decision table. The data used in the decision table is shown in table 1:

Table 1 Dataset of Operating services

Condition Stub	*Operating Services*	*Scalling*
		Castrasi
		OH
		Sectio Caecaria
		Ortopedica
	Services	*Public*
		Lab
		Grooming
		ETC
Action Stub		*no need for hospitalization*
		infectious space hospitalization
		non-infectious space hospitalization
		hospitalized healthy room
		advance payment Rp. 3.000.000
		advance payment Rp. 3.500.000
		advance payment Rp. 2.500.000

RESULTS AND DISCUSSION

This study uses three methods of Black Box Testing, namely Equivalence Class Partitioning, Boundary Value Analysis, and Decision Table coupled with the LOC approach to estimate the calculation of the number of program lines. To find out the percentage of valid and invalid test cases, this study used the standard testing metrics formula (Akshatha & Illango, 2018).

$$Test\ case\ Pass\ \% = \left(\frac{No.of\ test\ case\ passed}{Total\ No\ of\ Cases}\right) x\ 100 \ldots\ldots (1)$$

From the equivalence class partitioning method, the test results are obtained as in the Table 2 below:

Table 2 Testing the Equivalence Class Partitioning

No	Menu Elements	Rule	Data used	E. Result	Result	Statue
1	Nama Pemilik	Min input of 1 char	a	data is not saved	stored data	Fail
2	Nama Pemilik	Input 3-20 char	adi	stored data	stored data	Pass
3	Nama Pemilik	Input 3-20 char, strings & numbers	Ad1n$e	data is not saved	stored data	Fail
4	Nomor Telepon	Min input of 1 number	1	data is not saved	stored data	Fail
5	Nomor Telepon	Input 10 – 13 number	0812345679	stored data	stored data	Pass
6	Nomor Telepon	Input other than numbers	Asds$@	data is not saved	stored data	Fail
7	Alamat	Min input 1 char	a	data is not saved	stored data	Fail
8	Alamat	Input at least 10 char & numbers	Jl. Kaliurang no 70	stored data	stored data	Pass
9	Alamat	Input char, numbers, & strings	Jl. K@l1urang no 118	data is not saved	stored data	Fail
10	Nama Hewan	Min input 1 char	a	data is not saved	stored data	Fail
11	Nama Hewan	Input 3-25 char	Leo	stored data	stored data	Pass
12	Nama Hewan	Input 3 – 20 char, string, & num	@sing	data is not saved	stored data	Fail
13	Ras	Min input 1 char	a	data is not saved	stored data	Fail
14	Ras	Input 3-25 char	Kampung	stored data	stored data	Pass
15	Ras	Input 3 – 20 char, string, & numbers	@sing	data is not saved	stored data	Fail
16	Jenis Kelamin	Please select a gender	Jantan	stored data	stored data	Pass
17	Jenis Kelamin	Not choose	-	Asked to choose	Must choose a gender	Pass
18	Tanggal Lahir	Input number	03-04-1990	stored data	stored data	Pass
19	Tanggal Lahir	Input other than numbers	sadef@#$	Cannot be typed	Must be number	Pass

From the test results in Table 2, there were 19 test cases consisting of 9 valid or pass test cases and 10 invalid or failed test cases. So that the percentage of valid test cases is 47.4% and the percentage of invalid test cases is 52.6%.

In the BVA method, test results are obtained as in Table 3:

Table 3 Testing the Boundary Value Analysis

No	Menu element	Rule	Data used	E. Result	Result	Status
1	Nama Pemilik	Empty it		Data is not saved	The message "Please fill in" appears	Pass
2	Nama Pemilik	Min input of 1 char	a	Data is not saved	Stored data	Fail
3	Nama Pemilik	Min input of 3 char	adi	Stored data	Stored data	Pass
4	Nomor Telepon	Empty it		Data is not saved	The message "Please fill in" appears	Pass
5	Nomor Telepon	Min input of 1 number	1	Data is not saved	Stored data	Fail
6	Nomor Telepon	Input 10 – 13 number	0812345679	Stored data	Stored data	Pass
7	Nomor Telepon	Input more than 13 numbers	09123456789192	Data is not saved	Stored data	Fail
8	Alamat	Empty it		Data is not saved	The message "Please fill in" appears	Pass
9	Alamat	Min input of 1 char	a	Data is not saved	Stored data	Fail
10	Alamat	Input at least 10 char & numbers	Jl. Kaliurang no 70	Stored data	Stored data	Pass
11	Nama Hewan	Empty it		Data is not saved	The message "Please fill in" appears	Pass
12	Nama Hewan	Min input of 1 char	a	Data is not saved	Stored data	Fail
13	Nama Hewan	Input 3-25 char	Leo	Stored data	Stored data	Pass
14	Nama Hewan	Input above 25 char	Sadhadsgyqwewwqegwuqegwrwqrwqrwqr	Data is not saved	Stored data	Fail
15	Ras	Empty it		Data is not saved	The message "Please fill in" appears	Pass
16	Ras	Min input of 1 char	a	Data is not saved	Stored data	Fail
17	Ras	Input 3-25 char	Kampung	Stored data	Stored data	Pass
18	Ras	Input above 25 char	@sing	Data is not saved	Stored data	Fail
19	Jenis kelamin	Empty it		Data is not saved	The message "Please fill in" appears	Pass
20	Jenis Kelamin	Please select a gender	Jantan	stored data	stored data	Pass
21	Tanggal Lahir	Input number	03-04-1990	stored data	stored data	Pass
22	Tanggal Lahir	Input other than numbers	sadef@#$	Cannot be typed	Must be number	Pass

From the test results in Table 3, there are 22 test cases consisting of 14 valid or pass test cases and 8 invalid or failed test cases. So that the percentage of valid test cases is 63.6% and the percentage of invalid test cases is 36.4%.

For the decision table method, the part that is processed and tested is the operation service section and the selection of inpatients. In both parts, a rule shortening process is carried out so that when choosing an operating service with another service, it will be notified whether it needs to be switched in or not. The result of the shortening rule from the decision table is shown in figure 2.

P-ISSN: 2656-1743 |E-ISSN: 2656-1735
DOI: https://doi.org/10.34288/jri.v3i2.59

1. Operating Services	Scaling			Castrasi		OH			Sectio Caecaria		Ortopedica		
2. Services	Public	Grooming	Lab or ETC	Public or Grooming	Lab or ETC	Public	Grooming	Lab or ETC	Public	Grooming or Lab or ETC	Public	Grooming	Lab or ETC
1. no need for hospitalization	.	x	.	.	x	.	x	.	.	x	.	.	x
2. infectious space hospitalization	x	.	.	.	.	x	.	.	.	.	x	.	.
3. non-infectious space hospitalization	.	.	x	.	.	.	.	x	.	.	.	.	.
4. hospitalized healthy room	.	.	.	x	.	.	.	.	x	.	.	x	.
5. Advance Payment Rp. 3.000.000	x	.	.	.	.	x	.	.	.	.	x	.	.
6. Advance Payment Rp. 3.500.000	.	.	x	.	.	.	.	x	.	.	.	.	.
7. Advance Payment Rp. 2.500.000	.	.	.	x	.	.	.	.	x	.	.	x	.
	1	2	3	4	5	6	7	8	9	10	11	12	13

Figure. 2. The result of the decision table

From the results of the shortened rule, when implemented into the algorithm, the number of algorithm lines is reduced due to the algorithm being changed and the result of the number of lines of code is shown in Table 4.

Table 4 Comparison of LOC before and after the decision table

	Before use *Decision Table*	**After use *Decision Table* (Estimation)**
LOC	466	425
P-SLOC	456	415
L-SLOC	111	111
BLOC	6	6
C&SLOC	1	1
CLOC	4	4
CWORD	10	10

After getting the final amount of LOC, the next step is to calculate the LOC efficiency. The formula for finding the LOC efficiency is:

$$\frac{Initial\ LOC\ amount - Final\ LOC\ amount}{Initial\ LOC\ amount} x100\% \quad \text{.....(2)}$$

In the LOC calculation, the results are as follows:

$$\text{Efficiency of LOC} = \frac{466-425}{466} x100\% = 8{,}8\% \quad \text{.....(3)}$$

In the P-SLOC calculation, the results are as follows:

$$\text{Efficiency of P-SLOC} = \frac{456-415}{456} x100\% = 9\% \quad \text{.....(4)}$$

In testing using the decision table method, there are 32 test cases from the Scaling operation service with other services. When the input of Scaling operating services is True and Public services is True, the output expectation is infectious space hospitalization and Advance Payment of Rp. 3,000,000. When the input of Scaling operating services is True and Grooming services is True, then the output expectation is no need for hospitalization. When the input of Scaling operating services is True and Lab or ETC services is True, then the expected output is non-infectious space hospitalization and Advance Payment of Rp. 3,500,000. The test result becomes an error if other input is outside of the four test cases above. Likewise with testing Castrasi operation services with other services.

CONCLUSIONS AND SUGGESTIONS

Conclusion

This study is a continuation of previous research that only shortened the rule by using a decision table (Joosten et al., 2020). From the results of the research above, the percentage of valid ECPs is lower than invalid ones, so the software needs to be fixed. For the BVA test results, the percentage of valid BVA is higher than the invalid one and can be increased again which is the valid part so that the BVA test is even better. For testing using the decision table and LOC approach, after changes to the code part of the program were made, the number of LOCs was reduced and the LOC efficiency was 8.8% and the P-SLOC was 9% so that it became efficient. In addition to reducing rules, testing is also carried out on the operation service section with other services.

Suggestion

From this research it is hoped that there will be other methods that can be combined with the three methods so that it is more efficient and the testing is getting better so that it does not take a long time and is expensive

REFERENCES

Akshatha, V., & Illango, V. (2018). A Comparative Analysis On Equivalence class partitioning And Boundary value analysis. *International Journal of Recent Trends in Engineering and Research*, *4*(3), 542–554. https://doi.org/10.23883/ijrter.2018.4163.2aczv

Ardana, I. M. S. (2019). Pengujian Software Menggunakan Metode Boundary Value Analysis dan Decision Table Testing. *Jurnal Teknologi Informasi ESIT*, *14*(11), 40–47.

Bohme, M., & Paul, S. (2016). A Probabilistic Analysis of the Efficiency of Automated Software Testing. *IEEE Transactions on Software Engineering*, *42*(4), 345–360. https://doi.org/10.1109/TSE.2015.2487274

Hierons, R. M., & Member, S. (2015). for Distributed Testing. *IEEE Transactions on Software Engineering*, *41*(3), 279–293.

Jahanbin, S., & Zamani, B. (2018). Test model

generation using equivalence partitioning. *2018 8th International Conference on Computer and Knowledge Engineering, ICCKE 2018*, *Iccke*, 98–103. https://doi.org/10.1109/ICCKE.2018.8566335

Jaya, T. S. (2018). Pengujian Aplikasi dengan Metode Blackbox Testing Boundary Value Analysis (Studi Kasus: Kantor Digital Politeknik Negeri Lampung). *Jurnal Informatika Pengembangan IT (JPIT)*, *3*(2), 45–46. http://www.ejournal.poltektegal.ac.id/index.php/informatika/article/view/647/640

Joosten, J., Permanasari, A. E., & Adji, T. B. (2020). The use of decision table for reducing complex rules in software testing. *IOP Conference Series: Materials Science and Engineering*, *732*(1). https://doi.org/10.1088/1757-899X/732/1/012086

Khalid, R. (2017). Towards an automated tool for software testing and analysis. *Proceedings of 2017 14th International Bhurban Conference on Applied Sciences and Technology, IBCAST 2017*, 461–465. https://doi.org/10.1109/IBCAST.2017.7868094

Lynch, J. (2017). *The Worst Computer Bugs in History: The Ariane 5 Disaster*. Https://Www.Bugsnag.Com/Blog/Bug-Day-Ariane-5-Disaster. https://www.bugsnag.com/blog/bug-day-ariane-5-disaster

Rep, N. (2002). *The Economic Impacts of Inadequate Infrastructure for Software Testing*. Http://Www.Abeacha.Com/NIST_press_release_bugs_cost.Htm. http://www.abeacha.com/NIST_press_release_bugs_cost.htm

Satrio, C. ., Saputra, M. ., & Rachmadi, A. (2018). Perbandingan test case generation dengan pendekatan genetic algorithm mutation analysis dan sampling. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer*, *2*(1).

Shahbazi, A., & Miller, J. (2016). Black-Box String Test Case Generation through a Multi-Objective Optimization. *IEEE Transactions on Software Engineering*, *42*(4), 361–378. https://doi.org/10.1109/TSE.2015.2487958

Solution, Q. (2018). *Masih Bertanya Kenapa Anda Harus Melakukan Testing Pada Software? Ini Alasannya"*. Http://Www.Quadras.Co.Id/2016/05/25/Masih-Bertanya-Kenapa-Anda-Harus-Melakukan-Testing-Pada-Software-Ini-Alasannya/. http://www.quadras.co.id/2016/05/25/masih-bertanya-kenapa-anda-harus-melakukan-testing-pada-software-ini-alasannya/

Spolsky, J. (2020). *Top Five (Wrong) Reasons You Don't Have Testers*. Https://Www.Joelonsoftware.Com/2000/04/30/Top-Five-Wrong-Reasons-You-Dont-Have-Testers/. https://www.joelonsoftware.com/2000/04/30/top-five-wrong-reasons-you-dont-have-testers/

Wong, W. E., Gao, R., Li, Y., Abreu, R., & Wotawa, F. (2016). A survey on software fault localization. *IEEE Transactions on Software Engineering*, *42*(8), 707–740. https://doi.org/10.1109/TSE.2016.2521368

Xu, D., Xu, W., Kent, M., Thomas, L., & Wang, L. (2015). An automated test generation technique for software quality assurance. *IEEE Transactions on Reliability*, *64*(1), 247–268. https://doi.org/10.1109/TR.2014.2354172