International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-4, February 2020

A Qualitative Interpretation of Code Clone Detection Techniques

Amandeep Kaur, Sandeep Sharma

Abstract *Clone is the software code snippets that are similar to each other with little modifications. There is a 10-20 percent clone mostly present in the software. Many techniques are developed for detection. With the code clone detection, the software developer gets an idea of removing, refactoring the clone. Code clone has both advantages and disadvantages in the particular software. In this paper, we explore the types of code clones, advantages, and disadvantages, the reason for cloning. Typically, this paper describes various techniques by using several parameters. Lastly, we discuss gaps in the research.*

Keywords: *Code Clone Detection, Data mining, Tokens, Code Clone, Abstract Syntax Trees.*

I. INTRODUCTION

Code snippets that are copied and paste in the software code with or without change results into code clone. Different authors provide different definitions of code clones. Various studies provide information about the percentage of duplication in the source code. Commonly, there is a 10-20 percent cloning present in the software code[3][5][22].

Due to advancements in technology, the companies spend more cost on the maintenance of software. For the maintenance of software, code clone is detected and should be removed or refactored according to the requirements. Because if code fragment that contains bug is copied or duplicated then each it is clone also contains that bug and it is difficult to discover the bug in the large software that contains thousands or millions of LOC.

Roy[25] and Koschke[16] presents a comprehensive review of the clone found in the software. In the remaining paper, Section 2 presents the background. Section 3 detailed the various techniques of clone detection. Section 4 shows the result analyses. In the last, we conclude this paper.

II. BACKGROUND

In this, we discuss the classification of clones, advantages, and disadvantages, why clone occurs in the software.

A. Classification of Clones

Broadly, the code clone divided into four categories. Fig. 1 shows the classification of clones.

- **Type-1:-** Code snippet similar to other snippets with the only changes in whitespaces, comments leads to this clone. Type-1 can be detected easily with the text-based and token-based technique.
- **Type-2:-** Type-2 clone occurs due to alteration in name of the identifier, literals, and variables, keywords. They can be detected with the token and metric-based technique.
- **Type-3:-** Results from addition, deletion of lines of codes. They can be detected with a tree-based and graph-based approach.
- **Type-4:-** A Graph-based and hybrid approach can be used to detect a type 4 clone. This clone occurs when two code snippets have similar functionality but the difference in their structure.

B. Advantages of code clone

- **A better understanding of the program:** Code clone helps to better understand the code fragment as a similar code appears many times in the program [24].
- **Code reuse:** The code fragment is difficult to design but similar functionality is available in another code fragment then the programmer can easily reuse that.
- **Bug identification:** Bug can be identified in the clone which helps to correct an error in the original code and vice versa.
- **Plagiarism detection:** Plagiarism can be easily detected through the code cloning techniques. Mostly the plagiarism did at the time of the assignment or project-related work [3][12].
- **Copy infringement:** Copy infringement can be performed through the code cloning technique as the many software companies use the code of another company's software without their permission [31].
- **Clone removal/ refactoring:** Code cloning helps to determine which clone should be removed or refactored according to requirements.

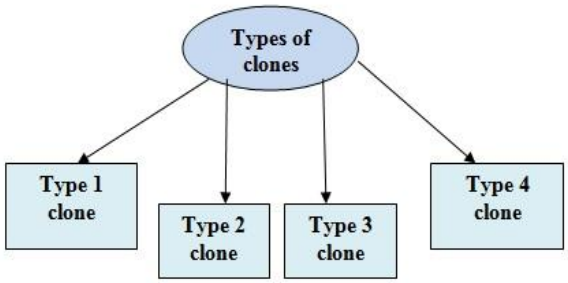

Fig 1. Classification of clones

C. Disadvantages of code clone

- **Bug propagation:** The bug occurs in the particular code fragment then all its clone also contains that error which is known as bug propagation [20].

Revised Manuscript Received on February 28, 2020.
***** Correspondence Author
Amandeep Kaur*, Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar, India.
Sandeep Sharma, Head Professor, Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar, India.

- **Maintenance cost increases:** Various studies show that about 90 percent of software development cost spends on software maintenance [15].
- **Bad quality design:** The software that consists of too much clone is considered to be bad quality software as it is difficult to understand the program [13].
- **Difficulties in Modifications or Improvements:** Modifications are difficult to make as changes in the original code requires the changes in its entire related clone [11].

D. Why clone occurs

- **Cloning by accident:** Sometimes the programmer writes the codes that are similar to another code fragment as no other solution for the given problem is available [25].
- **Time/resource constraint:** Programmer copy, paste or make some modification in the original code if the resources or time available with him is limited [14].
- **Language constraints:** Sometimes the programming language has limited features, therefore, programmers use the code with or without modifications [14].
- **Programmer's constraints:** The Programmer does not have the knowledge of problem domain sometimes that leads to the use of the already existing code in the program [25].
- **Risk of designing the new code:** Cordy[7] says that in financial software, updations or some modifications are made in the existing code as the risk associated with it related to money. So, financial products are not changed very much frequently.

III. TECHNIQUES FOR DETECTION OF CODE CLONE

Many researchers propose various detection techniques.
These techniques are differentiated according to source code representation, comparison method, etc. Fig. 2 shows the classification of techniques.

A. Textual techniques

Text-based clone detection can easily identify the Type-1 clone. Text-based clone detection technique has a very less false positive clone. Johnson [11] proposes the tool which is applied to the source code of length 40 megabytes. Roy and Cordy [26] presents a technique that involves three phases. First, using parsing to remove noise, make the standard format, and break the program statements into parts. Second, using island grammar and parsing to effectively find the potential clone. Third, code normalizations are provided using transformation rules to filter out uninteresting part of a potential clone. Lee and Jeong [19] describe the algorithm that detects exact copy and bit/pieces of similar code.

B. Lexical techniques

In these techniques, tokens are generated from code using lexer. The lexical technique has more false-positive then textual. Li et al. [20] developed a tool named CP-Miner. It represented statements into number and similar statements have similar numbers. The sequence of numbers is stored into a database and frequent mining is performed to find a similar sequence of numbers. The tool focuses on the analysis of code by size, by granularity and by a degree of modifications. The tool finds the clone in two popular operating systems.

Basit et al. [4] describe a technique that finds repeated tokens by implementing the concept of the suffix array. The token-based technique involves the least code transformations. Yuan and Guo [33] depict the approach that using cosine similarity to find the clone. Experiments performed on the JDK7 and Linux kernel.

C. Syntactic techniques

These techniques deal with the program's syntax. It incorporates a tree-based and metric-based approach. Table 1 shows the comparative portrait of these techniques.

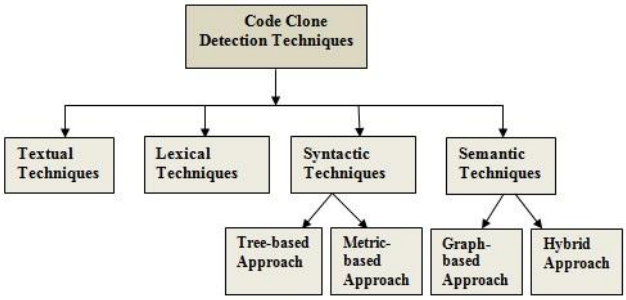

Fig 2. Classification of techniques of code clone detection

- **Metric-based approach**

The metric-based technique detects the syntactic clone. Mayrand et al. [22] describe the approach that includes functional metrics for detection. These metrics are incorporate into points for comparison to identify the function clone in the source code. As the level increases the false positive rate also increases. Patenaude et al. [23] describe the technique that considers source code object metrics and interface metrics for the identification of JAVA clone. The experiment performed on the 500KLOC of java source code. Lanubile and Mallardo [18] describe the method for the detection of clone present in web applications. The homonym function refers to potential clone function. The homonym function is represented together by three measures LOC, ELOC, CLOC. Abd-El-Hafiz [1] presents the method that includes the four steps. Firstly, the preprocessing of code performed. In the next step, the function and metric are extracted. In the third step, the fractal clustering algorithm is applied to cluster the functions and in the last, the clone classes are extracted.

- **Tree-based approach**

In this technique, the Abstract syntax tree generated from the code snippet. Yang [32] presents the algorithm. The text-based technique doesn't identify what kind of differences because some program has rigid syntactic structure.
The algorithm is based on a dynamic programming scheme. The pretty-printer is used to highlight the difference between the two programs. Baxter et al. [5] describe the approach that uses three algorithms to find clone. The first approach finds the subtree clone. The next algorithm finds the sequenced clone.
The last algorithm detects the near-miss clone. The method also enables the removal of detected clones.

This technique applied to the C language software system to detect the clones. Wahler et al. [30] describe the clone detection mechanism that detects type1, 2 clones. In this, the frequent itemset finding technique is used. XML representations exist for Java, Prolog, and C++. Koschke et al. [17] describe the technique that finds the syntactic clones in linear complexity This technique is compared with other techniques using Bellon benchmark. The suffix tree detection applied to find the clone pairs. Sager et al. [27] describe the method for clone detection in JAVA language. Tree similarity algorithm which includes the tree edit distance, Top-down maximum common subtree isomorphism, bottom-up maximum common subtree isomorphism is applied to find the common subtree. Bulychev and Minea [6] describe the approach that implemented the anti-unification method. Duplicated code is identified by setting the minimum distance threshold. The tool currently supports Java and Python languages.

Table I. Comparative Portrait of techniques of code clone detection

Properties / Author	Tool	Code Representation	Comparison Granularity	Comparison technique	Types of clone detected	The language used for an experiment
Johnson[11]	Not available	Substring	Chunks of text	Karp and robin fingerprinting based string matching	Type 1, 3	C
Roy and Cordy [26]	NICAD	Sequence of items	Text	LCS(longest common subsequence)	Type 1, 2, 3	C
Lee and Jeong [19]	SDD	Text	Not fixed	n-Neighbor distance	Type1, 3	JAVA
Li et al. [20]	CPMiner	Block represents a sequence of numbers	Token sequence	Frequent subsequence mining	Type1,2,3	C, C++
Basit et al. [4]	RTF	Token sequence	Token	Suffix array algorithm	Type1,2,3	C
Yuan and Guo [33]	Boreas	Variable	Count vector	Cosine similarity	Type1,2,3	C, C++, JAVA
Yang [32]	Not available	Parse tree	AST node as Token	Tree matching using dynamic programming	Visualization only	C
Baxter et al. [5]	CloneDR	AST	AST node	Hash value based matching	Type 1, 2, 3	C, Java, C++
Wahler et al. [30]	Not available	An XML representation of AST	Statement	Frequent itemset mining	Type 1, 2,	Java, Prolog, C++`
Evans et al. [8]	Not available	An XML representation of AST	AST node	Graph-theoretic approach	Type 1, 2, 3	Java, C++
Koschke et al. [17]	Not available	AST	Token(AST node)	Suffix tree detection	Type 1, 2	C, Java
Sager et al. [27]	Coogle	FAMIX tree	FAMIX node	Tree comparison algorithm	Type 1, 2, 3	Java
Bulychev and Minea [6]	Clone digger	Anti-Unifier tree	Subtree	Suffix tree detection	Type 1, 2, 3	Python, Java
Mayrand et al. [22]	CLAN	Metrics	Metrics for function	4 point of comparison	Type 1,2,3	C
Patenaude et al. [23]	Not available	Metrics	Metrics for method	Distinct Comparison	Type 1, 2,3	Java
Lanubile and Mallardo [18]	Not available	Homonym function	Function	Visual inspection	Type1,2,3	Web language
Abd-El-Hafiz [1]	Not available	Metrics for function	Function	Fractal clustering	Type1,2,3	C
Komondoor and Horwitz [15]	Not available	PDGs	PGD nodes	Isomorphic PDG subgraph matching by Backward slicing	Type 3, 4	C

Liu et al. [21]	GPLAG	PDGs	PGD nodes	Isomorphic PDG subgraph matching algorithm	Type 1,2, 3	C, C++, Java
Gabel et al. [9]	Not available	Vector characteristics of the subtree	Vector	Tree-based Locality-sensitive hashing detection	Type 3, 4	C, C++
Higo et al. [10]	Not available	PDGs	PGD edges	Code clone detection module	Type 3, 4	Java
Agrawal and Yadav[2]	Not available	Tokens	Tokens	Line by line comparison	Type 1, 2, 3	C
Sheneamer and Kalita[28]	Not available	Feature Vector	Method	Classification algorithms	Type 3, 4	Java
Sheneamer [29]	CCDLC	Feature Vector	Method	Clustering and Deep learning classifier	Type 3, 4	Java

D. Semantic techniques

Semantic clone occurs when the syntax of two fragments is different but they perform the same functionality. These techniques find the semantic clones. The semantic approach mostly detects type 3 clone. It comprises of graph-based approach and a hybrid approach.

- **Graph-based approach**

Program dependency graphs of code snippets are made in this approach. Komondoor and Horwitz [15] describe the tool that identifies clones using backward slicing. After that the potential clone is removed that is not the actual clone and then the actual clone is combined to form the larger groups. Liu et al. [21] present the tool that detects plagiarism through a program dependency graph. Many plagiarism tools are available for academic use. GPLAG makes use of the combination of lossy filter and lossless filter which properly prunes the search space. Gabel et al. [9] describe the method that focuses on the simple representation of a similar graph to a similar tree by mapping the program dependency graph to related structured syntax. The vectors are generated from the abstract syntax trees which are clustered according to the similarity to generate clone pair, clone classes. Higo et al. [10] describe the approach that comprises the analysis and detection phase for finding the clone pair.

- **Hybrid approach**

The hybrid techniques are developed to encounter the limitation of the existing techniques. The hybrid technique includes both semantic and syntactic properties. Agrawal and Yadav[2] describe the approach in which the Textual and token technique are combined to identify the clone of type 1, 2 and 3. Sheneamer [29] describes the method that makes use of abstract syntax tree and program dependency graph for detection of the clone.

IV. RESULT ANALYSES

In this section, we are analyzing these techniques in the form of the gap in the technique or tools that are discussed above.

Almost every technique or tool identifies Type 1, 2 clones as they are easy to identify. Type 4 or a functional clone is tough to identify. The efficiency, scalability, time and space requirement of the clone detection tool and technique can be improved by designing the detection algorithm by employing other domains. Table 2 presents gaps in the research of the papers discussed above.

Table II. Gaps in research

Sr No.	Reference No.	Gaps in Research
1	[11]	It does not compute the complexity of the technique.
2	[26]	NICAD is experimented to find the clones in small and medium-size software. It can detect a clone in only one language.
3	[19]	It cannot compute the recall and precision of the algorithm.
4	[20]	The CP-Miner could not detect the bug related to semantic information.
5	[33]	The complexity of this technique is not computed.
6	[32]	The technique ignores the semantic information of the program. The pair of non-terminal should not be considered for comparison. This technique requires parser and pretty-printer for language independence.
7	[5]	This technique does not focus on the removal of the clone
8	[30]	The clone detection mechanism does not find the Type-3 clone.
9	[8]	The complexity of this algorithm is not computed.

10	[27]	Bottom-up maximum common subtree isomorphism detection is not good for similarity purposes. Minor modifications in the method result in the change at the bottom level of the tree.
11	[6]	The complexity of the tool is not computed. This technique considers only syntactic similarity.
12	[22]	This technique is applied only to procedural language software.
13	[23]	The complexity of this technique is not computed. The technique can assess the quality of only the Java language.
14	[18]	It does not compute precision and recall.
15	[15]	A code surfer is required to generate the PDGs. The complexity of the tool is not computed.
16	[21]	The Code surfer is required to generate the PDGs.
17	[9]	The complexity of the algorithm is not computed. It requires a code surfer to generate the PDGs. The compiled code is only reflected in the AST and PDGs.
18	[10]	The complexity of the approach is not computed.
19	[2]	It does not compute the complexity of the approach.
20	[28]	It does not compute the complexity of the approach. This approach works only on java language. The computational cost of this approach is expensive.
21	[29]	This framework works only on java language. The complexity of the tool does not compute.

V. CONCLUSIONS

Software clone results from the copy and paste of the code snippets from one to another software or to same the software. More research is done on the detection than the management of the clone find in the software.

In this paper, we discuss the different techniques and tools. Researchers and authors used different algorithms for the identification of clones. From the past two decades, most of the techniques detect the syntactic clone that deals with the syntax. Type 4 or functional clones are hard to find due to their nature. This paper guides the software developer to use techniques that are corresponding to their requirements for the detection of a clone. The researcher also makes changes or updates the existing techniques.

ACKNOWLEDGMENT

I am very thankful to Dr. Sandeep Sharma.

REFERENCES

1. Abd-El-Hafiz S.K., (2012) “A metrics-based data mining approach for software clone detection”, *2012 IEEE 36th International Conference on Computer Software and Applications*, Izmir, pp. 35-41.
2. Agrawal A. and Yadav S.K., (2013) “A hybrid-token and textual based approach to find similar code segments”, *2013 Fourth International Conference on Computing, Communications and Networking Technologies (ICCCNT). IEEE, 2013*, Tiruchengode, pp. 1-4.
3. Baker B.S, (1995) "On finding duplication and near-duplication in large software systems”, *Proceedings of 2nd Working Conference on Reverse Engineering*, Toronto, Ontario, Canada, pp. 86-95.
4. Basit H.A., et al., (2007) "Efficient token-based clone detection with flexible tokenization", *In Proceedings of the 6th European Software Engineering Conference and Foundations of Software Engineering, ESTEC/FSE 2007*, Dubrovnik, Croatia, pp. 513-515.
5. Baxter I.D., Yahin A., Moura L., and Sant‘Anna M., Bier L., (1998) “Clone Detection Using Abstract Syntax Trees”, In *Proceedings of the 14th International Conference on Software Maintenance (ICSM'98)*, Bethesda, Maryland, USA, pp. 368-377.
6. Bulychev P., Minea M.,(2008) “Duplicate code detection using anti-unification”, *In Spring Young Researchers Colloquium on Software Engineering, SYRCoSE 2008*, St. Petersburg, Russia, pp. 51-54.
7. Cordy J.R, (2003) “Comprehending reality: Practical barriers to industrial adoption to software maintenance automation”, *In Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC'03)*, Portland, OR, USA, pp. 196-206.
8. Evans W.S., Fraser C.W., and Ma F., (2007) “Clone detection via structural abstraction”, *In Proceedings of the 14th Working Conference on Reverse Engineering, WCRE 2007*, pp. 150-159.
9. Gabel M., Jiang L., and Su Z.,(2008) "Scalable detection of semantic clones", *In Proceedings of the 30th International Conference on Software Engineering, ICSE 2008*, Leipzig, Germany, pp. 321-330.
10. Higo Y., et al., (2011) “Incremental code clone detection: A PDG-based approach”, *18th Working Conference on Reverse Engineering, WCRE 2011*, Limerick, pp. 3-12.
11. Johnson J.H., (1994) “Substring Matching for Clone Detection and Change Tracking”, *In Proceedings of the 10th International Conference on Software Maintenance*, Victoria, British Columbia, Canada, pp. 120- 126.
12. Kamiya T., Kusumoto S., and Inoue K., (2002) "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code", IEEE *Transactions on Software Engineering* 28(7), pp. 654- 670.
13. Kapser C. J. and Godfrey M.W., (2008) “Cloning considered harmful considered harmful: Patterns of cloning in software", *Empirical Software Engineering* 13(6), pp. 645-692.
14. Kim M., Bergman L., Lau T., Notkin D., (2004) "An Ethnographic Study of copy and paste programming practices in OOPL”, *In Proceedings of 3rd International ACM-IEEE Symposium on Empirical Software Engineering (ISESE’04)*, Redondo Beach, CA, USA, pp. 83–92.
15. Komondoor R. and Horwitz S., (2001) "Using slicing to identify duplication in source code", *In Proceedings of the 8th International Symposium on Static Analysis, SAS*, Paris, France, LNCS 2126, pp. 40-56.
16. Koschke R, (2006) "Survey of research on software clones", *Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software*, p. 24.
17. Koschke R., Falke R., and Frenzel P., (2006) “Clone detection using abstract syntax suffix trees”, *In Proceedings of the 13th Working Conference on Reverse Engineering, WCRE 2006*, Benevento, Italy, pp. 253-262.
18. Lanubile F., Mallardo T., (2003) “Function clone detection in web applications” *In Proceedings of the Seventh European Conference On Software Maintenance And Reengineering (CSMR’03)*, Benevento, Italy, pp. 379-386.
19. Lee S. and Jeong I., (2005) “SDD: high-performance code clone detection system for large scale source code”, *In Proceedings of the Object-Oriented Programming Systems Languages and Applications Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA Companion 2005*, San Diego, CA, USA, pp. 140-141.

20. Li Z., Lu S., and Myagmar S., Zhou Y., (2006) "CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code", *IEEE Transactions on Software Engineering* 32(3), pp. 176-192.
21. Liu C., Chen C., Han J., and Yu P.S., (2006) "GPLAG: Detection of software plagiarism by program dependence graph analysis", *In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2006,* ACM, pp. 872-881.
22. Mayrand J., Leblanc C., and Merlo E.M., (1996) "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics", *In Proceedings of the 12th International Conference on Software Maintenance (ICSM'96),* Monterey, CA, USA, pp. 244-253.
23. Patenaude J.F., et al., (1999) "Extending software quality assessment techniques to java systems" *In Proceedings of the 7th International Workshop on Program Comprehension, IWPC* Pittsburgh, PA, pp. 49-56.
24. Rieger M., (2005) "Effective Clone Detection without Language Barriers", *Ph.D. Thesis, University of Bern, Switzerland*.
25. Roy C. K. and Cordy J. R., (2007) "A survey on software clone detection research", *Queen's School of Computing TR*541, Kingston, Ontario, Canada, p. 115.
26. Roy C.K. and Cordy J.R, (2008) "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization", *In Proceedings of the 16th IEEE International Conference on Program Comprehension, ICPC 2008,* pp. 172-181.
27. Sager T., et al., (2006) "Detecting similar Java classes using tree algorithms", *In Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006,* pp. 65-71.
28. Sheneamer A. and Kalita J., (2016) "Semantic Clone Detection Using Machine Learning", *15th IEEE International Conference on Machine Learning and Applications,* pp.1024-1028.
29. Sheneamer A., (2018) "CCDLC Detection Framework: Combining Clustering with Deep Learning Classification for Semantic Clones" *17th IEEE International Conference on Machine Learning and Applications,* pp.701-706.
30. Wahler V., et al. (2004) "Clone Detection in Source Code by Frequent Itemset Techniques", *IEEE 4th international workshop on Source code analysis and manipulation,* pp. 128-135.
31. Walenstein A. and Lakhotia A., (2007) "The Software Similarity Problem in Mal-ware Analysis", *In Proceedings Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software*.
32. Yang W., (1991) "Identifying syntactic differences between two programs", *In Software Practice and Experience, 21(7),* pp. 739-755.
33. Yuan Y. and Guo Y., (2012) "Boreas: an accurate and scalable token-based approach to code clone detection", *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering,* ACM, pp. 286-289.

AUTHORS PROFILE

Amandeep Kaur, Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar, India. Amandeep Kaur is pursuing Mtech (CSE) from Guru Nanak Dev University, Amritsar. Her research interests include software engineering and machine learning. She did her Bachelors of Engineering in Computer Science and Technology from Sant baba bhag Singh institute of engineering and technology, Jalandhar, India

Sandeep Sharma, is head Professor in the Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar, India. He received his B.E, M.E, and PhD in Computer Science and Engineering His area of research interest is in Big Data, Cloud computing, and parallel processing.