International Journal of Electrical and Computer Engineering (IJECE)
Vol. 8, No. 5, October 2018, pp. 3333~3340
ISSN: 2088-8708, DOI: 10.11591/ijece.v8i5.pp3333-3340

Determining Basis Test Paths Using Genetic Algorithm and J48

Achmad Arwan, Denny Sagita
Department of Informatic Engineering, University of Brawijaya, Indonesia

Article Info

Article history:

Received Jun 22, 2017
Revised Jan 22, 2018
Accepted Feb 11, 2018

Keyword:

Basis test paths
Code metrics
Genetic algorithm
J48
Software tests

ABSTRACT

Basis test paths is a method that uses a graph contains nodes as a representation of codes and the lines as a sequence of code execution steps. Determination of basis test paths can be generated using a Genetic Algorithm, but the drawback was the number of iterations affect the possibility of visibility of the appropriate basis path. When the iteration is less, there is a possibility the paths do not appear all. Conversely, if the iteration is too much, all the paths have appeared in the middle of iteration. This research aims to optimize the performance of Genetic Algorithms for the generation of Basis Test Paths by determining how many iterations level corresponding to the characteristics of the code. Code metrics Node, Edge, VG, NBD, LOC were used as features to determine the number of iterations. J48 classifier was employed as a method to predict the number of iterations. There were 17 methods have selected as a data training, and 16 methods as a data test. The system was able to predict 84.5% of 58 basis paths. Efficiency test results also show that our system was able to seek Basis Paths 35% faster than the old system.

Corresponding Author:

Achmad Arwan,
Department of Informatic Engineering,
Faculty of Computer Science (FILKOM), Brawijaya University,
8th Veteran Road | Malang, 65145 - Indonesia.
Email: arwan@ub.ac.id

1. INTRODUCTION

Software testing is a process that is performed to determine whether a program code is free from errors or not. Testing can be done by various methods (eg. regression test [1], black box, and white box). White Box Testing is a testing method that uses the source code as the basis of knowledge in finding code defects [2]. To be able to do the White Box testing, the source code and then converted into the form of Graph called the Control Flow Graphs (CFG) [3]. CFG contains nodes that represent commands in a code/pseudo code. Node is a feature to search for the number of statements in Java code [3]. Edge is the liaison between the nodes to one another [3]. DD-Graph (decision-to-decision Graph) is a refinement of CFG where not all the code was made into a graph, but only the beginning of the code until you find the branching conditions are recruited graph[4]. DD-Graph then used as knowledge to the test scenario. Testing is done by trying all the existing path on the DD-Graph from beginning to end with a code assign values to variables that exist in the node. This method is then called the basis path testing.

In the white box testing, there is a path that must be passed/tested at least once to make sure there are no errors in the generated code. To obtain these basis paths can be done manually or automatically. Swarm intelligence takes part in solving test problems. Some study was used Ant Colony Optimization to reduce test case[5]. Others previous study provided recommendations independent paths automatically by using Genetic Algorithms [6],[7],[8]. Combination of Genetic Algorithm with Greedy Algorithm also was used to determine pairwise testing case [9]. The study was able to recommend an independent path on the Basis Path Testing after certain iterations. Other research is also using slicing technology which applied on CFG to create regression test [10]. Model J48 is a development method of decision tree algorithm ID3 [11].

The J48 algorithm can classify the data with decision tree method has its advantages can process numerical data (continuous) and discrete, can deal with missing attribute values, generates rules easier to interpret, and the fastest of algorithms that use main memory in computer [11].

One of the factors which determine the success of basis path discovery using genetic algorithm is the number of population and iteration. Sometimes for a specific code appearing all basis paths, but for other cases its not even appear. For example for a specific code is given 30 iterations, but until the last iteration, not all combinations of independent basis path appears all. This condition we called less iteration. The solution of the problem by increasing the number of iterations of mutations/crossover so that basis paths appear. Another example for a specific code we assigned 100 iterations but until 70th iterations than all basis paths have emerged all so that this condition is called over iteration.

The uncertainty of the number of iterations to make the performance of research belongs Ghiduk be less because the user should try to include the number of iterations until an basis path appears. Optimization of the use of genetic algorithms in the search for an basis path to do with predicting the exact number of iterations, so users do not need to try the various number of iterations to get the appearance of all basis paths. Some metrics can be used as a feature in predicting the exact number of iterations. NBD metric was a metric that calculates the depth of complexity in structure if [12], LOC (Line of Code) was a metric to calculate how many the line of the code [12]. Node and Edge as mention earlier also were used as features. V (G) was a complexity metric that measures the amount of branching in the if the Java code [13]. The larger the V (G), the more complex the code.

This research proposes finding of the basis path by adopting the Ghiduk's methods (Based on Genetic Algorithm) and determine the exact number of iterations corresponding to the characteristics of code metrics LOC, NBD, Node, Edge, V (G) using the J48 decision tree. So there is no longer such a thing over iteration or fewer iteration conditions, and the impact will increase accuracy and efficiency when generating basis test paths.

2. RESEARCH METHOD

In order to achieve the goals, the methods used in this study are illustrated in Figure 1. There is a five process which used on this research. All of them would explain in section 2.1 to 2.5. All experiments were conducted in computer laboratory of the University of Brawijaya.

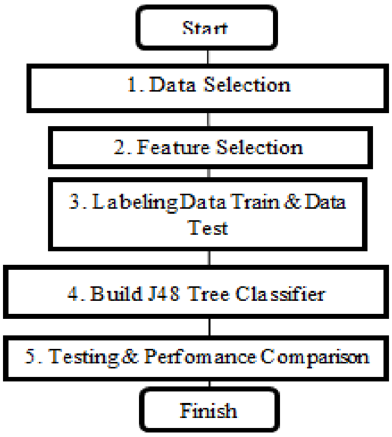

Start
1. Data Selection
2. Feature Selection
3. Labeling Data Train & Data Test
4. Build J48 Tree Classifier
5. Testing & Perfomance Comparison
Finish

Figure 1. Research methodology

2.1. Data Training and Testing Selections

The first process was data training and testing selections. This process aims to refine Java code which was used as the dataset. These files were taken from http://freesourcecode.net/ and some are from our fellow lecturer in the department of software engineering FILKOM University of Brawijaya. Total files were 50 files with the number of LOC 7600. These files may then be selected to be used as the dataset. Selection of data has done by considering the number of lines of code, code complexity, the number of regions, the number of nodes, NBD metrics. Source codes were varied ranging from small (5-20 LOC), medium (40-50

LOC), and large (>100 LOC) on each method. After selection total of a file was decreased into 11 Java files which contain 700 lines of codes and 33 methods.

2.2. Feature Selections

The second process was Feature Selection. This process aims to select which features will be taken from codes. Feature selection has done by using the metrics which were used on Ghiduk's research such as E (Edge), N (Node), V (G) (Complexity). The logical reason was this research were using Ghiduk's method. We have also added LOC and NBD metrics to sharpened prediction. The logical reason was both metrics have correlation regarding complexity.

2.3. Labeling Data Train & Data Test

The third process was labeling data training and data test. We did this process manually by executing all process to each of data. This process was depicted in Figure 2. The First step was calculating E, N, V (G), LOC and NBD on every method in the Java file classes. The second step was set initial iteration with 20 and put label L (Low). We assumed iteration less than 20 is categorized as Low iteration. The third process was generating basis path using Ghiduk's approach with iteration as parameters. So the first time of this process will find basis path with 20 iterations. If all paths have emerged, the process continued to process number seven, otherwise went to process number four increase the value of iterations with 50 and set the label with M (Medium). We assumed that 50 iterations were categorized as a medium.

Step went to the third process to generate some of the basis paths. After that, we checked whether all path have emerged or no? If yes next process would go to process number seven, otherwise went to the 5th process which is increasing of iterations into 100 and label H (High). We assumed 100 iterations as a high number of iteration because it took much time to compute to find basis paths. When iterations have reached 100 and all basis path have not emerged, we eliminate data because we assume its too complex to solve. The 7th process was save the name of the method, E, N, V (G), LOC, NBD, Label into a file which would be used as data training and data testing. Figure 3 is the sample of the content files.

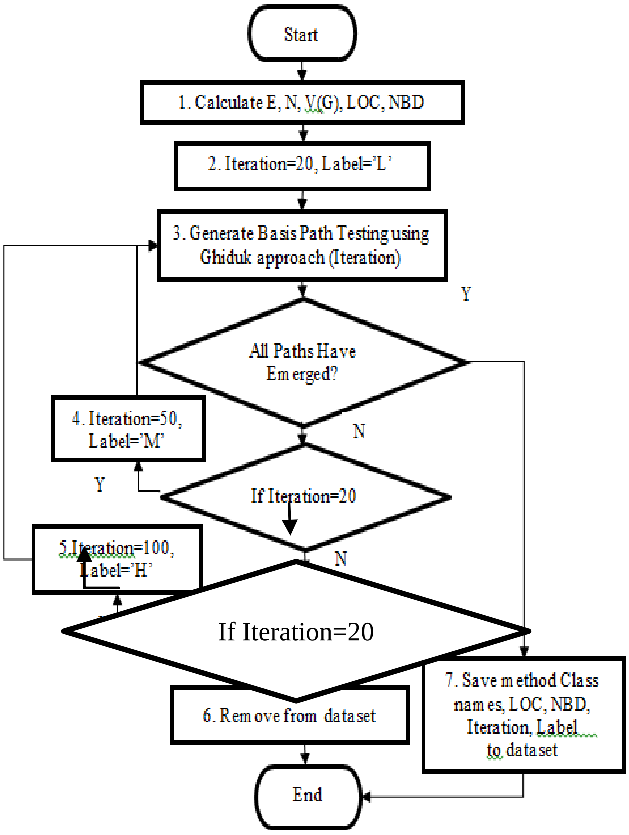
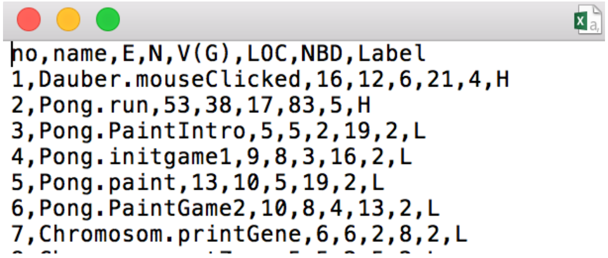

Start
1. Calculate E, N, V(G), LOC, NBD
2. Iteration=20, Label='L'
3. Generate Basis Path Testing using Ghiduk approach (Iteration)
Y
All Paths Have Emerged?
4. Iteration=50, Label='M'
N
Y
If Iteration=20
5.Iteration=100, Label='H'
N
If Iteration=20
7. Save method Class names, LOC, NBD, Iteration, Label to dataset
6. Remove from dataset
End

Figure 2. Features selection



```
no,name,E,N,V(G),LOC,NBD,Label
1,Dauber.mouseClicked,16,12,6,21,4,H
2,Pong.run,53,38,17,83,5,H
3,Pong.PaintIntro,5,5,2,19,2,L
4,Pong.initgame1,9,8,3,16,2,L
5,Pong.paint,13,10,5,19,2,L
6,Pong.PaintGame2,10,8,4,13,2,L
7,Chromosom.printGene,6,6,2,8,2,L
```

Figure 3. Sample of content of data test and data training files

The results of this process were 33 methods which were then divided into 2 parts. The First part contains 17 methods which were used as data training set depicted on Table 1. The Second part contains 16 methods which were used as data test set depicted on Table 2.

Table 1 Data Train Set

NO	NAME	E	N	V(G)	LOC	NBD	LABEL
1	Edge.printByNode	6	6	2	8	2	L
2	Edge.getEdgeByNode	6	6	2	8	2	L
3	Edge.getSibling	9	7	4	14	3	L
4	Edge.addsucessor	7	6	3	8	2	L
5	Pong.PaintIntro	5	5	2	19	2	L
6	operasiGenetik.rankChromosome	11	9	4	12	4	M
7	DDG.SameEdge	10	8	4	7	3	L
8	DDG.closingBlockEdge	8	7	3	18	3	L
9	DDG.AllEdges	5	5	2	9	2	L
10	DDG.buildExitEdges	11	9	4	107	4	L
11	DDG.printNodeList	6	6	2	10	2	L
12	DDG.setEdgeSucessorWhile	7	6	3	93	3	L
13	DDG.setNodeList	7	6	3	7	3	L
14	PorterStemmer.Step3	84	56	30	41	2	H
15	PorterStemmer.Step4	31	22	11	19	2	H
16	PorterStemmer.m	31	22	11	31	2	H
17	TrafficSimulation.init	6	6	2	25	2	L

Table 2. Data Test Set

NO	NAME	E	N	V(G)	LOC	NBD	LABEL
1	dauber.mouseclicked	16	12	6	21	4	H
2	dauber.paint	10	8	4	5	3	L
3	edge.checkedgesucesor	8	7	3	8	2	L
4	operasigenetik.cekkeberadaan	8	7	3	3	3	L
5	operasigenetik.copychromosome	6	6	2	3	2	L
6	operasigenetik.selectbest	8	7	3	13	3	L
7	ddg.printnodelist	6	6	2	10	2	L
8	pong.run	53	38	17	83	5	H
9	pong.initgame1	9	8	3	16	2	L
10	pong.paint	13	10	5	19	2	L
11	pong.paintgame2	10	8	4	13	2	L
12	chromosom.printgene	6	6	2	8	2	L
13	chromosom.setzero	5	5	2	5	2	L
14	chromosom.isequal	8	7	3	10	3	L
15	chromosom.generaterandom	6	6	2	11	3	L
16	porterstemmer.step5	97	62	37	35	1	H

2.4. Build J48 Tree Classifier

After data training and testing have labeled, next process was to build J48 tree classifier based on data training. We built J48 tree classifier using WEKA library [14]. The J48 tree is depicted in Figure 4. Based on J48 tree view, we can conclude that V (G) has much information rather than another metric. For V (G) less or equal than four, there was 14 number of data which have complexity L (low) with one misclassified. For V (G) more than 4, there was 3 number of data which have complexity H (High) with no misclassified data.

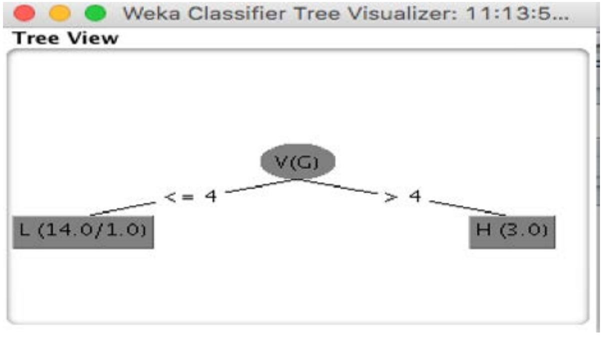


Figure 4. J48 classifier tree view

2.5. Testing & Performance Comparison

We have developed an application to reach our goals. This application based on Java language which contains several libraries (Software Architecture depicted in Figure 5). The input was Java File with specific Class and method, and as the output was basis paths. We have used Spoon to implement Ghiduk's methods . We have employed Spoon to parse Java code into visualizing model and build the graph [15]. We also have used the plugin on Netbeans which called SourceCode Metric to calculate Node, Edge, and V (G) within the Java code. We also have used WEKA to classify the number of iterations based on J48 tree classifier. To predict the label of data test, we have used model classifier from data train (Section 2.4). We also used Ghiduk's method (based on Genetic Algorithm) to generate basis path using specific iteration which determined by J48 classifier. The user interface of our system was depicted in Figure 6.

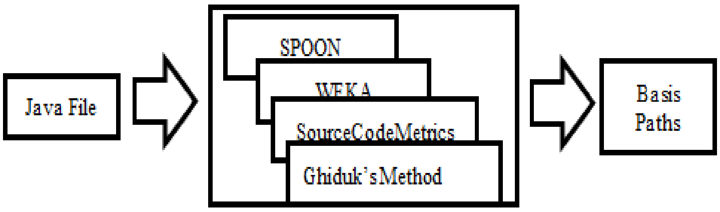


Figure 5. Software architecture

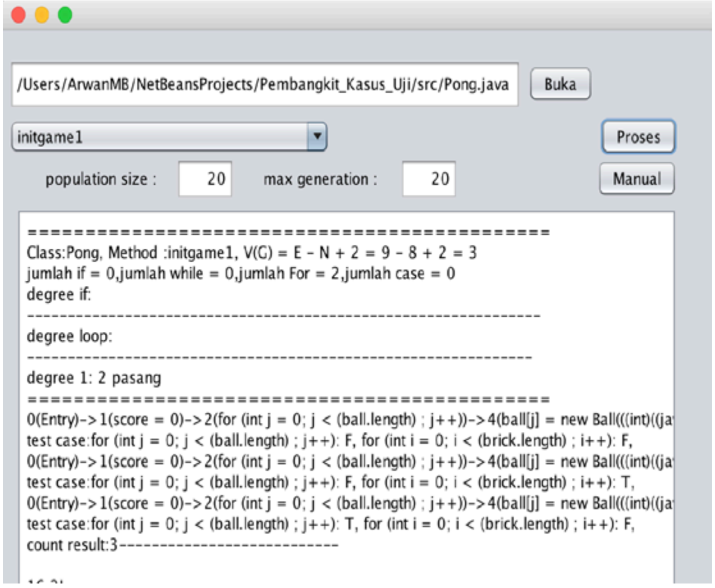
To evaluate the results of our research, we have used measurements of accuracy and efficiency of time. Accuracy means, how the system is able to recommend correct and complete basis path. To measure accuracy we calculate using the following equations.

$$Accuracy = \frac{System\ Path}{Manual\ Path}\ X\ 100\ \% \quad (1)$$

System Path variable was the number of basis paths which have discovered by our system, meanwhile Manual Path was the number of paths which have discovered by our manual calculation. In order to measure performance improvements, we compare our system against Ghiduk's system using same data. We put time usage as metric of measurement to conduct efficiency test. The efficiency means, how fast was our system be able to reveal all basis paths all over our data test. The efficiency measurement was based on how much time needed to generate independent paths using our method against the Ghiduk's methods. The following equations are how we get the measurement of efficiency.

$$Efficiency = \frac{Ghiduk's\ system\ time\ usage - Our\ system\ time\ usage}{Ghiduk's\ system\ time\ usage}\ X\ 100\ \% \quad (2)$$

Ghiduk's system usage time variable was how much time needed by Ghiduk's method to seek basis paths within all code from data test. Our system time usage variable was how much time needed by our system to seek basis paths within all code from data test.



/Users/ArwanMB/NetBeansProjects/Pembangkit_Kasus_Uji/src/Pong.java

initgame1

population size : 20 max generation : 20

```
============================================
Class:Pong, Method :initgame1, V(G) = E - N + 2 = 9 - 8 + 2 = 3
jumlah if = 0,jumlah while = 0,jumlah For = 2,jumlah case = 0
degree if:
--------------------------------------------------------------
degree loop:
-------------------------------------------------------------
degree 1: 2 pasang
============================================
0(Entry)->1(score = 0)->2(for (int j = 0; j < (ball.length) ; j++))->4(ball[j] = new Ball(((int)((ja
test case:for (int j = 0; j < (ball.length) ; j++): F, for (int i = 0; i < (brick.length) ; i++): F,
0(Entry)->1(score = 0)->2(for (int j = 0; j < (ball.length) ; j++))->4(ball[j] = new Ball(((int)((ja
test case:for (int j = 0; j < (ball.length) ; j++): F, for (int i = 0; i < (brick.length) ; i++): T,
0(Entry)->1(score = 0)->2(for (int j = 0; j < (ball.length) ; j++))->4(ball[j] = new Ball(((int)((ja
test case:for (int j = 0; j < (ball.length) ; j++): T, for (int i = 0; i < (brick.length) ; i++): F,
count result:3--------------------------
```

Figure 6. User Interface of our system

3. RESULTS AND ANALYSIS

We have two measurements in measuring how successful of our method. These methods were accuracy and efficiency. The following are those methods.

3.1. Accuracy Result

To measured the accuracy of our system, we have searched the basis paths from the data test manually. Next process was generating basis paths which done automatically by our system. Both result then compared, to proof how well our system was able to reveal basis paths on every method on data test sets. Table 3 The result of accuracy measurement shown in the following table.

Table 3. Accuracy Test Result

No	Name	Basis path by System	Manual Basis Path	All Path Appear
1	Dauber.mouseClicked	5	5	YES
2	Dauber.paint	4	4	YES
3	Edge.checkEdgeSucesor	3	3	YES
4	operasiGenetik.Cekkeberadaan	3	3	YES
5	operasiGenetik.copyChromosome	2	2	YES
6	operasiGenetik.selectBest	3	3	YES
7	DDG.printNodeList	2	2	YES
8	Pong.run	6	6	YES
9	Pong.initgame1	3	3	YES
10	Pong.paint	5	5	YES
11	Pong.PaintGame2	4	4	YES
12	Chromosom.printGene	2	2	YES
13	Chromosom.setZero	2	2	YES
14	Chromosom.isEqual	3	3	YES
15	Chromosom.generateRandom	2	2	YES
16	PorterStemmer.Step5	6	9	NO
	Total Paths	49	58	

Accuracy of our system has shown on the following equation.

$$Accuracy = \frac{49}{58} \;\; X\ 100\ \% = 0.845\ \%$$

3.2. Efficiency Result

To measure an efficiency of our system, we only chose the code that all the basis path have revealed by our system. It means the data was only 15 from 16. Table 4 shows the efficiency result of our system. We have displayed Label attribute on Table 4 as the representation of the number of iteration which has predicted by J48 tree classifier. By using this label, our system was able to determine how many iterations needed to reveal basis path on a single method. As explained in Table 4, we were able to increase efficiency on time usage as follows.

$$Efficiency = \frac{9570 - 6087}{9570} \quad X\, 100\,\% = 35\,\%$$

Table 4. Time Usage to Reveal Basis Path

No	Name	Label	Our Time System	Time Ghiduk's Method	All Path Appear
1	Dauber.mouseClicked	L	126	126	YES
2	Dauber.paint	L	9	9	YES
3	Edge.checkEdgeSucesor	L	39	39	YES
4	operasiGenetik.Cekkeberadaan	L	17	17	YES
5	operasiGenetik.copyChromosome	L	20	20	YES
6	operasiGenetik.selectBest	L	18	18	YES
7	DDG.printNodeList	L	9	9	YES
8	Pong.run	H	5743	9100	YES
9	Pong.initgame1	L	102	102	YES
10	Pong.paint	L	57	57	YES
11	Pong.PaintGame2	L	30	30	YES
12	Chromosom.printGene	L	11	11	YES
13	Chromosom.setZero	L	4	4	YES
14	Chromosom.isEqual	L	15	15	YES
15	Chromosom.generateRandom	L	13	13	YES
	Total Time Usage (s)		6087	9570	

4. RESULTS AND DISCUSSIONS

Our method has reached accuracy 84.5%, it can reveal 49 basis path of 58 actual basis path. These results were in line with Ghiduk's methodology. Our results were better 4.5 % than Ghiduk's did which was only 80%. These results are likely to be influenced by a lack of uniformity of test data since it only one data which has label H(need high iteration).

Time efficiency results reach 35% faster than Ghiduk's method. An efficiency of our method is obtained from data number eight with labeled H in Table 4. The time needed by Ghiduk's method on 8th data takes much more time because we need three times trials to reveal basis paths (we try with L (low, 20) iterations, M(medium, 50) iterations, and finally H(High, 100) iterations). While our system only requires once trial with High(100) iterations to reveal basis path, because it already predicted by J48 algorithm previously. This results can be increased by using more data test with label M or H since the Ghiduk's need several trials and our method only once trial.

5. CONCLUSION

We have discovered that combination of Genetic Algorithm (illustrated by Ghiduk's) and J48 has the ability to reach accuracy 84.5 %. It better 4.5% from what Ghiduk's did (80%). This combination was also able to reach efficiency 35 % faster than what Ghiduk's done before. Heterogeneity of complexity of data also can lead into increasing of efficiency, because the system is able to predict the iterations. Our future work, we will investigate how to generate basis path using other methodology such a machine learning or network computation. We also will try to extend on how to produce test case based on the basis path testing.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the member of Software Engineering Research Group in Faculty of Computer Science, University of Brawijaya. This work was supported in part by the State Budget-Operational Assistance Universities Scheme (Grant no.: 4084.3/UN10.36/KP/2016)

REFERENCES

[1] N. T. Binh, T. C. Duy, and I. Parissis, "LusRegTes: A Regression Testing Tool for Lustre Programs," *Int. J. Electr. Comput. Eng.*, vol. 7, no. 5, p. 2635, 2017.

[2] I. Sommerville, *Software Engineering*, 9th ed. Pearson, 2011.

[3] V. Elodie, "White Box Coverage and Control Flow Graphs," pp. 1–33, 2011.

[4] A. Bertolino, R. Mirandola, and E. Peciola, "A case study in branch testing automation," *J. Syst. Softw.*, vol. 38, no. 1, pp. 47–59, 1997.

[5] S. K. Mohapatra and S. Prasad, "Test case reduction using ant colony optimization for object oriented program," *Int. J. Electr. Comput. Eng.*, vol. 5, no. 6, pp. 1424–1432, 2015.

[6] A. S. Ghiduk, "Automatic generation of basis test paths using variable length genetic algorithm," *Information Processing Letters*, vol. 114, no. 6. pp. 304–316, 2014.

[7] A. Ghiduk, M. R. Girgis, and A. S. Ghiduk, "Automatic Generation of Data Flow Test Paths using a Genetic Algorithm Automatic Generation of Data Flow Test Paths using a Genetic Algorithm," no. February, 2014.

[8] W. Xibo and S. Na, "Automatic Test Data Generation for Path Testing Using Genetic Algorithms," 2011.

[9] Y. Wang, H. Wu, and Z. Sheng, "A Prioritized Test Generation Method for Pair-wise Testing," vol. 11, no. 1, pp. 136–143, 2013.

[10] Y. Li, J. Du, Q. Hu, and X. Liu, "A Method for Structure-Oriented Regression Test Path Generation," pp. 30–36, 2016.

[11] G. Kaur and A. Chhabra, "Improved J48 Classification Algorithm for the Prediction of Diabetes," *Int. J. Comput. Appl.*, vol. 98, no. 22, pp. 13–17, 2014.

[12] S. Herbold, J. Grabowski, and S. Waack, "Calculation and optimization of thresholds for sets of software metrics," *Empir. Softw. Eng.*, vol. 16, no. 6, pp. 812–841, 2011.

[13] a. H. Watson, T. J. McCabe, and D. R. Wallace, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric," *NIST Spec. Publ.*, vol. 500, no. 235, pp. 1–114, 1996.

[14] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, p. 10, 2009.

[15] R. Pawlak *et al.*, "Spoon : A Library for Implementing Analyses and Transformations of Java Source Code To cite this version : Spoon : A Library for Implementing Analyses and Transformations of Java Source Code," 2015.

BIOGRAPHIES OF AUTHORS

Achmad Arwan has received his Bachelor of Computer Science from STIKI Malang in 2006. He also has got Master of Computer from 10 November Institute of Technology Surabaya in 2015. He is now a member of Software Engineering Research Group (SERG) in Faculty of Computer Science, Brawijaya University. His current research interest is Software Engineering, Software Testing, Software Maintenance, Software Evolution.

Denny Sagita Rusdianto has received his Bachelor of Computer Science from Universitas Brawijaya Malang. He also has got Master of Computer from 10 November Institute of Technology Surabaya. He is now a member of Software Engineering Research Group (SERG) in Faculty of Computer Science, Brawijaya University. His current research interest is Software Engineering, Software Design, Software Testing, Software Evolution.