View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by TELKOMNIKA (Telecommunication Computing Electronics and Control)

TELKOMNIKA, Vol.16, No.3, June 2018, pp. 1264~1275
ISSN: 1693-6930, accredited **A** by DIKTI, Decree No: 58/DIKTI/Kep/2013
DOI: 10.12928/TELKOMNIKA.v16i3.4442

Behavioral Analysis for Detecting Code Clones

Bayu Priyambadha*[1], Siti Rochimah[2]
[1]Informatics Department, Faculty of Computer Science
University of Brawijaya, Indonesia
[2]Informatics Department, Faculty of Information Technology
Institut Teknologi Sepuluh Nopember, Indonesia
*Corresponding author, e-mail: bayu_priyambadha@ub.ac.id

Abstract

The activities of copy and paste fragments of code from a source code into the other source code is often done by software developers because it's easier than generate code manually. This behavior leads to the increase of effort to maintain the code. One of the detection methods of semantic cloning is based on the behavior of the code. The code behavior detected by observing at an input, output and the effects of the method. Methods with the same value of input, output, and effect will indicate that semantically the same. However, the detection method based on the input, output, and effect could not be used in a void method or method without parameters, another side comprehensively detection is required. The challenge is how to detect which variable in a method that acts as input, output, and effect. Detection of the variable input, output, and effects in a void method done using Program Dependence Graph. The use of clone detection methods semantically based on behavior can increase the agreement value.

Keywords*: clone detection; semantic clone; behavioral cloning; software maintenance*

Copyright © 2018 Universitas Ahmad Dahlan. All rights reserved.

1. Introduction

The process of copying and insertion of code fragments specific code section to another section of code referred to as cloning. The process can be accompanied or not by changes or the modification of code [1]. Software developers often do so for various reasons, such as software limitations of working time. Some researchers claim that there is a 20-30% overall code are cloned in the source code [1]. The existence of this cloning requires good management, given that there are some impacts that may be caused.

Code cloning is believed can cut the time of generating the source code in software development. With the cloning, a developer has a very simple way to generate the code. However, it will be able to cause problems when one of the copied code fragments containing the bug. This condition will increase the cost of maintaining the source code. Increasing maintenance costs are shown by an increase in efforts [2-3], the time and expense [4-5]. There is also a correlation between the presence of clones in code with the onset of defects in software. Sjoberg and Yamashita [2], states that if the maintenance is not carried out consistently, it can lead to defects in the software. The same thing also delivered by Bettenburg et al., [6] which states that disability may occur if the developer did not perform well on cloning maintenance code.

Several methods have been used in the process of clone discovery. These methods are grouped into two groups, a method that is executed based on the similarity of syntax and meaning (semantics). In the case of code, the semantic similarity of code cannot be looked from the relatedness between the words [7] or the ontology [8]. Elva and Leavens [9-10], mention in different terms, the representational and functional similarities. The similarity of the syntax is done by looking at how the source code it is presented or written. This approach considers only the similarity of syntax and has minimal semantic information. Meanwhile, the semantic similarity, the search process is done by looking at the meaning of the code. The meaning of a code fragment can be seen from the function of the code [5],[9], and the relationships of the elements of the code [10-11]. Because of the presence of clone can lead to the increase of the maintenance cost, then the cloned code in a source code of the software must be found to help facilitate the management process of cloning. The importance of knowing the existence of clone is to simplify the process of refactoring the code. Refactoring code is done to maintain the

Received August 20, 2017; Revised April 3, 2018; Accepted April 26, 2018

consistency of the code and to reduce the possibility of the appearance of the bug. Clones code that is not consistent can increase maintenance costs and increase the likelihood of bugs.

In the semantic similarity detection process, the additional information is needed to enhance the meaning of the fragment of the source code. In the semantic similarity detection process, the additional information is needed to enhance the meaning of the fragment of the source code. The semantic similarity detection process has been studied by several researchers. Gabel et al. are using the Program Dependence Graphs (PDG) to gain the semantic information from code [12]. PDG is used to represent control data and dependencies between statements in code. Gabel et al. are assuming that the isomorphism model that represented by PDG from a couple of code fragment indicate that they are the clone [12].

Keivanloo and Rilling propose an approach that the meaning of code can be gained from the relationship between token. They assume that if we consider the text, will be difficult to measure based on the meaning of the code (semantic). Keivanloo and Rilling did an enhancement of the method using semantic web technology. They connect the code based on inheritance relation between object. The relationship between tokens is known from the inheritance relation from the token [13]. Jiang and Su using difference approach for semantic clone detection. In their research, the semantic similarity in code is a functional similarity. If two fragments of code produce the same output when given the same input value, then those fragments are semantically similar [5]. There is an automatic mechanism for generating input value in this research.

Elva and Leavens have the same vision as Jiang and Su. They are focused their research on method or function at Java language. They also believe that the function of the method can be identified by the value of input and output. In this research, Elva and Leavens give an additional parameter to consider in the process of functional similarity detection. The parameter is the effect. Effect of the method in Java is assumed can give additional information for strengthening the meaning of method. Then, they named the detection process as IOE-Behavior (Input, Output, Effect-Behavior) [9],[14],[10]. Inputs are all parameter on the method. The output is the result of the method. And, effects are all attributes of the classes that have possibility affected by the method at the runtime.

Input and output are needed in the testing phase. The method that does not have the input parameters and return value will be assigned as a cloned as long as they have the same definition of input, output, and effect [10]. All of the methods that do not have the definition of input, output, and effect have possibility contain the clone. Therefore, the deeper study is indeed to solve that problem.

To resolve that problem, in this research will produce the framework for semantic clone detection. The framework has the function to detect clone code based on the behavior of the method. The challenges of this research are, how we can identify all variable that acts as input, output, and effect to detect semantic clone. All void and parameterized method are identified so that the analysis of method can be done to all method without exception. The preliminary research has been done and the result shown that this method is very promising [15]. In the preliminary research, the PDG analysis is done to all of Java methods. In this research, the PDG analysis is applied more selective, not all methods are processed but only on void and without parameters methods. And, there is additional source code that is analyzed.

2. Research Method

In this part describe the research methodology that is used in this research. The phases consist of (1) Identification and collection Java methods, (2) Grouping the methods based on the input, output and effect definition, (3) Reconstruction method, (4) Testing/execution method, (5) Grouping methods based on the result of testing/execution, (6) Analysis of result. The contribution of this research is in the first phase. At the first process identify and collect all methods from the source code, then identify the definition of input, output, and effect by using PDG. All the phases are described in Figure 1.

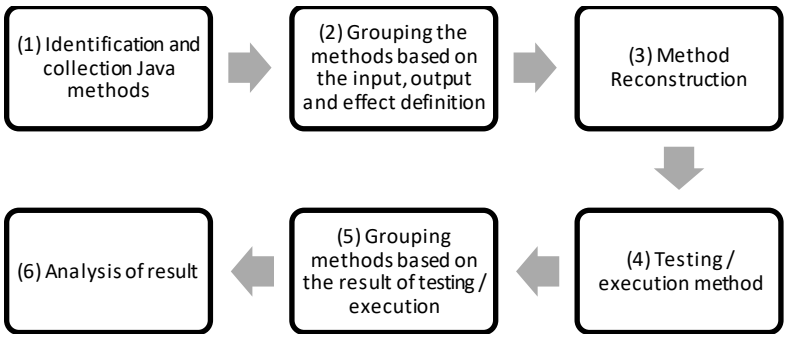

Figure 1. Our Approach

2.1. Identification and collection java methods

The first technical phase of this research is collecting all methods in Java class. In the process of collection methods, there is a way to find all methods in object-oriented source code. This process is applied to all source code files by using a library that works based on byte-code analysis approach. The library is System Dependencies Graph API (SDGAPI). The result of method analysis is saved in one object. Finally, all object of methods are saved in list data type. The collection of methods is described in Figure 2.

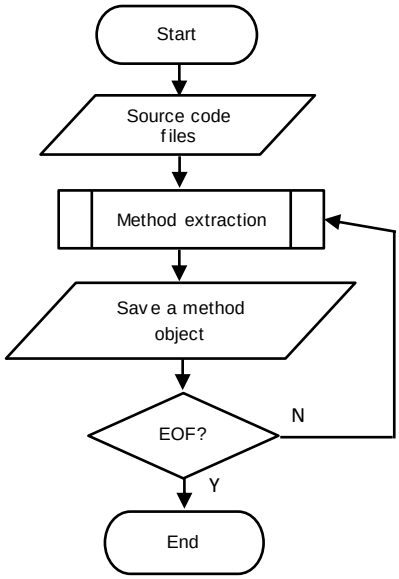

Figure 2. The Process of Collection Methods

There is method extraction process in Figure 2. This process is how to gain all information on the methods. An extraction of void or non-parameterized method is done using Program Dependence Graph (PDG) from SDGAPI. PDG describe the dependencies between the variable in the method body. The analysis of input, output, and effect is done using the dependencies relation between variable. Input and output variables are all nodes that start and end the PDG.

The process of collection and identification of Java methods is done by an application that developed specifically for this research. The application is developed with Java language and using the SGDAPI. The application has two kinds of input type files. The files are .java files and .class file. Because of SGDAPI is working with bytecode analysis approach, that why the

application needs a .class file for the input. The architecture of the application will descript in Figure 3.

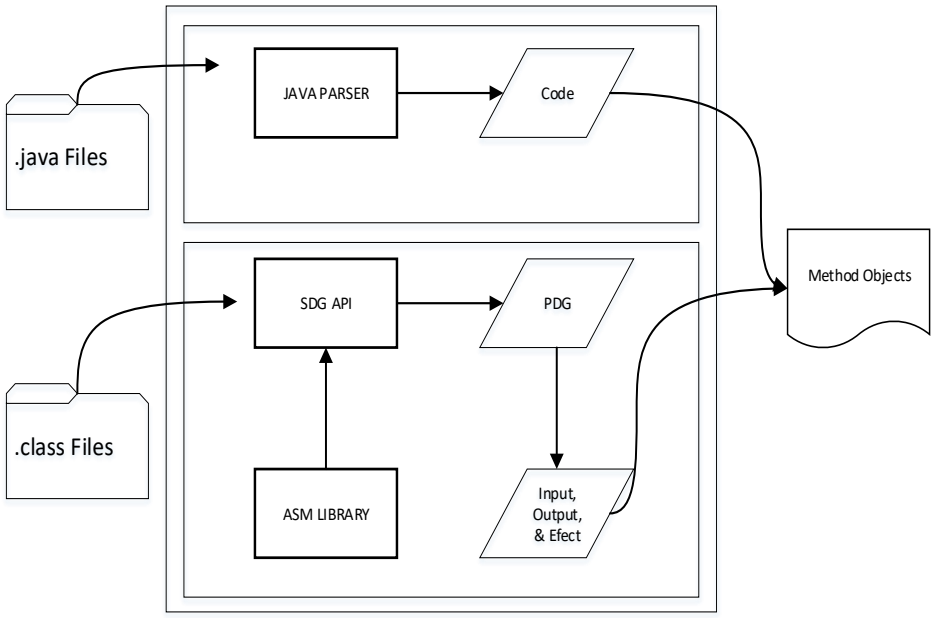

Figure 3. The Architecture of Application

2.1.1. Input identification

Identification of input from the Java methods is made using PDG. The information that will be gained from this process is the type of input variable. The input variables determine the type of random value that uses in testing or execution phase. Definition 1. Input is all of the types of input parameters of one method. In the case of a method with parameters, all type of parameters will be an input definition of the method.

Figure 4 describes the example of a method with parameters. From that example code, by using definition 1, the input definition of method inc are int and int. The condition will be different if the method does not have parameters. By using PDG, input in a method that does not parameters have will be discovered. Figure 4 shows the example of the method without parameters.

```
1  int inc (int x, int y)
2  {
3         int result;
4         result = x + y;
5         return result;
6  }
```

Figure 4. The Example of Method with Parameters

Definition 2. Input is all node that affects another node, not be affected by another node. In this case, all of the nodes in PDG that become the starter of the graph are input node. The PDG of code on Figure 5 is described in Figure 6. There are three nodes that become a starter in this graph, “int x”, “int y” and “String result”. But, not all of that starter node becomes an input. According to definition 2, all nodes that affect and not affected by another node become an input. “String result” are excluded from input because it will be affected by another node. So, the input definition of that method is `int` and `int`.

```
1   String print()
2   {
3           int x;
4           int y;
5           x = 2;
6           y = 3;
7           String result;
8           result = "Resutl : " + x * y;
9           return result;
10  }
11
```

Figure 5. The Example of Method Without Parameters

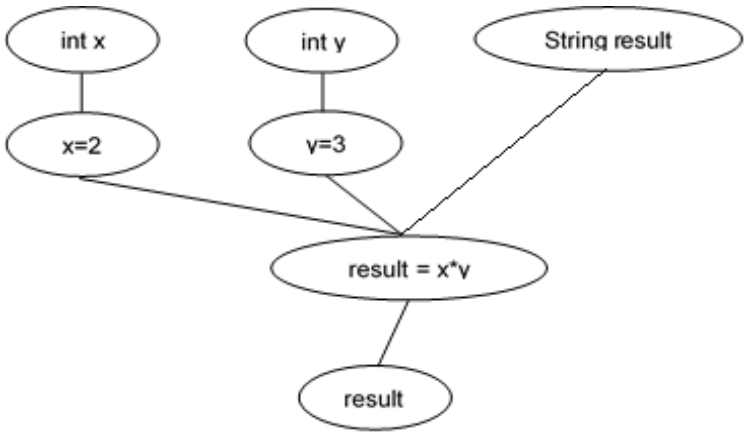

Figure 6. The PDG of Print Method

2.1.2. Output identification

As well as input identification, output definitions are obtained using the same process. PDG is also considered in this process. But, output has a different definition of input. Definition 3. The output is the type of return value of the methods. All of the type of return value of the method, if any, will be the type definition of output for a method. For all methods that have the return value (not void method), it can directly be assigned to output definition.

Definition 4. The output is the node that becomes an estuary for all node or the node that only affected by another node. All of the nodes that have a position at the end of the graph will be assigned to output. For example Figure 7, that is a void method. There is no return value of the method. The method only does some process and not return the result value of the process. To get which variables act as the output, the use of PDG is very useful. According to definition 3 and 4, and the information from PDG, then output variable can be found. The dependence graph of the example in Figure 7 are described in Figure 8.

```
1   void print(int x, int y)
2   {
3           int result;
4           result = x * y;
5           System.out.println("The      result      of
6   multiplication  "+x+" and "+y+" is "+ result);
7   }
```

Figure 7. The Example of Void Method

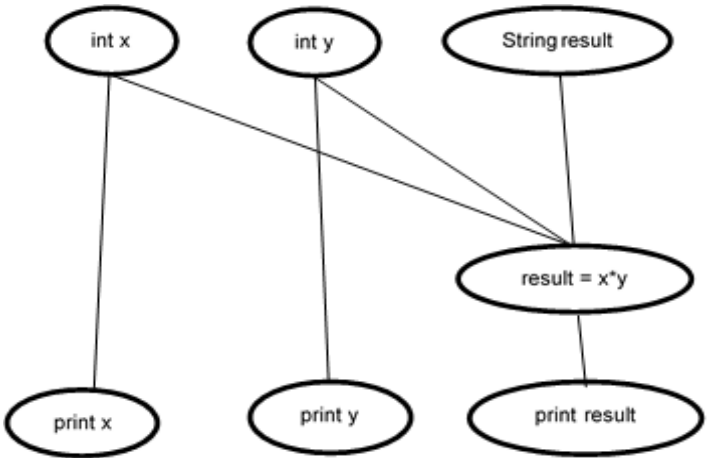

Figure 8. Dependence Graph of Code from Figure 6

According to definition 3 and 4, if the method has the return value, then it can be directly decided that the output is the return type of the method. If the method is a void method, then definition four will be used to find the output. For the example of dependency graph in figure 8, there are three nodes that can be output according to definition 4. The nodes are "print x", "print y" and "print result". From the nodes, there are variable `x`, `y`, and the `result`, and each of them has a type of int. From the example, the definition of output is `int`, `int`, and `int`.

2.1.3. Effect identification

The effect has a different definition of output. The effect is something that happens during the process of generating or resulting output. In the case of Java programming, it is identified with the entity or attribute that lies in the body of the method.

Definition 5. The effect is all attributes from the class that has the possibility to be manipulated during the process resulting in the output.

All of the attributes of a class that has the possibility to be manipulated are listed and become the effect of the method. For example, there is class A and has an attribute A1. There is class B that has method MB1. In the method MB1, A1 of A is manipulated and has the possibility the value of A1 changed. From that description, method MB1 affect A1.A.

2.2. Grouping the methods based on input, output, and effect definition

Grouping the Methods is the first step of grouping methods. This process produces several groups of a method that has a possibility having a similar function. Grouping methods are to eliminate methods that have no possibility having a similar function. All methods that have no similar definition of input, output, and effect will be erased from a collection of candidate methods.

Definition 6. (Similarity of Methods), A couple of methods are stated as similar whether it has a similar definition of input, output, and effect. Input, output, and effect defined as string data. The matching process is done using exact match comparison between string. The process is performed on every element data input, output, and effect. On the other hand, this process is done to keep and find the group of methods that contextually the same. All of the methods must be given the same value of the input to keep the contextual of the method in one group [9].

2.3. Method reconstruction

The method reconstruction process is for preparing the method that will be run at testing phase. Reconstruction is done by altering the parameters of the method and appending some script inner the body method manually. This process is done at all of the methods that do not have input parameter or return value as an output of the method. The result of input, output and effect identification is used as base information for doing reconstruction.

All methods that do not have input parameter and output or one of them, and identified as a clone candidate, will be reconstructed. The method that there is no input parameter in the

code definition, and identified own an input; then the code is altered by adding a parameter as the result of PDG analysis. The method that does not have a return value, and identified own output by PDG analysis, the body of code is appended with the code that has a function to listen to the value of the variable that becomes output node. The output modification cannot do by adding "return" keyword in java code because the output of method resulted by PDG analysis is possibly more than one node. All methods are prepared in this phase to all of them can be run or simulated using random data. Several things that need to consider in the method reconstruction is defined as follow.

a. The name of parameters of the method is adapted from the name of the variable that becomes an input node of the method.
b. The code that defines the variable that becomes an input is deleted because this variable will put to the method parameter.
c. The code that gives an initial value to a variable that becomes an input is deleted because the initial value is got from the parameter and it can cause the death code in the body.
d. Input node that calls a method from another method is replaced with a variable that has the same type with the return value of the method.

2.4. Testing or execution method

All methods that have similarity the definition of input, output, and effect are tested using the same data for one group. The data are generated using a random function that can produce the value that fits with the type of parameters. The random function produces a value of primitive type. The parameter that has a type as user define reference type are defined manually.

The initial code for test the method is generated automatically at the same time with grouping the methods. Then, in the final process, the SQL script is produced. The scripts contain many information, example, the name of the method, the name of the class where the method takes place, the name of the cluster, input value, output value, and effect. With SQL script, all data can be saved to database easily. The example of initial code for executing an example of the method is descript in Figure 9.

```
testing_sql.reset();
testing_sql.method = "getTextHeight ";
testing_sql.class_nm = "TextCalculator";
testing_sql.cluster = "Cluster 10 (Clone)";
testing_sql.app = "jDraw";
TextCalculator TextC = new TextCalculator ();
TextC.getTextHeight();
System.out.println(testing_sql.sql());
```

Figure 9. Example of Initial Code

From the code in figure 9, the tested method is getTextHeight, from class TextCalculator, cluster 10, and source code of jDraw. getTextHeight is the method that already reconstructed and ready to invoke. From the example, all the needs are the code that defines to generate random data to pass to the method. The final code that already appends the additional code is descript in Figure 10. The code that blocked with yellow is additional code. Random Data is the class that has a responsibility to generate random data. There are several methods to generate data that divided based on the primitive type of data. At the example, random for an integer type of data. Then, if all random data required in this testing method is ready, the data passed to a method to proceed.

```
int a = new RandomData().RandomInteger(100);
int b = new RandomData().RandomInteger(100);
testing_sql.reset();
testing_sql.input = Integer.toString(a)+"|"+Integer.toString(b);
testing_sql.method = "getTextHeight ";
testing_sql.class_nm = "TextCalculator";
testing_sql.cluster = "Cluster 10 (Clone)";
testing_sql.app = "jDraw";
TextCalculator TextC = new TextCalculator ();
TextC.getTextHeight(a,b);
System.out.println(testing_sql.sql());
```

Figure 10. The Final Code in Testing Phase

2.5. Grouping methods based on the result of testing

Before grouping the methods, the SQL script that resulted from the previous phase is executed. All log of the testing process is stored in the database. This phase is the determination phase. In this phase, we decide which method that has the same behavior. The grouping method is the way to know which method has the same behavior. The behavior similarity detection is based on the definition as follow [10]:

$$\forall A\ \forall B |\ A,B \in Methods$$
$$IOE - Behavior(A) = IOE - Behavior(B) \leftrightarrow SemanticEquivalent(A,B) \quad (1)$$

All methods that have the same value of input and output, and the same definition of effect, then those methods are semantically or behaviorally equivalent. Not all of the methods that are grouped in previous grouping process are semantically similar. The only method that has the same input, output, and effect definition and also has the same value of input and output is decided as a semantic clone.

2.6. Analysis of result

Analysis of result is conducted with the aim to evaluate the result of the implementation of the purposed method. The evaluation is done by comparing the result of the system (purposed method) and the result of manual analysis by an expert. Expertise is a very familiar and experienced in designing a software source code. Then, from the result, the value of agreement between the system and an expert are counted using Cohen's Kappa coefficient [16].

$$K = \frac{P_o - P_c}{1 - P_c} \text{ where,}$$

$$P_o = \frac{a+d}{n}, \text{ and} \quad (2)$$

$$P_c = \frac{\left(\frac{f_1 x g_1}{n}\right) + \left(\frac{f_2 x g_2}{n}\right)}{n}$$

Where P_o is the proportion of the similarity of observation and P_c is the proportion expected by chance [16]. Then, the data obtained from observations of two observers described in tables relevance shown in Table 1. Kappa value can determine the level of agreement between the expert and system. The value can be interpreted using interpretation table as shown in Table 2. The Table 2 explains that there are some categories for the result of Kappa coefficient. The value of Kappa can be classified based on that categories.

Table 1. Tables Relevance [16]

Observer		II		Total
		Relevant	Not Relevant	
I	Relevant	a	b	g_1
	Not Relevant	c	d	g_2
	Total	f_1	f_2	n

Table 2. Interpretation Table [17]

Kappa	Level Agreement
< 0	less than chance agreement
0.01 – 0.20	slight agreement
0.21 – 0.40	fair agreement
0.41 – 0.60	moderate agreement
0.61 – 0.80	substantial agreement
0.81 – 1	almost perfect agreement

3. Dataset and Testing Scenario

This section descript dataset and testing scenario to perform the purposed method. The dataset that used in this research is the open-source software written in Java.

3.1. Dataset

The dataset consists of some source code written in the Java language. The source code is available on several sites code repository. From some source code, the code is grouped into code sizes small, medium and large based on the number of methods contained in each source code. The code with the small size is a code that has a method of fewer than 1000 methods. The code with a medium-size contains more methods from 1000 to 1500. And, with the large size of the code is code that has methods over 1500. Provisions grouping source code based on the number of such methods is still an open issue. There are no references say for certain groupings of code size based on the number of methods. The source code used in this research is the source code that meets size classifications small, medium and large. Applications used to test in this study are described in Table 3.

Table 3. List Dataset

No.	Application	Size	Number of Methods	Repository
1.	DNSJava 2.1.6	Small	806	http://www.dnsjava.org/
2.	jDraw 1.1.5	Medium	1243	http://jdraw.sourceforge.net/
3.	jEdit 5.2	Large	4004	http://www.jedit.org/

3.2. Testing scenario

Testing of methods offered in this study conducted by implementing a workflow that has been proposed and described in the methodology. The workflow is implemented in the case of data that is appointed for the trial. The data used for the test is divided into three categories: small, medium and large. Testing is done by comparing the results with the results of the identification system experts. Experts elected for identification is a person who is familiar with the source code and understands the types of clones exist in the source code. There are three experts, each expert identification at the source code an application system.

The identification process by the system is done in two ways, namely automatically and manually. The result of identification by the system is recorded in a database. Then, the log data will be analyzed by comparing the results of the identification by experts. In the final phase, Kappa coefficient was calculated to see the degree of match between the identification results with expert systems.

4. Results and Analysis

Tests performed on the source code of the three application system that is DNSJava, jDraw, and jEdit. Each has a varied number of methods. Total methods from three sources code are 6053 methods. 6053 method will get the same treatment from the beginning of the process until the end.

4.1. Analysis of agreement between system and expert

The results of detection which is recorded from the detection process using the proposed framework in this study are called as a result of the system detection. The results of the system detection will be compared with the results of the detection of an expert. The agreement between the expert and system and is calculated using the kappa coefficient. Table 4 shows the results of the calculation of kappa on all source codes.

Table 4. The Result of Kappa of the Proposed Method

DNSJava				jDraw				jEdit			
System	**Expert**			**System**	**Expert**			**System**	**Expert**		
Clone	299	179	478	Clone	486	191	677	Clone	886	675	1561
Not	5	69	74	Not	16	62	78	Not	375	445	820
Total	304	248	552	Total	502	253	755	Total	1261	1120	2381
Po	0.666667			Po	0.725828			Po	0.559009		
Pc	0.537125			Pc	0.630828			Pc	0.509215		
Kappa	0.279864			**Kappa**	0.257331			**Kappa**	0.101458		

On average Kappa coefficient obtained from the calculation in each application is 0.2128. Of the average yield, it can be interpreted that the result of the agreement between the system and the experts in this study was quite (fair agreement).

4.2. Comparison with previous research

The comparison process is done by comparing the results of the analysis proposed method with the method in previous studies. The calculation method is done by eliminating the PDG analysis process on the proposed method. Results kappa coefficient calculation on the previous method (without analysis PDG) is described in Table 5.

Table 5. The Result of Kappa of the Previous Method

DNSJava				jDraw				jEdit			
System	**Expert**			**System**	**Expert**			**System**	**Expert**		
Clone	304	299	603	Clone	208	828	1036	Clone	1229	1749	2978
Not	0	87	87	Not	0	66	66	Not	0	459	459
Total	304	386	690	Total	208	894	1102	Total	1229	2208	3437
Po	0.566667			Po	0.248639			Po	0.491126		
Pc	0.455564			Pc	0.22603			Pc	0.395619		
Kappa	0.204069			**Kappa**	0.029211			**Kappa**	0.158025		

The clone detection by eliminating the PDG analysis as was done in previous studies resulted in the greater number of methods of clones compared with the result of the proposed method in this study. Table 6 describes the comparison of clones from the previous method (without PDG) and method performed in this study. The table shows the number of significantly different between the two methods. Previous method (without PDG) looked very good at detecting clones.

Table 6. The Result Comparison (Previous and Proposed Method)

	DNSJava		jDraw		jEdit	
	Previous	Proposed	Previous	Proposed	Previous	Proposed
Clone	603	478	1036	677	2978	1561
Not	87	74	66	78	459	820

4.3. Discussion

The analysis results in the comparison of the number of methods that claimed as clone method. There are the differences between the number of previous and proposed method. The previous method looked more accurate than proposed method. From the distribution of type of clones in Figure 13, the previous method, there is a considerable value obtained for this type of Not Clone (not clones).

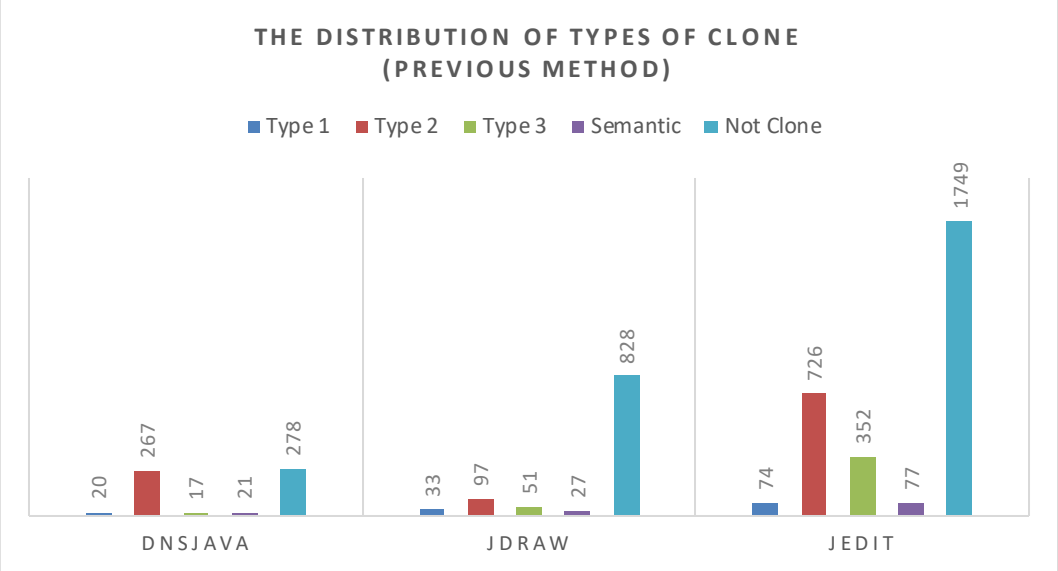

Figure 13. The Distribution of Types of Clones

The number of not clones of each source code (DNSJava, jDraw, jEdit) showed the largest number compare with another type that clones. The deeper analysis is needed to find out are all the methods that claimed as clones are really clones. This analysis is done by looking at every method that claimed as a clone or not clones. These results imply that, in the previous method (without PDG), there are many false positive values. Where false positive is the number of a method that identifies as clones by the system but in reality is not clones.

The number of the false positive method can cause a decline in the value of the accuracy or the degree of fit between the methods with experts. Comparison of the average value of Kappa coefficient between the previous method and the proposed method is 0.13043 and 0.2128. 0.08237 there is a difference value between the two methods. It is mean that the proposed method is more conformable with the expert than the previous method with the value of Kappa coefficient 0.08237 greater than the previous method. The proposed method analyzes every method even there is no parameter or return value at the method which was not done in the previous method.

5. Conclusion

Type of input, output, and effect of a method can be specified on all methods of both the void method or non-void method. The specification of input, output and the effect of the method is done by analyzing the PDG formed based bytecode of a method. Generation PDG using a library that runs using the method bytecode analysis.

The results of the experiment by the system are compared to the test by experts. This approach aims to get a Kappa coefficient of the proposed method in this study. The results of calculation of Kappa coefficient is 0.2128 which can then be interpreted that the results of the degree of fit between the system and the experts in this study were a fair agreement. The result of the system is also compared to the previous method. Comparison of the average value of Kappa coefficient between the previous method and the proposed method are 0.13043 and

0.2128. The difference values between the two methods are 0.08237. The proposed method is more conformable with the expert.

Some of the processes of clone detection on the source code are done manually. For the next study, we try to attempt all process can be done automatically. The recapitulation of the testing in this study shows there are still methods that cannot be identified. The assumption is this happens because of the failure of library SDGAPI in generating PDG. Therefore, some methods which do not have a definition cannot be analyzed. In upcoming research, this problem has to be optimized. Generating PDG will be attempted from source code.

References

[1] D. Rattan, R. Bhatia, and M. Singh, *Software clone detection: A systematic review*. Elsevier B.V. 2013; 55(7).
[2] D. Sjoberg and A. Yamashita. Quantifying the effect of code smells on maintenance effort. *IEEE Trans. Softw. Eng.* 2013; 39(8): 1144–1156.
[3] A. Lozano and M. Wermelinger. *Assessing the effect of clones on changeability*. 2008 IEEE Int. Conf. Softw. Maint. 2008; 227–236.
[4] M. Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.
[5] L. Jiang and Z. Su. *Automatic mining of functionally equivalent code fragments via random testing*. Proc. eighteenth Int. Symp. Softw. Test. Anal. - ISSTA '09. 2009: 81.
[6] N. Bettenburg, W. S. W. Shang, W. Ibrahim, B. Adams, Y. Z. Y. Zou, and a. E. Hassan. *An Empirical Study on Inconsistent Changes to Code Clones at Release Level*. 2009 16th Work. Conf. Reverse Eng. 2009; 77(6): 760–776.
[7] K. B. Fard, M. Nilashi, M. Rahmani, and O. Ibrahim. Recommender System Based on Semantic Similarity. *International Journal of Electrical and Computer Engineering (IJECE)*. 2013; 3(6):751–761.
[8] H. Xu and O. Musicant. Design and Implementation for Ontology Modeling of Design Knowledge Based on UML Class Diagram. *TELKOMNIKA (Telecommunication Computing Electronics and Control).* 2016; 14(3A): 326-332.
[9] R. Elva and G. Leavens. *Semantic clone detection using method IOE-behavior*. 2012 6th Int. Work. Softw. Clones. 2012; 80–81.
[10] R. Elva. Detecting Semantic Method Clones in Java Code Using Method IOE-Behavior. Orlando: University of Central Florida, 2013.
[11] L. Jiang, Z. Su, and E. Chiu. *Context-based detection of clone-related bugs*. Proc. 6th Jt. Meet. Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng. - ESEC-FSE '07. 2007; 55.
[12] M. Gabel, L. Jiang, and Z. Su. *Scalable detection of semantic clones*. Proc. 13th Int. Conf. Softw. Eng. - ICSE '08. 2008; 321.
[13] I. Keivanloo and J. Rilling. *Semantic-Enabled Clone Detection*. 2013 IEEE 37th Annu. Comput. Softw. Appl. Conf. 2013; 393–398.
[14] R. Elva and G. Leavens. Jsctracker: A semantic clone detection tool for Java code. Orlando, FL Univ. Cent. Florida. 2012.
[15] B. Priyambadha and S. Rochimah. Case Study on Semantic Clone Detection Based On Code Behavior. *Data Softw. Eng.* 2014; 1–6.
[16] J. Sim and C. C. Wright. The Kappa Statistic in Reliability Studies: Use, Interpretation, and Sample Size Requirements. *Phys. Ther.* 2005; 85(3): 257–68.
[17] J. R. Landis and G. G. Koch. The Measurement of Observer Agreement for Categorical Data. *Int. Biometric Soc.* 1977; 33(1): 159–174.