A Survey on Trending Algorithms for Software Code Clone Detection

Rashmi Sharma
Department of Computer Engineering,
Punjabi University, Punjab
Patiala, India

Brahmaleen Kaur Sidhu
Department of Computer Engineering,
Punjabi University, Punjab
Patiala, India

ABSTRACT
Most of the developers indulged in the coding phase of SDLC, try to copy the code that occurs again and again in the code, hence it becomes difficult to maintain the cloned data. If two functions or templates from a single source code are similar then it would be referred as "code clones". Cloning in the code can lead to the obstacles in the maintenance phase of the software. It also increases the probability corresponding to the occurrence of bugs in the software. When a code is reused by copy-paste, then it referred as "software clone". In order to detect the clone from the source code each and every template of the code is evaluated corresponding to the source code. The detection of clone is an issue hence various techniques had been developed in previous research works by various researchers for the detection of clone. In this study a brief introduction is given about the clones in the code, its types, reason of cloning, and process of clone detection. The second section depicts the clone detection techniques with their limitations and advantages. The traditional work conducted in this field is described in the third section of the study under the segment of related work.

Keywords
Source Code, Code Clone, Fragments, Lexical Clone, Semantic Clone, Syntactical Clone, Textual Clone

1. INTRODUCTION
Cloning mostly exists at the time when the code of the software is being written. In coding phase, developers frequently copy and paste the code which is being reused in coding [1]. This pasted copy of the code is known as the clone of the original data. The code that has been copied is known as the code cloning. It is very tedious task to recognize the original code and clone of the original code. The part of the code copied is known as fragments. It is not easy to maintain the clone of the code as compared to the original code [2]. Hence there is a need to remove the clones from the code, since cloning leaves adverse effects on the maintenance phase of the software [3].

Let's assume that there exists a system with huge amount of coding and the whole code is the clone of original code. Due to the cloning, it becomes quite expensive to maintain such system as compared to other systems. It is very difficult task to remove or find clones from such systems. The copy, paste leads to the in-accuracy in the maintenance of such system. Lots of research has been conducted to generate such technique that can find the cloning automatically when it exist [4].

Example of code cloning:

```
Int sum=0;
Void foo (iteratoriter)
{
For (item=first(iter): has more (iter); item =next(iter))
{
Sum=sum + value (item);
}
}
Int bar (Iteratoriter)
{
Int sum=0;
For(item=first(iter); has more (iter); item=next (iter))
{
Sum =sum +value (item);
}
}
```

In above ode two function have the similar statements. The function for() has same number of statements as well as structure to the function bar(). Hence this code is useful to understand the concept of code clone in an effective and easy way.

2. REASON BEHIND CODE CLONNING
Even though, the tactics of cut-copy and paste are measured not to be good from the aspect of software maintenance, lots of programmers uses this technique for their coding [5]. Here is a list of reasons that exist behind the code cloning:

2.1 Programmer's limitation and time constraints:
The software code is written or created under perfect surroundings [6]. The limited skills of professional or developer and the time constraints slowdown the process of software development, hence the only way out is to cut or copy and paste.

2.2 Complexity of the system:
Some systems or software are difficult to understand hence this problem leads to the reuse of the existing code of lines by the developers just by doing few alterations to this code [7].

2.3 Language limitations:
There are lots of programming languages available in the market but it is not possible that the every programmer should have the full fledge knowledge about particular programming language. Hence the language limitation is another reason behind the code cloning [7].

2.4 Phobia of fresh code:
The software developer fears to bring the new ideas to the market. There is a phobia in the developers which restricts them to create new innovations with the software because the introduction to ideas can lead to the lengthy software development process [7].

2.5 Forking/Templating:

Forking or templating is a process to reprocess the same solution more than once. This is again an aspect which motivates the developers to clone the code [7].

3. TYPES OF CLONE

As studied in previous sections about the code cloning and the reasons that give birth to the concept of cloning, in this section we define the types of clone. The clones are broadly categorized in four sections as follows [9]:

3.1 ***Type-1:*** Identical rule pieces except for modifications in whitespace, structure and feedback.

3.2 ***Type-2***: Syntactically similar pieces except for modifications in identifiers, literals, kinds, whitespace, structure and feedback [9].

3.3 ***Type-3:*** Copied pieces with further modifications such as changed, added or removed statements, in addition to modifications in identifiers, literals, kinds, whitespace, structure and feedback.

3.4 ***Type-4:*** Two or more rule pieces that perform the same calculations but are implemented by different syntactic versions [9].

4. ADVANTAGES OF DETECTING CODE CLONES

The advantages of code clone detection are as below [10]:

- Detects the code fragments to put into the library file
- Helps to have a clear understanding to the program
- Finds usage patterns
- Detects malicious software
- Detects plagiarism and copyright infringement
- Helps software evolution research
- Helps in code compacting

5. CLONE DETECTION PROCESS

Clone detection refers to the process of detecting the duplicate code in the source of the software [10]. For detecting these clones, clone detector software is used. The work of clone detector is to find out the duplicate line of code with the higher rate of similarity in the source code. The process of code clone detection is started by firstly comparing each and every possible fragment of the code with the rest of the code, because initially it is not known to the detector that which code from the source code is cloned. After detecting the cloned fragments from the source code, next step is to apply the tool that look out for actual clone in the code [10].

6. CLONE TERMINOLOGY

The cloned code is recognized in the form of clone classes and clone pairs. Clone classes and pairs are used to depict the match among the different code clone templates. If some similarities are found among the cloned code then it means that some relationship exists between the codes. The frequently used terms in code clone are as follows:

- Code Fragment
- Clone Pair
- Clone Set
- Clone Class

7. TECHNIQUES FOR CLONE DETECTION

From last few years, clone detection has been the most prominent area for the research work. Large number of clone detection techniques had been proposed by various scholars. This section reveals the techniques used for clone detection along with their limitations. The techniques for clone detection are divided into four main categories which are as shown:

- Textual Approach for clone detection
- Syntactic Approach for clone detection
- Lexical Approach for clone detection
- Semantic Approach for clone detection

7.1 Textual Approach:

Textual approach is a text based technique for clone detection. This method did not leave any changes or effects on the original source code before evaluating the source code and cloned code. Examples of textual approach to detect the cloned code are SDD, NICAD, and Simian1 etc [10].

Limitations of textual approach [4] [8]:

- As it processes the code line by line, hence it cannot maintain the remaining identifiers.
- The line breaks that exist in code are not referred as clones.
- The addition and subtraction of the brackets to the code can be problematic in case when brackets are found in one fragment of the code but not in other fragment.
- It is not usable in source code modulations due to the required normalization mechanism for the enhancements, without affecting the precision.

7.2 Syntactic Approach

Syntactic approach uses the concept of tree parsing for the detection of cloning in the code. It uses parser for moulding the source code into the form of parse trees or AST (Abstract Syntax Trees) [11]. Then this parsed source code is processed by applying the tree comparison or structure metrics in order to detect the code clone from the source. There is a parse tree based algorithm used for clone detection. It is a complex algorithm which creates the parse tree [11]. Parse tree based algorithm is used in various applications because it allows to add distinct algorithm for comparison also. CloneDr, Deckard, CloneDigger etc are the examples of syntactic approach.

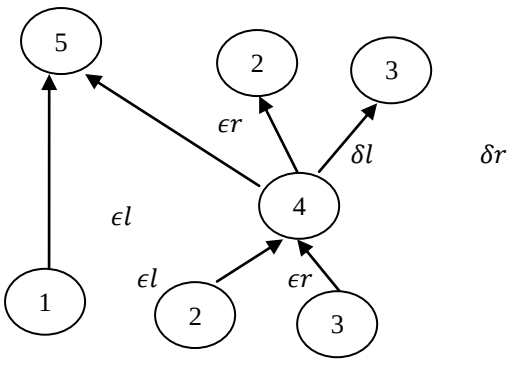

Fig 1 Example of AST based Syntactic Approach [2]

7.2.1 Tree matching Approach:

Tree matching approach is based upon the concept of comparing the sub trees in order to find the similarities among them with the purpose of detecting the clone from the code [12]. The tree represents the literals, identifiers, keywords, variable names and tokens that are abstracted form the source code [9].

Basic Algorithm for AST is as below [23]:

```
Clones=∅
For each subtree i
   If mass (i)>=Threshold
  Then hash i to bucket
For each subtree i and j in the same bucket
   If Compare tree (i,h) > Similarity Threshold
   Then for every subtree s(i)
      If IsMember r(clone s,s)
      Then DeleteClonePair(clone s,s)
   For each subtree s(j)
      If IsMember r(clone s,s)
      Then DeleteClonePair(clone s,s)
AddClonePair(clone i,j)
```

7.2.2 Metrics based Approach:

This approach evaluates or measures the number of metrics that form the code templates and then perform the comparison among them, on the behalf of metrics vectors. The metrics evaluation is performed specifically for classes, control statements, conditional statements, functions and loop statements and then these metrics values are utilized to detect the clone form the code. Mostly, ASTs and Flow Graphs are used for parsing the source code [13].

Limitations of Syntactical Approach:

- This technique is not able to control or manage the literal and identifiers from the source code.
- It is also not capable to detect the saved statement clones.
- The availability of the parser is must.
- On the basis of metrics, the similarities between the two code fragments are not found easily or found to be less effective.

7.3 Lexical Approach

It is also known as token based approach for clone detection. It is much reliable and able to generate more accurate results. In this technique transformation of the code is done by applying algorithms [14]. The transformation algorithm is created by using a stream known as token. This token is extracted from code itself. Transformations performed by using lexer (tokens) make the comparison phase easy. Compiler-style lexical analysis is done in order to change the source code into the sequence of "tokens" [15]. After transformation of the code, the lexer is compared with the original code and duplicity is detected. After the comparison, the original code is referred to as the cloned code and returned to the user. It makes the user capable of making small modulations in the code such as formatting the code, renaming the file etc as compared to other clone detection approaches, hence it makes it more robust than other mechanisms [16]. Lexical approach was initiated by the Baker's Tool Dup, which firstly splits the source file into small tokens by using a lexical analyzer. Then these tokens are further sub divided into parameterized tokens which consists of identifiers and literals as well as the non parameterized tokens. The parameterized tokens are encoded on the basis of their index position which depicts their occurrence in the particular line [3].

The basic algorithm for lexical analysis on the basis of parameterized tokens is as below [3]:

1. function report Clones (filename)
2. Let f be the list of tuples corresponding to *filename* sorted by the statement index either read from the index or evaluated on the fly.
3. Let c a list
   $$c(0) = \emptyset$$
4. For $i = 1\ to\ length(f)$
   Do
5. Retrieve tuples with similar sequence of has as *f(i)*
6. Save this in $c(i)$.
7. For $i = 1\ to\ length(c)$
   Do
8. $if|c(i)| < 2\ or\ c(i) \subseteq c(i-1)$ Then
9. Continue with step 11.
10. Let a=$c(i)$
11. For $j = i + 1\ to\ length(c)$
    Do
12. Let $\acute{a} = a\overline{\cap}c(j)$
13. If $|\acute{a}| < |a| then$
14. Report clones from $c(i)$ to a.
15. $a = \grave{a}$
16. If $|a| < 2or\ a \cong\ c(i-1)$ then
17. End inner loop.

The non parameterized tokens are reviewed by using the hash functions. CC Finder, CP Miner, Dup etc are the example of Lexical clone detection approach [17]. In lexical approach the full fledge source code is referred as input code and then uses each and every basic statement as a module for analyzing purpose. The process of lexical analysis has following steps:

STEP1: Filter uninterested information: As we know it is very cost effective process to detect the clone from the code. Hence in lexical approach the code of interest is separated from the code of uninterested first, so that the efforts or cost can be reduced to detect the clone. The whitespaces and comments in the code fall under the category of uninterested code.

STEP2: Analysis Statements: It is done to save the time for analyzing the code token by token. In this step along with the analysis the following information is gathered [17]:

- Used Data
- Structured Code
- Functional Code

STEP3: Suffix Comparison: Suffix comparison is done to increase the comparison speed by employing the suffix text searching mechanism. This process is initiated by classifying the similar line of code and then suffix for each statement is evaluated, then similar suffix values are extracted from the code [17].

Limitation of lexical approach:

- This approach works upon the sequence of lines in the source code therefore if the order of the lines is modified in the cloned code then it will not recognize the cloned code [18].
- The implementation of lexical analysis is quite time consuming and expensive as compared to other mechanisms with the extra memory requirements.

7.4 Semantic Approach

Semantic based clone detection mechanism provides more reliable and effective information regarding the cloned code in the source code. It is represented in the form of graphs where the nodes of the graph depicts the statements and expressions form the source code and edges of the graph depicts the control along with dependency of data [4][7][9].

The figure below depicts an example of a PDG graph that is used in semantic clone detection approach.

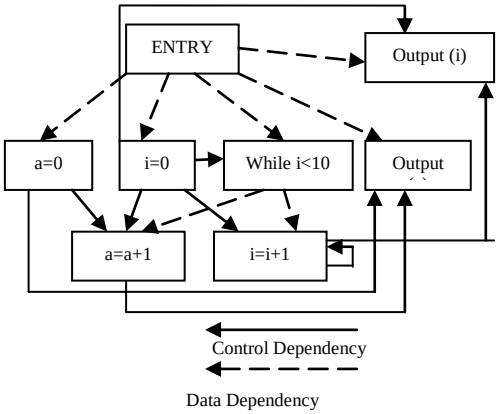

Fig2 Example of Semantic Approach (PDG) [21]

Limitations of Semantic Approach

- Graph based semantic approach is not suitable for the large system hence it is less scalable.
- The availability of the graph generator is mandatory for creating the graphs.
- The incurred cost is high for graph matching.

7.5 Coarse Grained Approach

Coarse grained approach detects the clones on the block level from the source code file. It explains the method for detecting the clones in a set of source code files, followed by the incremental detection description.

ALGORITHM

The detection procedure consists of the following four steps.

- STEP1: Parse given source files to detect the blocks.
- STEP2: Normalizing the blocks detected in step 1.
- STEP3: Calculation of the hash value from the each block.
- STEP4: Grouping the blocks on the basis of the calculated hash values.

STEP1: Detect Blocks: In the first step, all the blocks are detected from the given source code file. Where the blocks includes the classes, methods as well as the block statements i.e. statements like if or for. This step of the block detection requires both the lexical as well as the syntax analysis.

STEP2: Normalize Blocks: In the next step, normalization is done for the block detected in the first step. At the start, reformatting of every block was done with a regularized form. Detector can ignore the differences of the white space and tabs in this procedure. It can also replace variable names and the literal with a special token, allowing the detector to identify the Type-2 clones, thus allowing the detector to detect both the Type-1 and Type-2 clones. But not allowing the detector to identify the Type-3 clones.

STEP3: Calculate Hash Values: In the third step, hash value was calculated from the text of the respective blocks. Implementation uses the hash Code().String as a hash function. Apart from this any of the hash function can be used that can generate a numeric value from the given string.

STEP4: Group Blocks: In the last step, grouping of blocks was done on the basis of their hash values. Two blocks have the same hash values, if the text representations for both the blocks after the process of normalization are equal for both the blocks. Hence, a pair of block represents a clone pair if the hash value is same for the two blocks. The detector reports all the cloned pairs after grouping all the detected blocks with same hash values.

7.6 Fine Grained Approach

Fine grained approach detects the code clone for the source code file under the CVS (Concurrent Version System is a system which allows the user to save the different versions of the software and facilitates its access to multiple users) control by checking the revision of classes by using the CVS capabilities. It divides the Source code into the classes and then compares the methods and attribute of these classes in order to find the revision in these classes. The revised version and the original version were then compared for the cloning.

Step 1: Checking the revision of class.
Step 2: Profiteering the Method and attribute by the compare plug-in
Step 3: Creating the intermediate trees.

8. RELATED WORK

Ira et al. [29] proposed a tree based technique for detecting the clone from the source code. The proposed work was divided into three sub parts by using three algorithms. First algorithm was employed for extracting the sub trees from the code. Then these sub trees were evaluated with another sub trees. And then similarity proves the existence of clone in the code and it was measured on the basis of following equation:

$$2 * \frac{S}{2 * S + L + R} \qquad (1)$$

In above formula (1)

S defines the number of shared clones,
L defines number of differential nodes corresponding to sub tree1,
R depicts the nodes corresponding to the subtree2.

After this, the second algorithm was employed to evaluate the sequence of codes. For example if sub tree was distinguished as a clone in the sequence then that sub-tree was declared as single clone instead of individual clone.

The objective behind employing the third algorithm was to detect the near miss clone in the source code. The process was started by visiting the superior nodes of existing clones to reveal out that whether the superior was near miss clone or not. The work was employed not only to detect the clone from the file but also repair file from these clone. The only drawback was that, the technique was not able to detect the semantic clones from the code.

Yong Yuan et al. [26] proposed an initialized token based way to deal with distinguished clones named as Boreas. Boreas purposed a novel checking based strategy for qualities grids, which depicts the program fragments individually in a powerful way to recognize the clones. Boreas had presented three terms numbering conditions (CE), tally vectors (CV) and check framework (CM) [5]. CE portrays the sample for the variables that was isolated into three phases. These stages were innocent including stage, proclamation checking stage and the inner statement evaluation stage. This stage comprises of utilized variables. In instatement numbering stage, CE was utilized for the variables, where they were used as though predicate, exhibit subscript and where any operation was connected to the factors and where it was characterized by expression with constants. In Inner statement level, CE was utilized as the variable was in first level circle, second level circle or more profound level circle. CV with m measurements was framed by utilizing m CE. The i^{th} measurement of CV was comprised of number of count of that variable for i^{th} CE. Otherwise the CV was called as attributes vector. For n variable utilizing m-dimensional CV m*n Count Matrix was shaped. These CM were the unique type of code parts. Comparability of two parts was measured by the generation of their CM's similarity and likeness of their CV's of the keyword or punctuations. Likeness of two CVs was measured with the equation:

$$Prosim = 1/(a + 1) + b/(a + 1) \quad (2)$$

Boreas was platform independent and had the feature of scalability along with less time consumption and less comparison time.

Rochelle et al. [27] proposed a technique for detecting the semantic clone from code by using IOE behavior. In order to evaluate the clone form the file, the input and output was considered along with the variations in the heap state. The JAVA platform was most suitable platform for the implementation of the proposed work. This technique was comprised of four sub processes i.e. Abstraction, Filtering, Evaluation and Gathering. In first stage i.e. abstraction, AST trees was created in order to observe the method types and effects of the methods from the source code. In second stage i.e. filtering, two filters were concerned. The process of first filter was to return the functions from the source code which had the same return type and number of parameters or we can say it had the similar syntactical features and second filter detect the clone on the basis of semantic information from the code. After filtering, the evaluation phase was initialized by evaluating the dynamic nature of the functions. For this purpose test files were used. Then for the completion of the process, last phase i.e. gathering phase was done by executing the files. At last the clones from the source code were detected finally.

Thierry Lavoie et al. [28] proposed a levenshtein distance method to find out the clone from the code which was comprised of metric based and token based mechanism for clone detection. Metric trees and Manhattan separation were used for exact evaluation of levenshtein distance. Levenshtein separation was measure of the closeness between strings. It figures number of addition, cancellation and swapping of characters to change string s1 into s2. In the first step tokens were removed from source code with lexical analyzer. In the second step recurrence vectors were manufactured and unique id was given to every token. Id was given progressively and if any new thing founded then an accessible id was given to that token else the comparing vector of token was increased by one. Hashing Tables were utilized to store these vectors. In the next step, stride metric tree were manufactured by utilizing recurrence vectors. L1 metric i.e. Manhattan separation was utilized. Manhattan separation was picked as a result of the accompanying reasons given below:

a) L1 covers less space.
b) L1 is quick and high exactness.
c) In the fourth step Metric tree is worked with every one of the vectors.

Metric tree isolate the hunt space and increment the speed of addition, range questions and closest neighbors. The last stride was tree question step. In this progression, Manhattan separate between two metric trees was measured. In case if two trees had Manhattan separate not as much as limits then those were referred as clones.

Dandan Kong et al.[30] utilized the k-nearest algorithm for detecting the cloned code. This algorithm was comprised of the features of AST (Abstract Syntax Tree) and the advantages of k-nearest algorithm. By implementing the proposed work two templates from the code were considered as functionally equivalent only if the existences of permutation i.e. p1 and p2 between these templates are as:

$$OC1\left(p1\left(I\right)\right) = OC2\left(p2\left(I\right)\right) \quad (3)$$

Here OC refers to the output related to the set C1 and C2 coded templates.

Initially the code was passed from the lexical analyzer and syntax analysis for evaluating the control dependency information from the code by generating the abstract syntax trees and PDG. Then clustering algorithm was applied to obtain the functionally cohesive line of code. Then the output was observed and gathered into the related clusters.

Table2. Comparison table of Previous work on the basis of techniques used

Sr. No	Author	Technique Used	Scalability	Portability	Complexity
1.	Yong Yuan et al. [26](2012)	Token Based i.e. Lexical Approach	High	Language Independent	Medium
2.	Rochelle et al. [27] (2012)	On the basis of fragments behavior	High	Specifically developed for JAVA platform.	Less
3.	Theirry Lavoie et al. [28] (2012)	Hybrid(Token based and Metric Based)	Low	Only C Language	High
4.	Ira D. Baxter et al [29] (2012)	Tree Based (Syntactic Approach)	Low	Language Dependent and uses parser for transformation purpose.	High
5.	Dandang Kong et al. [30] (2012)	Hybrid(Tree and Graph based)	Low	Language Dependent parser	High

9. CONCLUSION

Clone in the source code can degrade the quality of the code written for the software since the code is referred as copied code. Cloning can lead to the enhancenments in the maintainence cost and occurrence of bugs to the software. Hence it is mandatory to remove the cloning from the source code. Each technique for clone detection has some advantages and limitations as discussed in this study. Some techniques can be used for specific clone detection in the source such as for semantic clone detection for evaluating the semantic related clone from the code.

10. FUTURE SCOPE

Number of techniques were evaluated for the clone detection in this study. Each of the techniques has some advantages and disadvantages. Based on these techniques, a hybrid technique based on the fine grained and coarse grain method can be proposed for the feature extration of code clone in future that can overcome the limitation of both of these technique and devlop an advance method with the advantages of these two methods combined andve ha higher efficiency for the detection of the code cloning.

11. REFERENCES

[1] Chanchal K. Roy, James R. Cordy and Rainer Koschke, “Comparison and evaluation of code clone detection techniques and tools: a qualitative approach”, Elsevier, Vol 74, Pp 470-495, 2009

[2] Martin White, Michele Tufano, Christopher Vendome and Denys Poshyvanyk,“Deep Learning code fragments for code clone detection”, ACM, International Conference of on Automated Software Engineering, Pp 87-98, 2016

[3] Benjamin Hummel, Elmar Juergens Lars Heinemann and Michael Conradit, “Index based code clone detection: Incremental, Distributed, Scalable”, IEEE, Pp 1-9, 2010

[4] Yoshiki Higo, Toshihiro Kamiya, Kusumoto and Katsuro Inoue “Methods and implementation for investigating code clones in a software system”, Elsevier, Vol 49, Issue 9-10, Pp 985-998, 2007

[5] Shruti Jadon , “Code clones detection using machine learning techniques: support vector machine”, IEEE, International Conference of computing, communication and automation, Pp 299-303, 2017

[6] Fang Hsiang Su, Jonathan Bell, and Gail Kaiser “Challenges and Behavioral code clone detection”, IEEE, International conference on Software analysis, evolution and reengineering, Pp 21-22, 2016

[7] Siim Larus and karl Kilgi, “Code Clone Detection using wavelets”, IEEE, Pp 8-14, 2015

[8] Kavitha Esther Rajakumari and T. Jebarajan, “A novel approach to effective detection and analysis of code clones”, IEEE, Pp 287-290, 2013

[9] Ritesh V.Patil, Lalit V. Patil, Sachin V. Shinde and S. D. Joshi, “Software code cloning detection and future scope development- latest short review”, IEEE, International conference on recent advances and innovation in engineering, Pp 1-4, 2014

[10] Mai Iwamoto, Shunsuke Oshima and Takuo Nakashima, “Token based code clone detection techniques in a student’s programming excercise”, IEEE, International conference on broadband wireless computing communication and application, Pp 650-655, 2012

[11] Toshihiro kamiya, Shinji Kusumoto and Katsuro Inouel, “A multi-Linguistic Token Based Code Clone Detection System for Large Scale Source code”, IEEE, Pp 1-37, 2002,

[12] Stephane Ducasse, “A Language Independent Approach for detecting Duplicate Code”, IEEE, Pp 1-10, 1995

[13] Simone Livieri, “Very ALrge Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder:DCCFinder”, IEEE, Pp 1-10,2007,

[14] Chanchal K. Roy, “An Empirical study of functions clones in open Source Software”, IEEE, Pp 1-10, 2008

[15] Elmar Juergens, “Do Code Clone Matters?”, IEEE, Pp 485-495, 2009

[16] Ginika Mahajan., “Implementing a 3-Way Approach of Clone Detection and Removal using PC Detector Tool”, in Proceedings of IEEE 2014, International Conference on Program Comprehension, pp.242-245,

[17] Nicolas Bettenburg , "An empirical study on inconsistent changes to code clones at the release level " ELSEVIER ,2010, Pp 1-17

[18] Nam H. Pham, "Complete and Accurate Clone Detection in Graph-based Models", IEEE, May 16-24, 2009, Pp286-276

[19] Elmar Juergens, "Do Code Clone Matters?", IEEE, Pp 485-495, 2009

[20] Kuldeep Kaur and Dr. Raman Maini, "A Comprehensive review of code clone detection Techniques", IJLTEMAS, Vol 4, Issue 12, Pp 43-47, 2015

[21] Abdullah Sheneamer and Jugal Kalita, "A Survey of Software Clone Detection Techniques", International Journal of Computer Applications, Vol 137, Issue 10, Pp 1-21, 2016

[22] Nils Gode, "Clone Removal: Fact or Fiction?", ACM, Proceedings of the 4th International Workshop on Software Clones , Pp 33-40, 2010

[23] Tahira Khatoon, Priyansha Singh and Shiksha Sukla "Abstract Sytax Tree Based Clone detection for java project", IOSR journal of Engineering, Vol 2, Issue 12. Pp 45-47, 2012

[24] Dhavleesh Rattan, rajesh Bhatia and Maninder Singh "Software Clone detection : a systematic review", Elsevier, Vol 55, Issue 7, Pp 1165-1199, 2013

[25] Michel Chilowicz , Etienne Duris and Gilles Roussel "Viewing functions as token sequence to highlight similarities in source code", Elsevier, Vol 78, Issue 10, Pp 1871-1891, 2013

[26] Yong Yuan and Yao Guo, "Boreas: An Accurate and Scalable Token-Based Approach to Code Clone Detection", IEEE, Pp 286-289, 2012

[27] Rochelle and Garry TR Leavens, "Semantic clone detection using method IOE-behavior", IEEE, Proceeding IWCS, Pp 80-81, 2012

[28] Theirry Lavoie and Ettore Merlo, "An accurate estimation of the Levenshtein distance using metric trees and Manhattan distance", IEEE, Proceedings in ISWC, Pp 7-15, 2012

[29] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna and Lorraine Bier, "Clone Detection using Abstract Syntax Tree", IEEE, Proceeding of ICSM'98, Pp 1-11, 2012

[30] Dandan Kong, Xiaohong Su ; Shitang Wu ; Tiantian Wang ; Peijun Ma "Detect functionally equivalent code fragments via k-nearest neighbor algorithm", IEEE, ICACI, Pp 94-98, 2012