International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-9 Issue-2, December, 2019

A Novel Clone-Based Reuse Method to Maintain Proficiency in Software Engineering Practice

Kavitha Esther Rajakumari

Abstract – The source code of an application paves way for a quality software product. Quality software in-turn helps in imposing software reuse. In this paper, pieces of similar codes also known as code clones or code duplications are considered as reusable software components. In general code clones are considered harmful in software engineering practice. They are considered to degrade the quality of software. Code clones are detected and removed without further processing. In this paper, a token- based CodeClone reuse method is proposed to detect type-1 and type-4 clones. Positive effects of clones are analyzed and beneficial clones are extracted from the cluster of clones detected. The proposed method aids in the art of developing software thereby enforcing the concept of software reuse. The working principle of the proposed method is implemented using open source software as inputs. Beneficial clones are further stored in a database for future use. Clone report is generated as it assists in knowing about the clone details within a software system.

Index Terms **- Code clones, software engineering, beneficial clones, software reuse.**

I. INTRODUCTION

Similar or exact pieces of code segments are known as clones. There are four categories of clones namely, type-1: copy clones, type-2: renamed clones, type-3: modified clones and type-4: functional clones. As the name implies, the detected clones are classified under each category. Clones are introduced in a software application either intentionally or accidentally. When different programmers work on different parts of components of an application, unintentionally clones are introduced [10]. Code clones are considered as bad smells as they lead to unpredictable behaviour of the system [9]. Though clones are considered as negative factors, careful analysis of clones has exhibited the positive aspects of their presence in the system [7, 8]. They support software evolution research and aids in finding usage patterns.

Revised Manuscript Received on December 30, 2019.
* Correspondence Author
Dr.Kavitha Esther Rajakumari, Dept. Of Computer Science And Engineering, Kcg College Of Technology, Chennai-97 Kavitha.Cse@Kcgcollege.Com

Clones also assist in software engineering paradigm. Useful code clones can be made into library candidates. In order to evaluate the clones, initially they must be detected. For detecting clones a handful of techniques are available. Based on the type of clones the detection technique can be chosen. Code clones if used positively, assists in software maintenance [3]. Maintenance is an important phase in software engineering. Most of the time is spent in this phase. Careful diagnosis of codes will reduce the overall time consumption. Software reuse is another important aspect of software engineering. New software developers and programmers will be greatly benefitted thereby introducing reuse strategy. In this paper, the main focus is on code clone usage which establishes the reuse policy in software development.

II. LITERATURE SURVEY

Clones are detected using customized automated tools. Hamid and Stan, have detected structural clones by using CloneMiner tool. They have enforced the reuse of structural code clones while developing software system [1]. Structural information of the codes is used to find similar code segments at the function-level [11]. Moha et.al, have proposed a new method to detect code clones. By combining different approaches they have presented this DECOR tool [6]. Clones can be detected more precisely by using abstract syntax tree, provided a special parser is needed to perform the operation [5]. Cloned codes are considered stable when compared to non-cloned codes [2]. Code clones if used positively will help in building software variants within a short period of time. So code clone concept should be integrated in systematic reuse policy [4].
Inference from the survey shows that main focus is on clone detection. The more efficient tool used, the more clones are obtained. Clones are considered harmful but careful analysis has showed that they are positively stable too. Therefore clones can be reused after careful examination.

III. PROPOSED METHODOLOGY

In this proposed methodology Token-Based CodeClone Reuse Method, token-based approach is used for effective detection of type-1i.e. exact code fragments and type-4 i.e. functional clones. Moreover the concept of software component reuse is integrated along with the detection approach. The two main emphasis of the proposed methodology are on code clone detection and their reuse.

The input for the proposed system is Java software applications.
The following diagram fig.1depicts the concept of proposed methodology.

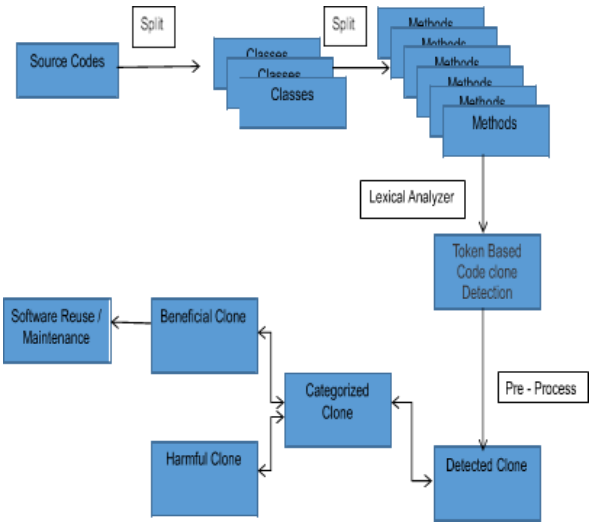

Fig.1. System architecture of Token-Based Code Clone Reuse Method

In this token-based methodology, the entire software source code is parsed as a sequence of tokens. The resultant sequences are scanned for exact functional clones. The Token-Based Code Clone Reuse Method is comprised of four phases. Each of the phases is elaborated below:

A. *Conditioning Phase*

In this phase, all Java files are scanned for cleaning the source codes. Here cleaning implies the removal of braces and blank spaces. The resulting codes are restructured to a standard form which is needed to find exact and similar code clone fragments.

B. *Splitting of Classes and Methods*

Here, the cleaned Java source files are taken as inputs. Line-by-line parsing is done to extract classes, methods and main methods. Count and line of occurrences of functions are obtained. Even the loops are obtained along with classes and methods. This phase is for knowing the number of occurrences of functional clones within a software system.

C. *Clone Detection Phase*

The detection process is carried out in an iterative fashion. Each file is compared with every other file and the clone results are stored. The advantage of this technique is that, checked file of the previous iteration is excluded from further comparison.

D. *Segregation Phase*

Here, the detected clones are classified into harmful and beneficial clones. Based on the quality and severity of harmful clones, either they are rejected or rectified and stored along with beneficial clones. Beneficial clones are stored in database for future use.

The Token-Based CodeClone Reuse Method comprises of the following algorithms:

Algorithm 1: Class Method Counter Algorithm
Input: Java Source code
Output: Classes, Methods, Main Methods

```
begin
    for each jp i to n do
    begin // class counter
          ccnt=1;
          while (readline!=null) do
          begin
                if readline contains "class"
then
                cf.add(ccnt);
                ccnt++;
           end
      end
begin // method counter
       mcnt=1;
        while (readline!=null) do
        begin
              if readline ends with ")" then
              mf.add(mcnt);
              mcnt++;
         end
 end
begin // main method counter
         mmcnt=1;
         while (readline!=null) do
         begin
           if readline contains "String args[]) or
String [] args)" then
           mmf.add(mmcnt);
           mmcnt++;
          end
    end
 end
```

Algorithm 2: Clone Detection Algorithm
Input: Java Files
Output: Code Clones

```
begin
      for each jp i to n-1 do
      begin
            for each jp j to n do
            begin
                while (readline1!=null) do
                 begin
                      fn1.add(readline1);
                 end
                 while (readline2!=null) do
                 begin
                      fn2.add(readline2);
                 end
                cct=0;

clounfound[]=findduplicate(fn1,fn2);
                stores the clone found lines of
file 1 and 2 in array.
                for each clonefound[] i to n do
                begin
                     if val notequal "{ or }"
                     then
                          cct++;
                end
                if cct!=0 then
                          cctr.add(cct);
          end
          end
    end
```

Algorithm 3: Segregation Algorithm
Input: Detected Code Clones
Output: Separation of Harmful Clones

```
 begin
str[]=clonefound[];
tokenizer st equal to tokenizer (str,",");
fcnt=0;
cnt=0;
ccnt=0;
hcnt=0;
while st has more tokens do
begin
      str1=tokens;
      if str1 contains "for( or while( or do" the
            hcnt++;
end
tokenizer st1=tokenizer (str,",");
while st1 has more tokens do
begin
      if hcnt ≥ 5 then
            cnt++;
      if str1 ends with ")" then
            fcnt++;
 end
 ccnt++;
end
```

Benefits of Token-based CodeClone Reuse Method:

- Acquisition of beneficial or useful code clones
- Removal of harmful or unwanted code clones
- Simple positioning and rectification of problematic codes

IV. RESULTS AND DISCUSSIONS

The implementation was carried out with Core Java and J2EE for front end design and back end support was provided using MySql. The types of software used are JDK, Apache Tomcat server and XAMPP. Java Development Kit JDK (64 bit) consists of tools and software required for compiling, debugging, and run the Java applications. XAMPP is free and open source cross-platform web server package developed by Apache. It consists of Apache HTTP Server, MariaDB database, and interpreters for PHP and Perl scripts.

The working of the Token-based CodeClone Reuse Method with an open source software as input is shown here. Mysql-database-1.8.0 and Mysql-database-

1.8.9 are taken as inputs. The clone detection procedure starts with the upload of folder containing Java source files. The number of classes, number of methods, their line numbers and total number of lines in the files along with their file names of the student online software are displayed as in Table 1.

Table-I: Class method counter status

S. No	Filename	File class count	File class line number	File method count	File method line number	File main method line number	File total lines
1	BufferedReaderExample.java	1	5	5	7,15,24,30,40	7	47
2	CountFilesInDirectory.java	1	3	6	4,9,11,17,18,24	4	32
3	demo.java	1	1	2	3,7	3	17
4	input.java	2	1,8	5	3,10,15,18,22	15	28
5	input1.java	2	1,8	4	3,10,13,17	no main method	23
6	onearr.java	1	2	11	4,12,17,22, 24,26, 34,39, 45,47, 52	4	62
7	onearray.java	1	2	11	4,12,17,22,24, 26,34, 39,45, 47,52	4	62
8	s.java	1	1	1	3	3	9
9	SimilarityCalculationDemo. java	1	24	6	33,39, 46,49, 61,63	39	72
10	twoarr.java	1	1	10	5,7,9,16,19,21, 28, 32,35, 44	44	52
11	twoarray.java	1	1	10	5,7,9,16,19,21, 28, 32,35, 44	44	52

Table 2 provides the clone detection report generated by the Token-based CodeClone Reuse method after the detection process.

Table-II: Clone detection report

S.No	Filename 1	Filename 2	Clone count	Clone code	View clone
1	BufferedReaderExample.java	CountFilesInDirectory.java	1	try	ViewClone
2	BufferedReaderExample.java	demo.java	1	try	ViewClone
3	BufferedReaderExample.java	input.java	1	try	ViewClone
4	BufferedReaderExample.java	input1.java	1	try	ViewClone
5	BufferedReaderExample.java	onearr.java	1	for(int i=0; i<n; i++)try	ViewClone
6	BufferedReaderExample.java	onearray.java	1	for(int i=0; i<n; i++)try	ViewClone

Table 3 illustrates the clone detection process in which two files are investigated for clone. It provides the number of classes, iterations, functional clones, percentage of beneficial clones and percentage of harmful clones. Fig.2 depicts the percentage of clones.

Table-III: Clone details

Total clone count	29
Iteration count	6
Functional clone count	4
Harmful clone	for(i=0;i<3;i++), for(j=0;j<3;j++), for(int k=0;k<3;k++), for(int l=0;l<3;l++) while(x<3), while(y<3)
Beneficial clone percentage	79.31%
Harmful clone percentage	20.69%

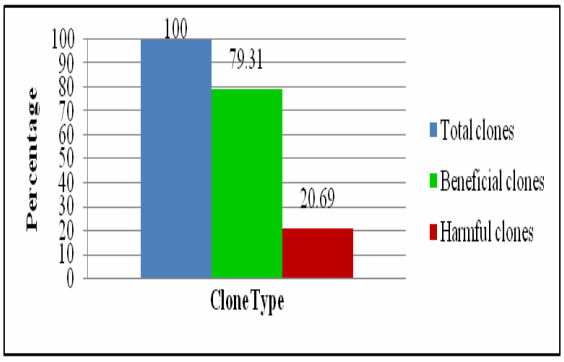

Fig.2. Percentage of Clones

Table 4 provides the reusable clones present in two files twoarr.java and twoarray.java.

Table-IV: Reusable clones

RC ID	Filename 1	Filename 2	Clone count	Iteration count	Functional clones count	Beneficial clones percentage	Harmful clones percentage	Reusable view
1	twoarr.java	twoarray.java	29	6	4	79.31	20.69	View

One of the major benefits of this technique is that it separates the harmful clones from beneficial clones which degrade the quality of software. Table 5 depicts the storage of clone details in database highlighting the function name, line number, number of functions, method name, line number and the number of methods similar to table 1. The clones can be copied, edited and reused for future use.

Table-V: Storage of clone details in database

S. No	Filename	File class count	File class line number	File method count	File method line number	File lines
1	BufferedReaderExample.java	1	5	5	7,15,24,30,40	47
2	CountFilesInDirectory.java	1	3	6	4,9,11,17,18,24	32
3	demo.java	1	1	2	3,7	17
4	input.java	2	1,8	5	3,10,15,18,22	28
5	input1.java	2	1,8	4	3,10,13,17	23
6	onearr.java	1	2	11	4,12,17,22,24, 26,34,39, 45,47,52	62
7	onearray.java	1	2	11	4,12,17,22,24,26, 34,39,45, 47,52	62
8	s.java	1	1	1	3	9
9	SimilarityCalculationDemo.java	1	24	6	33,39,46, 49,61,63	72
10	twoarr.java	1	1	10	5,7,9,16,19,21, 28,32,35, 44	52
11	twoarray.java	1	1	10	5,7,9,16,19,21, 28,32,35, 44	52

V. CONCLUSION AND FUTURE WORKS

The proposed template based on code clones aids in successful software engineering. The proposed customized Token-based Code Clone Reuse Method was used to detect exact functional clones from a given Java software application. The loops present within the software are considered to be harmful as it degrades the performance of software. Acquired beneficial clones are stored in a database for reuse. Clone counts, iteration counts and functional clones counts are displayed which helps in analyzing the application in detail.

Harmful clones are not removed; they are either eliminated or rectified based on the usage of the software application.

Quality software can be built by reusing the beneficial code clones. Cost and time spent in software development and maintenance are considerably reduced. Version enhancement need not be done from scratch. Enforces reuse-based software development at low cost and enables source code analysis. Detected beneficial clones can be built into useful library candidates. The proposed methodology reduces the testing time considerably. Positioning and rectification feature helps to reduce the complexity involved in software maintenance.

Token-Based Code Clone Reuse Method can be extended to support different programming languages. Beneficial clones' database can be further enhanced by domain-wise segregation.

REFERENCES

1. Basit, H. A., & Jarzabek, S. (2009). A data mining approach for detecting higher-level clones in software. IEEE Transactions on Software engineering, (4), 497-514.
2. Gode, N., & Harder, J. (2011, March). Clone stability. In Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on (pp. 65-74). IEEE.
3. Lin, Y., Xing, Z., Xue, Y., Liu, Y., Peng, X., Sun, J., & Zhao, W. (2014). Detecting differences across multiple instances of code clones. In Proceedings of the 36th International Conference on Software Engineering (pp. 164-174). ACM.
4. Linsbauer, L., Lopez-Herrejon, R. E., & Egyed, A. (2017). Variability extraction and modeling for product variants. Software & Systems Modeling, 16(4), 1179-1199.
5. Matsushita, T., & Sasano, I. (2017). Detecting code clones with gaps by function applications. In Proceedings of the 2017 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation (pp. 12-22). ACM.
6. Moha, N., Gueheneuc, Y. G., & Duchien, A. F. (2010). Decor: A method for the specification and detection of code and design smells. IEEE Transactions on Software Engineering (TSE), 36(1), 20-36.
7. Rezaei, A., Mueller, F., Hargrove, P., & Roman, E. (2017). DINO: Divergent node cloning for sustained redundancy in HPC. Journal of Parallel and Distributed Computing, 109, 350-362.
8. Roy, C. K., Zibran, M. F., & Koschke, R. (2014, February). The vision of software clone management: Past, present, and future (keynote paper). In 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE) (pp. 18-33). IEEE.
9. Sjoberg, D. I., Yamashita, A., Anda, B. C., Mockus, A., & Dyba, T. (2013). Quantifying theeffect of code smells on maintenance effort. IEEE Transactions on Software Engineering, (8), 1144-1156.
10. Tajima, R., Nagura, M., & Takada, S. (2018, March). Detecting functionally similar code within the same project. In Software Clones (IWSC), 2018 IEEE 12th International Workshop on(pp.51-57). IEEE.
11. Yang, Y., Ren, Z., Chen, X., & Jiang, H. (2018, July). Structural Function BasedCode Clone Detection Using a New Hybrid Technique. In 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC) (pp. 286-291). IEEE

AUTHORS PROFILE

Dr. Kavitha Esther Rajakumari, The author has completed PhD in the field of software engineering in October 2017. She has published 21 papers in international and national conferences and Scopus indexed journals. Currently her research work is ongoing in the field of software engineering, blockchain, cloud computing and networks. She is a Life Member in ISTE. She has conducted and organized seminar, guest lecture and certification courses in computer networks, cloud computing and PHP respectively.