Literature Survey of Software Clones

Vishwachi
Research Scholar,
AKGEC, Ghaziabad, India

Sonam Gupta
Assistant Professor,
Department of Computer Science
and Engineering, AKGEC,
Ghaziabad, India

ABSTRACT

Code cloning is the procedure where the developers reuse the code fragments implementing the paste option. They may or may not make the modification in the source code. The code thus developed after copying is known as clone. It is the synonym of duplicate. In the year 2002, Ira Baxter coined the term clones as the segments of code that are similar according to some definition of similarity. The similarity can be based on text, syntactic or semantic. Studies have revealed that almost 10-15% of the source code in large software are part of single or more clones[1]. Clones have adverse impact on the software maintenance, thus identification of clones is beneficial. In the past decade many tools have been developed to detect the clones but none was able to correctly identify all types of clones. In this paper the literature survey of all the clone detection techniques has been done. Along with this it also propose an approach which will use a combination of tree and token based approach in order to detect the code clones.

Keywords

Clones, textual comparison, LWH approach, token based approach, PDG approach, metric comparison, AST approach.

1. INTRODUCTION

A code clone is a section of code in the source files which is similar or identical to another code section[4]. The similarity can be based on text, syntax or sematic (behavioral). Code clone generally results from the practice of copy and paste followed by the developers. They result in increased maintenance efforts. Thus various tools have been developed to detect the code clones. Automated code clone techniques and tools utilize different similarity measures to find

clones in code[6]. Various techniques available for detecting the code clones are textual comparison, metric based approach, token based approach, AST based approach, PDG approach, LWH approach, CRD approach. In this paper we will briefly describe the clones and the measures to identify them. The consequences of cloning are that cloning of code generally increases the maintenance effort. If the

code is redundant then changes must be made consistently multiple times**.** This effort could have completely avoided if the code would have been implemented only once in a function. Code cloning is a purposeful implementation strategy which may make sense under certain circumstances [3]. Generic solutions can become overly complicated.

The paper is organized as: section II contains types of clones, section III describes the procedure of clone detection, section IV gives the detailed techniques of clone detection along with the limitations and advantages of each. And finally section V gives the proposed approach and section VI includes the conclusion.

2. TYPES OF CLONES

Clones are generally divided into four main categories:

Type 1: program fragments which are identical copies of each other expect for the whitespaces and comments variations.

Example:

```
int i=1;
if(i<=5)
printf (“
Continue…..”);
else
printf( “skip..”);
i++;
// fragment 1
```

```
int i=1; /*initializing the
value of i */
if ( i<=5)
printf( “Continue…..”);
else
printf( “skip..”);
i++;
// fragment 2
```

Type 2: Program fragments which are syntactically identical copies; except some changes in variables names, data type, identifier name, etc.

Example:

```
int i=1;  /* initializing
value of i */
if(i<=10)
printf(“ Hello…..”);
else
printf(“ Bye…..”);
i++;
//fragment 1
```

```
float k=1;
if(k<=10)
printf( “ Hello…..”);
else
printf(“ Bye…..”);
k++;

//fragment 2
```

Type 3: is a copied fragment with further modifications. Statements can be changed ,added or removed in addition to variations in identifiers, literals, types, layout and comments.

Example:

```
int main()
{
int a=1,b=5,sum=0;
sum=sum+a;
printf("%d", sum);
}
//fragment 1
```

```
int main()
{
if(sum<=5)
{
s=a+sum;
sum++;
}
printf ("%d", s);
}
//fragment 2
```

Type 4:Two or more code fragments that perform the same computation, but implemented through different syntactic variations.

Example:

```
void sum()
{
int x=1;
int y=x=5;
return y;
}
//fragment 1
```

```
intfunc()
{
int m=5;
return ++m;
}
//fragment 2
```

3. CLONE DETECTION PROCESS

The detection of clones has now become one of the most important part of the development process as the clones create trouble later in the maintenance phase. The process of clone detection can be sub-divided into six phases. They are:

1. Preprocessing

The detection of code cloning starts with the partitioning of the source code. This phase is mainly responsible for:

- Removal of all the comments, white spaces etc.
- Determine source units: the source code obtained after the removal of white spaces and comments is then partitioned into a set of disjoint pieces known as source units which are the largest source sections suspected to be involved in direct clone relations with each other[4].
- Determining the granularity: source units may further be partitioned into smaller units say into lines or tokens for a comparison purpose.

2. Transformation

After preprocessing step, the source code of the comparison units is transformed to a proper intermediate format for comparison [4]. Transformation includes extraction and normalization. Extraction may involve tokenization, parsing, control and data flow analysis depending on the approach we are following in the detection process. Normalization is an optional step.

3. Match Detection

The comparison units take transformed code as input and compares the transformed comparison units to each other to discover matches [4].

4. Formatting

Here the source coordinates of each clone pair obtained in the comparison phase are mapped to their positions in the original source files.

5. Filtering

Here the clones are manually analyzed, filtered and ranked or they may be fed under automated heuristics.

6. Aggregation

Clones may be aggregated to clone classes with the aim to diminish the measure of data or gather overview statics.

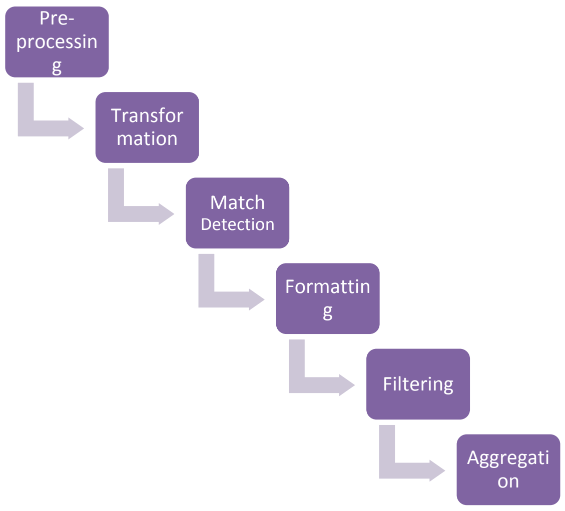

Fig 1. Clone detection procedure

4. APPROACHES OF CLONE DETECTION

4.1 Textual Comparison

This approach compares whole lines to each other textually. The targeted source code is considered to be the sequence of the strings or lines.in order to find the match/clone the two code parts are compare to each other. If the two parts of the code found out to be similar then they are considered as clones. The approach followed in textual comparison is light weighted and are able to detect the clones accurately with higher recall values, where recall refers to the overall percentage of clone exist in the source code that have been detected by the clone detector [1]. The work done in this technique has been summarized in table 1.

4.2 Token Based Approach

In token based clone detection techniques, firstly, tokens are extracted from the source code by lexical analysis. Then sequence is formed from some set of tokens which are then compared in order to find the clone. They are fast with high recall values. The most popular token-based tool named CCFinder was developed by T.Kamiya and S.Kusumoto [19] in the year 2002. Since then, it is a popular tool among researchers and has been used widely for code clone analysi sand management. Researchers are working to enhance the output of CCFinder for example Basit et al. [20] used CCFinder to study the patterns of clones in a standard template library. They increased the threshold of CCFinder to detect the smaller clones too. Another tool based on token approach is CP-Miner which uses frequent item set mining in order to detect the bugs in the softwares produced due to clones. Yamashina et al. [42] designed a tool called CCFinderX. The work done in this technique has been summarized in table 2.

Table 1. Work done on text based approach

Sno.	**Author**	**Tool**	**Year**	**Advantage**	**Disadvantage**
1.	Wettel et al. [17]	Dude	2005	Can detect duplication chains consisting of number of smaller size exact clones.	Can not be applied to large systems.
2.	C.K. Roy et al. [30,31]	NICAD	2008-2009	Can detect type 3 clones very effectively as compared to other text based tools.	Not exactly text-based but rather hybrid as it exploits the benefits of tree-based structural analysis.
3.	S. Lee et al. [18]	SDD	2005	Capable of detecting clones in large sized systems.	Its accuracy is not high.
4.	Baker et al. [36,37]	Dup	1992-1999	Can detect clone even if the names of variables are different.	Uses large search space as hashing is to be applied. it cannot detect clones if source code is written in different styles.
5.	Cordy et al. [39]		2004	Capable of detecting near-miss clones.	Precision value is not high.
6.	Ducasse et al.			The tool is language independent	Not able to detect meaningful clones

4.3 Metric Based Approach

In this technique, we gather different metrics for code fragments and compare these metric vectors instead of comparing the code directly. In this approach metric values for different methods are calculated to extract the potential clone pairs. The metrics used may involve number of lines, number of arguments, number of function calls etc. The two methods whose metrics comes out to be similar are considered as clone pairs. The major advantage of this technique is that it can detect both the syntactically and semantically similar clones. The work done in this technique has been summarized in table 3.

Table 2. Work done on token based approach

Sno.	**Author**	**Tool**	**Year**	**Advantage**	**Disadvantage**
1.	T.Kamiya et al. [19]	CCFinder	2002	Till date the most popular tool used for clone analysis and management.	It is unable to detect smaller clones.
2.	Basit et al. [20]		2005	Enhanced CCFinder by increasing its threshold to detect smaller clones too.	Unable to find out the semantic similarity between the codes.
3.	Z Li et al. [44]	CP-Miner	2006	Can detect bugs in software induced due to cloning.	Precision is not very high.
4.	Yamashina et al. [42]	CCFinderX	2009	Tokens are fed as input to suffix array which results in fast retrieval.	Needs improvement in ranking algorithm.
5.	Sasaki et al.	FCFinder	2010	Used hashing	It does not accept source files written in two or more programming language.

Table 3. Work done on Metric based approach

Sno.	Author	Year	Advantage	Disadvantage
1.	Mayrand et al. [21]	1996	Was one of the first approach to compare metrics obtained from AST of source code.	Not able to identify segments which are based on copy-paste operation.
2.	Kontogiannis et al. [22]	2004	Applies dynamic programming on the lines of source code by using minimum edit distance between them. Thus, it is able to detect the similarity more precisely.	Not being able to find the exact clones. It can only find out the similarities between the codes.
3.	Perumal et al. [43]	2010	Used fingerprint technique to detect the clones.	The technique used is quite costly.
4.	Li and Sun [44]	2010	The technique used in this tool is scalable as well as accurate.	It is yet to be verified for different systems.
5.	Lovoie et al. [45]	2010	Technique used is based on graphics processing unit (GPU) which results in increased performance.	

4.4 Program Dependency Graph (PDG) Approach

PDG is a semantic (behavioral) based approach. PDG considers the semantic information which is encoded in the form of a dependency graph that captures the data flow and control information. Clones may be identified as isomorphic sub graphs in a PDG [3]. R. Komondoor et al. [24] developed a tool based on PDG approach which uses program slicing to find out the isomorphic sub-graphs. Its main feature is that it helps in detecting the non-contigous clones. Another tool Scorpio was developed by Higo and Kusumoto [25] in the year 2011 which applies two-way slicing to detect clones. It was developed with the aim to address the problem of slow detection of contiguous clones which the existing systems faced. Krinke [35] used PDG as an iterative approach for finding maximal similar sub-graph but it suffers from the shortcoming that it was not able to give a formula that can be used on any type of system to find the clone[35]. All the researchers using PDG technique came to the conclusion that although PDG-based techniques can find non-contiguous clones but it cannot be applied to large systems[35]. The major disadvantage of this approach is that sub graph comparison is quite costly in this approach. The algorithms which uses this technique returns the approximate results. The tabular representation of the work done so far using this technique is given in table 4:

Table 4. Work done on PDG based approach

Sno.	Author	Year	Advantage	Disadvantage
1.	Horwitz et al.	1990	Can identify syntactic and semantic difference between two versions of program.	Cannot be applied to large systems.
2.	Krinke et al. [35]	2001	Finds maxiaml similar sub-graphs	Not able to give a single formula which can be used on any type of system for finding clone.
3.	R. Komondoor et al. [24]	2003	Finds isomorphic sub-graphs. Helps in detecting non-contigous clones.	Limited only to smaller sized systems.
4.	Higo et al. [25]	2011	Based on no. of PDG specialization for Java language and heuristics	

4.5 Abstract Syntax Tree (AST) Approach

The most commonly used representations in order to transform the source code into tree structure is Abstract Syntax tree and parse trees. It can be used to find out the syntactic differences between the two parts of the same source code. This technique is generally based on grammar and a parse tree is generated for both the parts of the code. Detection of the clone is applied synchronously to both the trees and it is based on the LCS(longest common subsequence). If the subsequence of the two parts comes out to be similar then they will be considered as clones. The work done in this technique has been summarized in table 5.

Table 5. Work done on AST based approach

Sno.	Author	Year	Tool	Advantage	Disadvantage
1.	I.D.Baxter et al. [26]	1998	Clone DR	Can detect as well as near miss clones.	Suffers from large execution time.
2.	Jiang et al. [46]	2007	Deckard	Capable of finding the behavioral similarity.	The approach used is heavy-weighted.
3.	Falke et al.[29]	2008		Used syntax tree. Has advantage of precision of syntax tree and high speed of syntax tree.	Takes longer time to traverse the tree.
4.	Ekoko et al.[49]	2008	Clone Tracker	Detect the clones in Java codes	Results in large number of false positives.
5.	W.S.Evans et al. [48]	2009	Asta	Works on structural abstraction of arbitary sub-trees of AST.	
6.	T.T.Nguyen et al. [23,41]	2009	ClemanX	Can detect clones in aassembly code.	

4.6 Hybrid Approach
As the name implies it is a hybrid technique so it will combine two or more of the above techniques. LWH (light weight hybrid) combines the textual comparison and metrics based approach in order to detect the method-level syntactic and as well as semantic clones. The tool developed using this technique accepts the source code and separates the functions/methods present in it. Also, it forms the template of each of the method present. After this, the code metrics is computed for each method and are stored in a database. The methods whose metrics values comes out to be nearly similar are subjected to textual comparison to detect the actual clone pairs. Egambaram Kodhai et al. [1] in the year 2014 developed a tool named CloneManager which used LWH approach. The tool is able to detect all four types of the clones with high precision. It first converts the source code into templates and then apply the metrics approach to find the similarities between different parts of the source code. Maeda[50] in the year 2009 introduced a technique based on PALEX source code representation.it is language independent and uses a suffix tree for comparison. Chilowicz et al. [51] used suffix array and metrics. The technique starts with collecting the tokens using lexical analysis. Basit et al. [52] developed a tool named Clone Miner in the year 2009 which used frequent item set mining and works on the output of a token based clone detection tool named RTF. Its disadvantage is that it doesn't implies the refactoring method. The major advantage of this approach is that it is light weighted so the time it takes to detect the clones is quite low as compared to other techniques. The accuracy of the tool developed using this technique comes out to be 88-100 % which is considerably high.

5. PROPOSED APPROACH
Till now, we have seen the advantages and disadvantages of all the well known approaches of locating the clones in source code. There is not a single approach which is capable of detecting all four kinds of clones precisely with 100% efficiency. The text based approach can detect the clones of only 2 types (in some cases upto type 4). So, here we propose a hybrid approach including the tree and token based approach. We will construct an AST of the given source code and will traverse it using DFS technique. After that, we will store the leaf nodes of the tree as tokens and the tokens of the same kind will be stored in the list. Then the Levenshetin distance between them will be find out. The two list of tokens among whom the levenshetin distance comes out to be minimum than a particular threshold can be considered as clones.

6. CONCLUSION
In this paper, a review about the clones and different techniques to detect them is shown. Clone are harmful as they increase the maintenance efforts. So, it is better to detect them beforehand in order to avoid any kind of problems in future. Also, the refactoring procedure is not easy because of the cost and risk associated with refactoring. Therefore, one of the above mentioned approach may be used at the time of development of the source code before handing over the product to the client so that in maintenance phase the product doesn't encounter much difficulty. The studies shows that the most advantageous method of detecting clones is the combination of metrics and textual approach as they results in light weight tool development occupying less space and resulting in a small execution time as compared to other techniques.

7. REFERENCES
[1] Kodhai, Egambaram, and SelvaduraiKanmani. "Method-level code clone detection through LWH (Light Weight Hybrid) approach." Journal of Software Engineering Research and Development 2.1 (2014): 1.

[2] Sonam Gupta, Dr. P.C Gupta, Clones: A Survey, International Journal of Computer Science and Technology 2012

[3] Koschke, Rainer. "Survey of research on software clones." Dagstuhl Seminar Proceedings.SchlossDagstuhl-Leibniz-ZentrumfürInformatik, 2007.

[4] Singh, Gurvinder, and Jahid Ali. "A Novel Composite Approach for Software Clone Detection." International Journal of Computer Applications 126.7 (2015).

[5] Singh, Manu, and Vidushi Sharma. "Detection of File Level Clone for High Level Cloning." Procedia Computer Science 57 (2015): 915-922.

[6] Tairas, Robert, and Jeff Gray. "Increasing clone maintenance support by unifying clone detection and refactoring activities." Information and Software Technology 54.12 (2012): 1297-1307.

[7] Rattan, Dhavleesh, Rajesh Bhatia, and Maninder Singh. "Software clone detection: A systematic review." Information and Software Technology 55.7 (2013): 1165-1199.

[8] Basit, Hamid Abdul, and Stan Jarzabek. "A data mining approach for detecting higher-level clones in software." IEEE Transactions on Software engineering 35.4 (2009): 497-514.

[9] Al-Omari, Farouq, et al. "Detecting clones across microsoft. net programming languages." 2012 19th Working Conference on Reverse Engineering.IEEE, 2012.

[10] Gupta, Sonam, and P. C. Gupta. "Algorithm to Detect Non-Contiguous Clones with High Precision." International Journal of Innovations in Engineering and Technology, Vol 5 Issue 1, February 2015.

[11] Gupta, Sonam, and P. C. Gupta. "A Novel Approach to Detect Duplicate Code Blocks to Reduce Maintenance Effort." International Journal of Advanced Computer Science & Applications 1.7 (2016): 311-314.

[12] Gupta, Sonam, and P. C. Gupta. "International Journal of Software and Web Sciences (IJSWS) www.iasir. net." International Journal of Software and Web Sciences (2015): 65.

[13] Chatterji, Debarshi, Jeffrey C. Carver, and Nicholas A. Kraft. "Code clones and developer behavior: results of two surveys of the clone research community." Empirical Software Engineering (2015): 1-33.

[14] Kanagalakshmi, K., and R. Suguna. "Software Refactoring Technique for Code Clone Detection of Static and Dynamic Website."

[15] Kapser, Cory J., and Michael W. Godfrey. "Supporting the analysis of clones in software systems." Journal of Software Maintenance and Evolution: Research and Practice 18.2 (2006): 61-82.

[16] Kapser, Cory, and Michael W. Godfrey. "" Cloning considered harmful" considered harmful." 2006 13th Working Conference on Reverse Engineering.IEEE, 2006.

[17] Wettel, Richard, and RaduMarinescu. "Archeology of code duplication: Recovering duplication chains from small duplication fragments." Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC'05). IEEE, 2005.

[18] Lee, Seunghak, and IryoungJeong. "SDD: high performance code clone detection system for large scale source code." Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications.ACM, 2005.

[19] Kamiya, Toshihiro, Shinji Kusumoto, and Katsuro Inoue. "CCFinder: a multilinguistic token-based code clone detection system for large scale source code." IEEE Transactions on Software Engineering 28.7 (2002): 654-670.

[20] Basit, Hamid Abdul, Damith C. Rajapakse, and Stan Jarzabek. "Beyond templates: a study of clones in the STL and some general implications."Proceedings of the 27th international conference on Software engineering.ACM, 2005.

[21] Mayrand, Jean, Claude Leblanc, and Ettore M. Merlo. "Experiment on the automatic detection of function clones in a software system using metrics."Software Maintenance 1996, Proceedings., International Conference on. IEEE, 1996.

[22] Patenaude, J-F., et al. "Extending software quality assessment techniques to java systems." Program Comprehension, 1999.Proceedings.Seventh International Workshop on.IEEE, 1999.

[23] Nguyen, Tung Thanh, et al. "Scalable and incremental clone detection for evolving software." Software Maintenance, 2009.ICSM 2009.IEEE International Conference on.IEEE, 2009.

[24] Komondoor, Raghavan, and Susan Horwitz. "Using slicing to identify duplication in source code." International Static Analysis Symposium.Springer Berlin Heidelberg, 2001.

[25] Higo, Yoshiki, and Shinji Kusumoto. "Code clone detection on specialized PDGs with heuristics." Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on. IEEE, 2011.

[26] Baxter, Ira D., et al. "Clone detection using abstract syntax trees." Software Maintenance, 1998.Proceedings., International Conference on. IEEE, 1998.

[27] Tool SimScanhttp://www.blue-edge.bg/download.html

[28] Project Bauhaus http://www.bauhuas-struggart.de

[29] Koschke, Rainer, RaimarFalke, and Pierre Frenzel. "Clone detection using abstract syntax suffix trees." 2006 13th Working Conference on Reverse Engineering.IEEE, 2006.

[30] Roy, Chanchal K., and James R. Cordy. "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization."Program Comprehension,

2008.ICPC 2008.The 16th IEEE International Conference on.IEEE, 2008.

[31] Roy, Chanchal K. "Detection and analysis of near-miss software clones."Software Maintenance, 2009.ICSM 2009.IEEE International Conference on.IEEE, 2009.

[32] Barbour, Liliane, Hao Yuan, and Ying Zou. "A technique for just-in-time clone detection in large scale systems." Program Comprehension (ICPC), 2010 IEEE 18th International Conference on.IEEE, 2010.

[33] Roy, Chanchal K., and James R. Cordy. "Are scripting languages really different?." Proceedings of the 4th International Workshop on Software Clones. ACM, 2010.

[34] Martin, Douglas, and James R. Cordy. "Analyzing web service similarity using contextual clones." Proceedings of the 5th International Workshop on Software Clones. ACM, 2011.

[35] Gupta, Sonam, and P. C. Gupta. "Literature survey of clone detection techniques." International Journal of Computer Applications 99.3 (2014): 41-44.

[36] Brenda S. Baker. A Program for Identifying Duplicated Code. In Proceedings of Computing Science and Statistics: 24th Symposium on the Interface, Vol. 24:4957, March 1992.

[37] Brenda S. Baker. Parameterized diff.In Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA'99), pp. 854-855, Baltimore, Maryland, USA, January 1999.

[38] Brenda S. Baker. On Finding Duplication in Strings and Software.Journal of Algorithms, 1993.

[39] Cordy, James R., Thomas R. Dean, and Nikita Synytskyy. "Practical language-independent detection of near-miss clones." Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research. IBM Press, 2004.

[40] Monden, Akito, et al. "Software quality analysis by code clones in industrial legacy software." Software Metrics, 2002.Proceedings.Eighth IEEE Symposium on.IEEE, 2002.

[41] Nguyen, Tung Thanh, et al. "Scalable and incremental clone detection for evolving software." Software Maintenance, 2009.ICSM 2009.IEEE International Conference on.IEEE, 2009.

[42] Kawaguchi, Shinji, et al. "Shinobi: A tool for automatic code clone detection in the ide." 2009 16th Working Conference on Reverse Engineering.IEEE, 2009.

[43] Perumal, A., Kanmani, S., &Kodhai, E. (2010, September). Extracting the similarity in detected software clones using metrics. In Computer and Communication Technology (ICCCT), 2010 International Conference on (pp. 575-579). IEEE.

[44] Li, Z. O., & Sun, J. (2010, April). A metric space based software clone detection approach. In Information Management and Engineering (ICIME), 2010 The 2nd IEEE International Conference on (pp. 393-397). IEEE.

[45] Lavoie, T., Eilers-Smith, M., & Merlo, E. (2010, May). Challenging cloning related problems with GPU-based algorithms.In Proceedings of the 4th International Workshop on Software Clones (pp. 25-32).ACM.

[46] Jiang, L., Misherghi, G., Su, Z., &Glondu, S. (2007, May). Deckard: Scalable and accurate tree-based detection of code clones. In Proceedings of the 29th international conference on Software Engineering (pp. 96-105).IEEE Computer Society.

[47] Tairas, R., & Gray, J. (2010, March). Sub-clone refactoring in open source software artifacts. In Proceedings of the 2010 ACM Symposium on Applied Computing (pp. 2373-2374).ACM.

[48] Evans, W. S., Fraser, C. W., & Ma, F. (2009). Clone detection via structural abstraction. Software Quality Journal, 17(4), 309-330.

[49] Duala-Ekoko, E., &Robillard, M. P. (2008, May). Clonetracker: tool support for code clone management. In Proceedings of the 30th international conference on Software engineering (pp. 843-846).ACM.

[50] Maeda, K. (2009). Syntax sensitive and language independent detection of code clones. World Academy of Science, Engineering and Technology, 60, 350-354.

[51] Chilowicz, M., Duris, É.,&Roussel, G. (2009). Finding similarities in source code through factorization. Electronic Notes in Theoretical Computer Science, 238(5), 47-62.