International Journal of Intelligent Computing and Information Sciences

AUTOMATIC DETECTING AND REMOVAL DUPLICATE CODES CLONES

Z. D. Al-Saffar S. S. Sarhan S. Elmougy

Department of Computer Science, Faculty of Computers and Information, Mansoura University, Mansoura 35516, Egypt.

zyad_thafer@yahoo.com shahenda_sarhan@yahoo.com mougy@mans.edu.eg

Abstract: *Code clones is considered now an important part of improving the overall design of software structure and software maintenance through making the source code more readable and more capable for maintenance. To remove code clones from a written code, refactoring technique could be used. Copying and pasting fragments of codes is a type of code clones that should be handled and has many practical applications such as software and project plagiarism detection clones and copyright infringements. To overcome this problem, we propose a computerized refactoring system to remove duplicate code clones. The simulation results of applying the proposed system showed that it increases the maintainability and quality of software system based on the total lines of code, blank lines and total methods count for the four used Java open source projects.*

Keyword: *Code Clones, Duplicated Code, Code Smells, Clone Refactoring, Clones Removing.*

1. Introduction:

Nowadays, programmers and software developers use code cloning when they are writing a code because 1) To obtain efficiency, a procedure call may be costly; 2) Programmers and programs developers mostly prefer copying parts of codes than writing them. They assumed that these copied parts have no errors because its original parts that copied from are previously tested, and 3) The code clones increases the difficulty of maintenance of programs because it increases the maintenance effort and cost, it is more unstable than new code. In addition, it increases the complexity and size of the source code. In reengineering software, the code clones detection is a main part toward the progression of refactoring code clones. The main activity of refactoring techniques is improving the source code without changing the external behaviors of the software system.

For removing code clones occurring because of using copy and paste, an automated refactoring of Java projects is proposed in this work. It is important to handle this type of clones to improve the quality of software and make it more readable and maintainable. Code clones detection is a useful tool in building some practical applications such as software and project plagiarism detection clones and copyright infringements.

In Section 2, basic definitions and background of code clones, refactoring, code clones types, techniques, and practical applications are discussed with presenting some previous work. In Section 3, the proposed system is discussed with illustrating it using a case study. Section 4 summarizes the experimentation and the final results. Conclusions are discussed in Section 5.

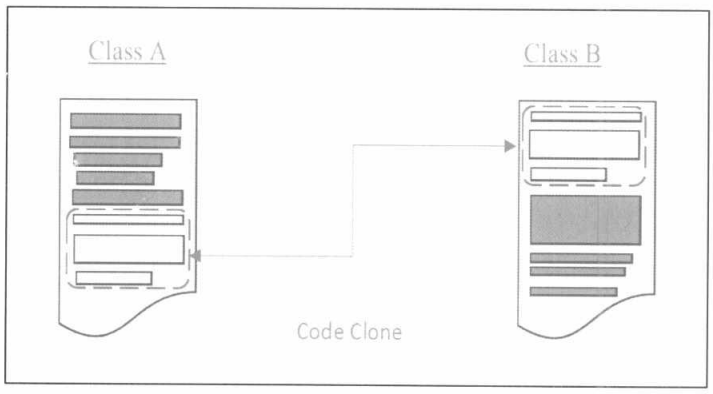


Figure 1: Code clone example

2. Background and Related Work:

Code clones is copying of the code fragment and then pasting it to another place of the source code without or with some changes and use it again. It is also called duplicated codes or just clones. The code clones in software development and maintenance can be considered code smells. Some recent studies showed that from 5.0% to 20.0% of source codes of systems can have code clones [1]. Figure (1) shows an example of type-I code clones in Class (B) as a result from copying and pasting a code from a class (A).

There are many reasons and benefits in reengineering software that push us to discover and to remove code clones from a code. We listed below some of these reasons and benefits [2]:

- **Decrease the bloat of source code:** Code clones refactoring decreases the size of original source code and reduces its executed time.
- **Avoid bugs:** Code clones may be cause bugs in the source code. When the clones from the source code are removed, the number of bugs is minimized and so it becomes easy to find such bugs.
- **Repair design-flaws:** One of the reasons for the problems of the design was code clones such as not using the inheritance in the source code.
- **Making the source code more understandable and readable:** Code clones detection may help the programmers and software developers to understand the software system.

Code clones detection is an important stage in improving the quality of source code. It could be applied in many applications such as:

- **Projects plagiarism:** One of the closely related fields to detect code clones is plagiarism detection clones [2].

- **Software plagiarism:** Code clones detection is also used in software plagiarism using program dependence graphs [3]. Sometimes software plagiarism occurs in the models in a very similar way.
- **Copyright infringement:** One important application of code clones detection is copyright infringement. It is represented by similarity measuring between source codes of software systems [4].

2.1 Types of Code Clones

Code clone is two similar fragments in two places in source code. It has the main types [1]:

- Type 1: Same parts of codes except for whitespace differences, and/or comments and may also be differences in the planning.
- Type 2: In this code clone type, syntactically or structurally same parts of codes except for the differences in variable layout, names, and comments.
- Type 3: Two code fragments with new changes such as change in statements, removed or added variable, layout and comments.
- Type 4: Two or more code fragments run the similar job but implemented through changed syntactic modifications.

2.2 Clone Detection Processes

The main objective of code clones detector is find similarity parts in the source code of software system. There is only one problem facing us when we try to find the code clones, which we do not know the place of code clones in source code. So, to detect the code clones, the comparison in the detector should compare every possible part of code with every other possible part of code in the original code [5]. The process of code clones detection is shown in Figure (2).

2.2.1 Preprocessing

The preprocessing step found in any code clones detector as an initial step, preprocessing step has three main objectives:

- Removing uninteresting parts: Remove uninteresting parts from the source code by filtered out the uninteresting parts to comparison, like embedded code to attached different languages, such as SQL embedded in Java code.
- Determining source units by splitting the remaining source code after removing the uninteresting parts to fragment. These fragments named source units.
- Determining comparison units: in the preprocessing step we need to determine the comparison units, comparison units is a smallest units of source units, we obtained the comparison units by split the source units. For example: "(if-statement) can be separated to (conditional) expression, (then) and (else) blocks".

2.2.2 Transformation

The comparison unit in the code is mapped to another intermediate internal representation. There are two types of transformation of source code:

- Extraction: In this type, the source code mapped to a suitable input format to be used in the matched algorithm, extraction have three forms: (a) Tokenization: this form divides each line of the code into tokens, (b) Parsing: this form parses the source code to generate abstract syntax tree or a parse tree, and (c) Data and control flow analysis: this form builds from the source code a program dependence

graphs, where the node in program dependence graphs represents the statements, and edges represent the data and control dependencies.

- Normalization: This is an optional type, the main objective of this type is to remove superficial differences like differences in commenting, whitespace and/or identifier names.

2.2.3 Match Detection

The transformed comparison units from the previous step are compared to each other to detect the matches. The result of this step is a set of code clone pair candidates.

2.2.4 Formatting

The code clone pair list is converted to a corresponding clone pair list for the original code base. Source coordinates of each code clone pair obtained in the comparison phase are listed to their locations in the original source code.

2.2.5 Post-processing

Code clones are filtered or ranked in this step, there are two ways using to rank the code clones, they are manual analysis and automated heuristics.

2.2.6 Aggregation

In the final step, the code clones pairs should be aggregated into clone classes to reduce the amount of data, collect overview statistics or to make later analyses.

2.3 Code Clone Detection Techniques

Code clones detection techniques can be used to reduce the clones in the code. These techniques have a similar processing for code clone detection, but they differ in data representation [6]. The following sub-sections summarize these techniques.

2.3.1 Text-Based Technique

The simplest, fastest and oldest way to detect code clone is text based technique. The input is each line of the code. It deals with type-I code clones, and with type-II code clones when there is an additional data transformation. It is fast because it does not execute any syntactical or semantically analysis. Where lexical analysis is "the process of converting a sequence of characters into a sequence of tokens".

2.3.2 Token-Based Technique

This type deals with both of type-I and type-II code clones. This technique takes a sequence of token as input. It converts each line of the code using a lexical analyzer to a sequence of token using lexical analyzer. The tokenization step makes this technique slower than text-based technique.

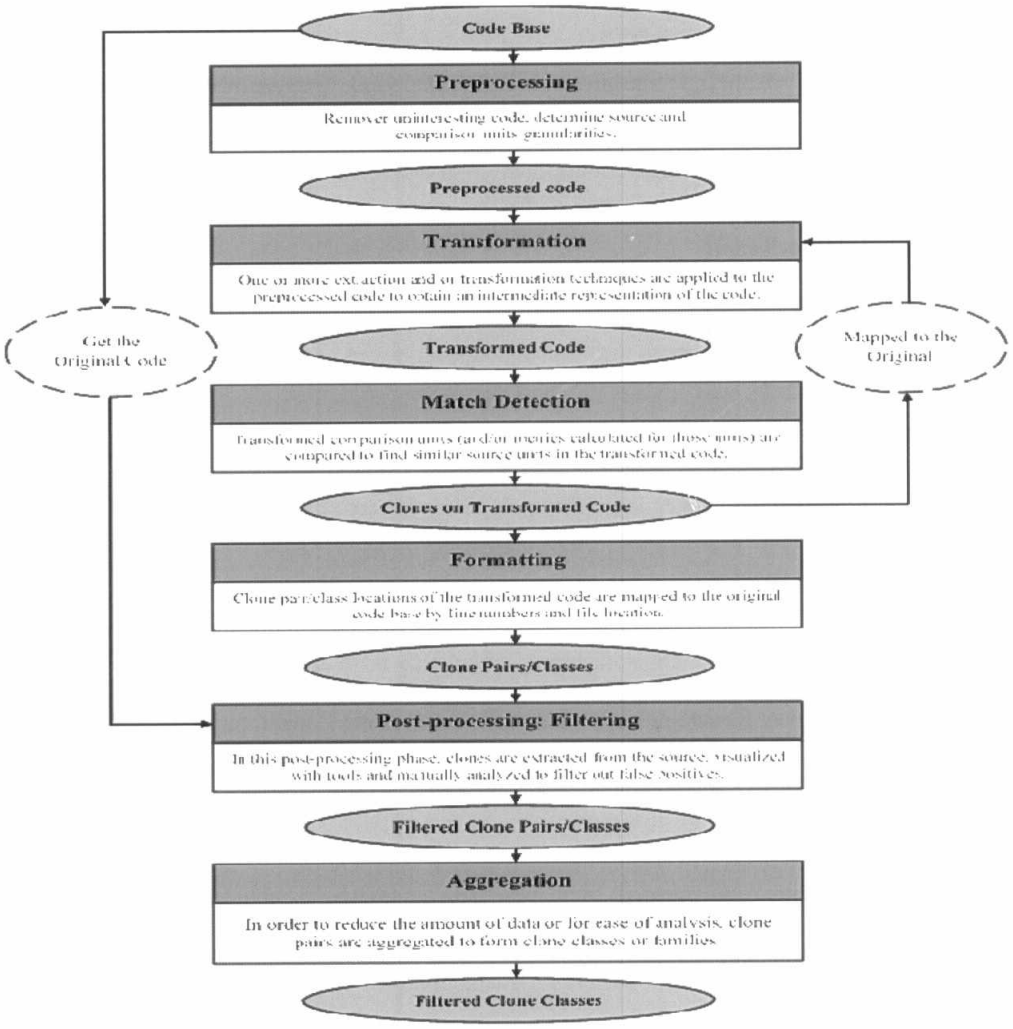


Figure 2: The Process of Code Clones Detection [5]

2.3.3 Abstract Syntax Tree (AST)

AST parses the code to get a syntactical representation and thus calculates the hash code of each sub tree, and compares results with others to find the similar sub-trees in the AST. AST has an advantage in easiest understanding of the code structure because it uses AST as code representation. It deals with all types of code clones.

2.3.4 Program Dependence Graph (PDG)

This technique is based on the relationship between the data and structure because it represents the control and data dependencies. The isomorphic sub graphs are computed following the dependence order from any equal node. So, the code clone pairs can be extracted from the isomorphic sub graph. In addition, PDG is the only technique that has information of control and data dependency that makes this technique able to deal with type 3 code clones precisely.

2.3.5 Metric-Based Technique

Metric-based technique depends on different metrics of source code. By comparing these metric, the code clone pairs are obtained. Also, by parsing the source code, these metrics are calculated into AST and PDG representation. These metrics were calculated form layout, expression, names, and simplicity of functions control flow. A clone is detected only when pairs that have the same metrics values are recognized.

2.3.6 Hybrid Technique

In addition to above code clones detection techniques, many research had been investigated in this area of using a hybrid of different code clone technique to give better accuracy.

2.4 Refactoring Techniques

Refactoring techniques is used to improve the internal structure of software with preserving the external behaviors. The main objective of the refactoring techniques is to increase the quality of software systems. Programmers and software developers use the refactoring techniques to avoid errors like bugs and to provide an easy way for adding features to software systems. Therefore, refactoring techniques preserve the structure's quality of the source code with saving time and effort [7].

2.5 Related Work

Singh and Sharma [8] proposed a hybrid technique based on metric based and text based approaches to detect code clones. They implemented their proposed approach to able to work on different programming language. Li et al. [9] proposed hybrid technique based on metric-based technique and AST techniques for code clone detection depending on four steps: "code standardization, generation of comparison units, node mapping, and similarity calculation". In addition, they tested the proposed approach only in C language.

Sajnani et al. [10] proposed code clones detector uses a token-based technique, named SourcererCC. The proposed detector based on improved inverted index to quickly query the potential code clones of an assumed code block. Kaur and Lal [11] proposed hybrid technique based on text-based technique and metric-based technique for code clone detection. In this approach, first metric-based technique is used to detect any presented potential code clone, then template conversion is done following by text-based technique comparison is executed.

Koschke et al. [12] proposed a code clones detector based on AST. The proposed approach can detect syntactic code clones. They proved that their approach is linear space and time. Sarkar et al. [13] proposed hybrid technique based on metrics-based technique, PDG and AST techniques. They concluded that the process of code clone discovering has higher reliability.

3. The Proposed System

In this paper, we propose an automated refactoring system to remove code clones type-1 resulting from copy and paste. This system works on methods level in Java open source code projects in which Java Code Clone Detection (JCCD) tool [14] is used. Figure (3) shows the process of (JCCD) tool. The used tool for code clones detection in our proposed system is based on AST-based technique shown in Figure (4).

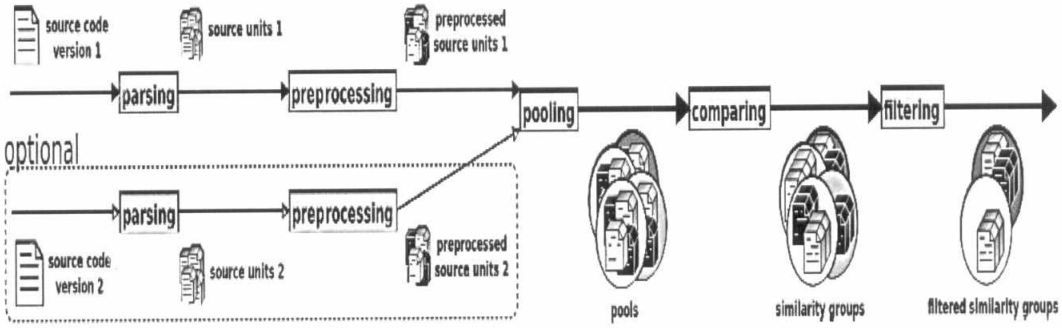


Figure 3: The Process of (JCCD) Tool [14]

```
1. Clones=∅
2. For each subtree i:
      If mass(i)>=MassThreshold
      Then hash i to bucket
3. For each subtree i and j in the same bucket
      If CompareTree(i,j) > SimilarityThreshold
         Then { For each subtree s of i
                   If IsMember(Clones,s)
                   Then RemoveClonePair(Clones,s)
                For each subtree s of j
                   If IsMember(Clones,s)
                   Then RemoveClonePair(Clones,s)
                AddClonePair(Clones,i,j)
              }
```

Figure 4: AST Algorithm [15]

Figure (5) shows a flow chart of the proposed system. In the proposed automated refactoring system, the processes of removing code clones depend on giving a weight for each method. These weights are based on number of calling times in the source code. In addition, the multithreading concept is used to reduce the execution time. In general, we represent the proposed system in five steps which listed below:

1. Detect the Code Clones
2. Extracting All Clones Methods' Names
3. Deciding which Methods should be removed
4. Removing Code Clones

5. Fixing Compilation Errors

Figure (6) shows a case study consists of a class ReTest that contains the main method and two main methods namely (zyadAdd1) and (zyadAdd2). (zyadAdd2) method is a copy of (zyadAdd1) in which the difference in two methods is variable names. The work of both methods is just collecting two numbers. In this example, the main objective is to apply the proposed system to remove code cloned methods. In our proposed system, the process of removing code clones is not random, in general, each method is given a weight based on number of calling times, and then the methods that have less weight are removed with keeping the method with the highest weight.

When applying the proposed system to the used case study, the following actions are occurred as shown in Figure (7):

1. (zyadAdd1) method is kept because it has the highest weight while the other method (zyadAdd2) is removed.
2. The access level of remaining method is changed from private to public. This action is highly needed specially when we deal with code clone existed in more than one class in the source code.
3. The calling method is fixed by replacing the name of removed method with the name of remaining method.

Figure (8) shows the case study after refactoring. When this class is executed before and after refactoring, we get the same results. Thus, we have improved the quality of software system as the main objective of refactoring technique through changing the internal structure of the software system with preserving the external behavior of it.

4. Experimentations and Results

To apply the proposed system, four Java open source code projects are used to evaluate the proposed system through measuring the quality of source code including lines of code, blank line, method's count, and cyclomatic complexity using NetBeans plugin.

Table (1) shows the total lines of code, blank lines and total methods Count in four projects after refactoring the source code has decreased. In addition, the average cyclomatic complexity has decreased after refactoring the source code of the four projects, where Cyclomatic Complexity *is "a software metric used to measure the complexity of a program. These metric, measures independent paths through program source code. This metric was developed by Thomas J. McCabe in 1976 based on a control flow representation of the program*".

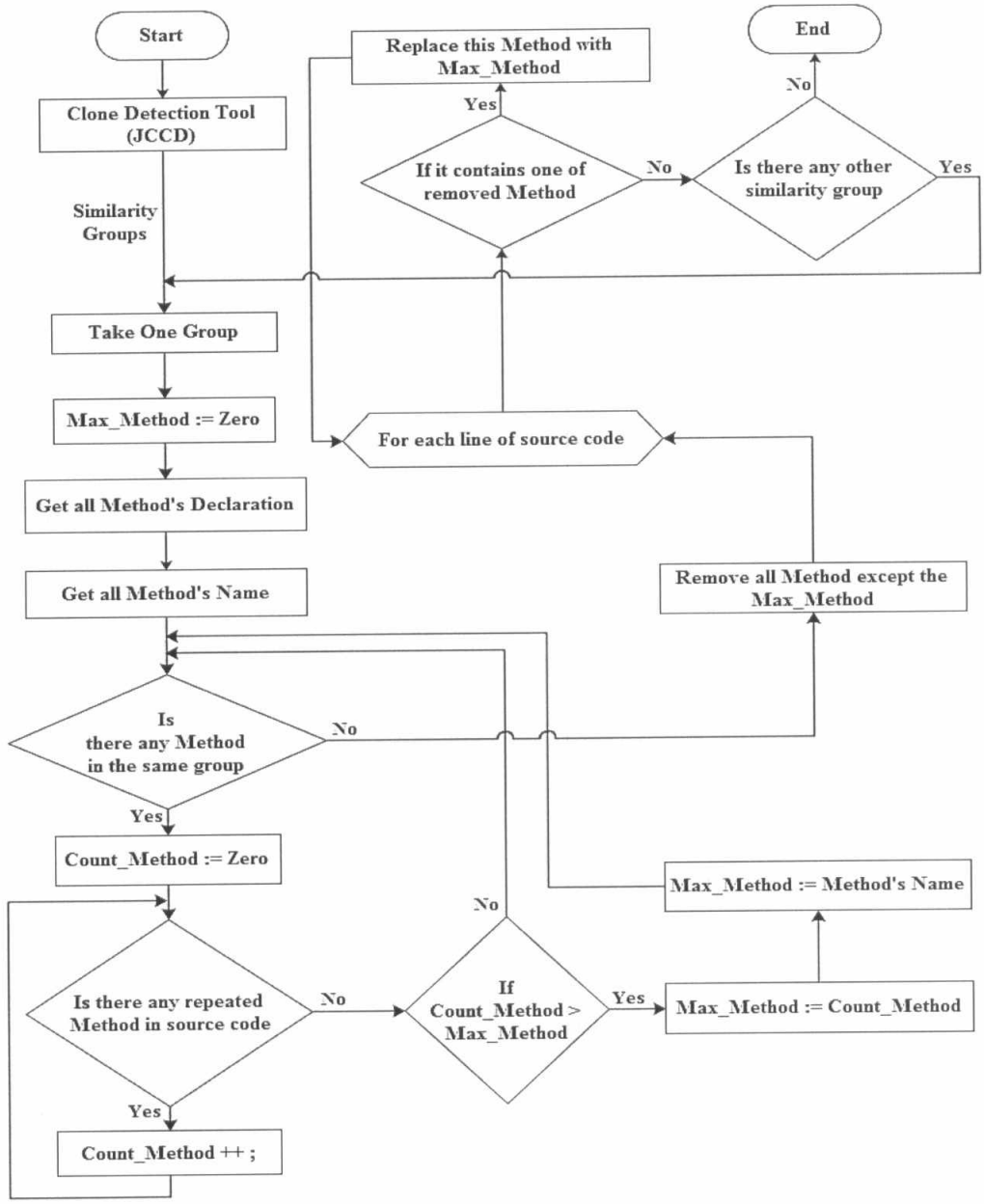
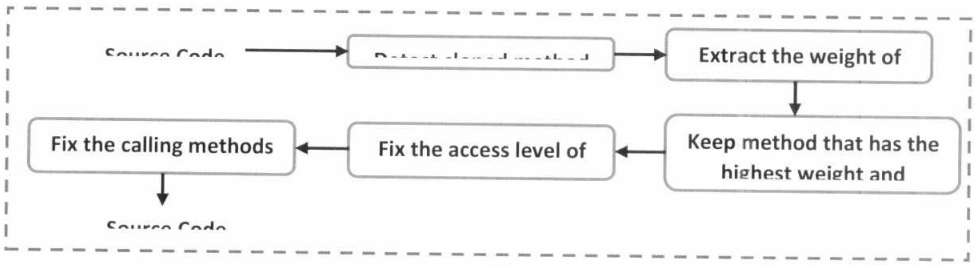


Figure 5: The flow chart of the proposed system

```
1   package retest;
2
3   public class ReTest {
4
5       public static void main(String[] args) {
6
7           int x = zyadAdd1(4, 6);
8           System.out.println("x =  " + x);
9
10          int y = zyadAdd1(3, 6);
11          System.out.println("y =  " + y);
12
13          int z = zyadAdd2(2, 6);
14          System.out.println("z =  "+z);
15
16      }
17
18      private static int zyadAdd1(int x, int y) {
19          return x + y;
20      }
21
22      private static int zyadAdd2(int x, int y) {
23          // This is Cloned Method by Copy and Paste
24          return x + y;
25      }
26
27  }
28  Code Clone:        Method Calling:
```

Figure 6: The case study before refactoring

Figure 7: The actions of the proposed system



```
1   package retest;
2
3   public class ReTest {
4
5       public static void main(String[] args) {
6           int x = zyadAdd1(4, 6);
7           System.out.println("x =  " + x);
8           int y = zyadAdd1(3, 6);
9           System.out.println("y =  " + y);
10          int z = zyadAdd1(2, 6);
11          System.out.println("z =  " + z);
12      }
13
14      public static int zyadAdd1(int x, int y) {
15          return x + y;
16      }
17  }
18
```

Figure 8: The case study after refactoring

Table 1: Project's Metrics

Project's Name	Properties	Before Refactoring (The Original Code)	After Refactoring (Using the Proposed System)
JFreeChart [16]	Total Line of Code	306207	232514
	Total Blank Lines	27758	16230
	Total Methods Count	10206	10071
	Average Cyclomatic Complexity	1.97844	1.98014
JRuby [17]	Total Line of Code	209796	174108
	Total Blank Lines	27318	21385
	Total Methods Count	12304	12233
	Average Cyclomatic Complexity	2.21740	2.21540
JCommon [16]	Total Line of Code	65423	45225
	Total Blank Lines	6223	3974
	Total Methods Count	1937	1919
	Average Cyclomatic Complexity	2.00877	2.00990
	Total Line of Code	262042	221529

Apache ant [18]	Total Blank Lines	27574	21860
	Total Methods Count	10894	10810
	Average Cyclomatic Complexity	1.97104	1.97090

5. Conclusions

Through this paper, we proposed an automated refactoring system for removing code clones from Java open source code project, through removing cloned methods arising from (copy and paste). The results of applying the proposed system on a case study of four Java open source projects shows that it improves the software quality by removing cloned codes with the possibility of obtaining more safely to keep the external behavior of the Java open source code project based on the total lines of code, blank lines, total methods count and average cyclomatic complexity metrics for the four used Java open source projects. In future work, we will try to apply automated refactoring on other difficult types of code clones.

References

1. C. K. Roy and J. R. Cordy, "A survey on software clone detection research," Technical Report 541, Queen's University at Kingston, 2007.
2. E. Merlo. Detection of Plagiarism in University Projects Using Metrics-based Spectral Similarity. In Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, Dagstuhl, Germany, Dagstuhl, 2007.
3. C. Liu, C. Chen, J. Han and P. S. Yu, "GPLAG: detection of software plagiarism by program dependence graph analysis," Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, Pages 872-881, August 20-23, 2006, Philadelphia, PA, USA.
4. B. S. Baker, "A program for identifying duplicated code," Computing Science and Statistics, pp. 49-49, 1993.
5. C. K. Roy, J. R. Cordy and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming,* vol. 74, no. 7, pp. 470-495, 2009.
6. Y. Jia and M. Harman, "Clone Detection Using Dependence Analysis and Lexical Analysis," M.Sc. project, *Department of Computer Science, King's College London, UK,* 2007.
7. E. Kodhai, V. Vijayakumar, G. Balabaskaran, T. Stalin and B. Kanagaraj, "Method level detection and removal of code clones in C and Java programs using refactoring," *Int. J. Comput. Commun. Inf. Syst.(IJCCIS),* vol. 2, no. 1, pp. 93-95, 2010.
8. M. Singh and V. Sharma, "Detection of File Level Clone for High Level Cloning," *Procedia Computer Science,* vol. 57, pp. 915-922, 2015.
9. W. Li, D. Li, C. Qiu and J. Hou, "Efficient Metric Vector-Based Code Clone Detection Using Function-calling Tree," *International Journal of Hybrid Information Technology,* vol. 8, pp. 139-150, 2015.
10. H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy and C. V. Lopes, "SourcererCC: Scaling Code Clone Detection to Big Code," In Proceedings of the 38th International Conference on Software

Engineering (ICSE 2016), 12 pp., Austin, TX, May 2016 (Accepted for publication). arXiv preprint arXiv:1512.06448, 2015, http://arxiv.org/pdf/1512.06448v1.pdf, Last Acces: April 2016.

11. M. Kaur and M. Lal, "Code Clone Detection Using Function Based Similarities and Metrics," International Journal of Emerging Research in Management &Technology, vol. 4, no. 7, 2015.
12. R. Koschke, R. Falke and P. Frenzel, "Clone Detection Using Abstract Syntax Suffix Trees," *2006 13th Working Conference on Reverse Engineering*, Benevento, 2006, pp. 253-262.
13. M. Sarkar, S. Chudamani, S. Roy and N. Mukherjee, "A Hybrid Clone Detection Technique for Estimation of Resource Requirements of a Job," *Advanced Computing and Communication Technologies (ACCT), 2013 Third International Conference on*, Rohtak, 2013, pp. 174-181
14. B. Biegel and S. Diehl, "Highly Configurable and Extensible Code Clone Detection," 2010 17th Working Conference on Reverse Engineering, Beverly, MA, 2010, pp. 237-241.
15. I. D. Baxter, A. Yahin, L. Moura, M. S. Anna and L. Bier, "Clone detection using abstract syntax trees," in Proceedings., International Conference on Software Maintenance, Bethesda, 16-20 Nov 1998.
16. "JFree," [Online]. Available: http://www.jfree.org/jfreechart/. [Accessed 25/1/2016].
17. "JRuby," [Online]. Available: http://jruby.org/. [Accessed 25/1/2016].
18. "THE APACHE ANT PROJECT," [Online]. Available: https://ant.apache.org/bindownload.cgi. [Accessed 25/1/2016].
19. M. Rieger and S. Ducasse. Visual detection of duplicated code. In ´ Stephane Ducasse and Joachim Weisbrod, editors, ´ Proceedings ECOOP Workshop on Experiences in Object-Oriented Re-Engineering, number 6/7/98 in FZI Report. Forschungszentrum Informatik Karlsruhe, 1998.
20. "NetBeans," [Online]. Available: http://plugins.netbeans.org/plugin/9494/simple-code-metrics. [Accessed 25/1/2016].