International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 6, Issue 5, September- October 2017 ISSN 2278-6856

CODE CLONE DETECTION: A REVIEW AND COMPARATIVE ANALYSIS

Ekta Manhas[1], **Er.Samriti Rana**[2]

[1]Research Scholar Rayat Group Of Institutions, Ropar

[2]Assistant Professor Rayat Group Of Institutions, Ropar

Abstract
Copying, modification of a code block is recognized as cloning is the general mean of software usage. It is mostly utilized in the community of software development. The main objective of code clone detection is the identification of code clone with the replacement with a unique function that mimics the single instance behaviour from the clone set. In the last decade, the detection of code duplication is resulted into different tool that finds the duplication of code blocks. This paper discusses the varied methods of code clone detection with tools and the techniques. The process of clone detection with the removal is also discussed.

Keywords: Code clone, code clone detection, code clone removal.

1.INTRODUCTION

The research has shown that applications for large scale normally include code clones [1]. The code clone existence or redundant code fragment duplication develops more ambiguity and instability that give rise to tasks of route maintenance being complex and gives moretime for detecting a single code clone fragment. The concept of cloning may enhance the probability of bugs only if the bug is being detected in the source code and is reutilized by pasting and copying and later the bug can be found in the fragment of pasted code. For eliminating the bugs, the framgments should be detected. The code clones are generally consisted of four types, namely, Type I, Type II, Type III are textual and Type IV are functional [2].
There are number of reasons for code duplication and are defined below

i. When the situation occurred when the merging of two same system with same functionalities takes place for developing a novel one that is resulted in code duplication even if the systems are implemented by varied teams.
ii. The main cause for code duplication is the less time span given to the developers for completing the project. The developers find the easy way for handling the problem because of the time limit. Therefore, they find the same code related to the project assigned and later just copy and paste the related code.

Few drawbacks are there for the code duplication as discussed below:

i. The code clone has wrong impact on the reusability, maintainability and software quality.
ii. If some code has bug in the code segment and is copied and pasted from some system than the big will remain the segment which is later hard to maintain.
iii. When the duplicated code is presented in the system than it may result to bas design that can enhance the system cost.
iv. In the software system if the duplicate code for understanding the additional time required than it become difficult for update the system.

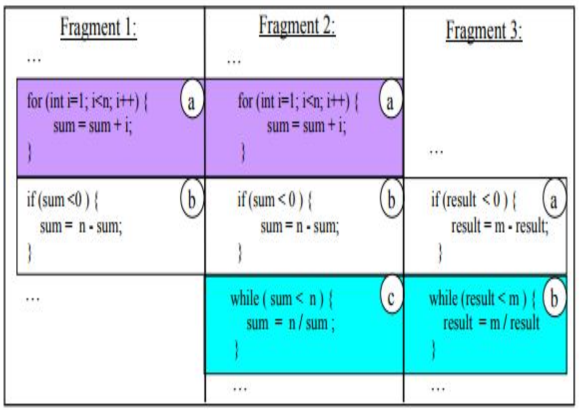
2.CLONE TERMINOLOGIES

The clones are recognized in the form of cline pairs and clone classes that tells about the similarity among the different code clone fragments. If there is a similarity for code sequences, than code relation exists among the code fragments. Like, strings with no white space, character strings, token type sequence, transformed token sequence and so on [3].
Below table is defining the terminologies of code clone:

Table 1: Code Clone Terminologies

Terminology	**Explanation**
Code Fragment	• It is consider as the code line sequence with different kinds of similarity among various code fragments in source code. • The source code can have comments or it can be without comments. • Example: Begin end block, sequence of statements and so on.
Clone Pair	• It is a term used if there exists any clone relation in the code fragment pair. • It is consider as the pair of code fragment with some similarity among them. • Example is shown in

	figure 1.
Clone Set	• It is known as the set of all similar or identical fragments.
Clone Class	• It is known as the set of the clone pairs having existing clone pairs with similar clone relation among them. • Example is depicted in figure 1.
Code clone types	• As per it program text and functionalities, the two code fragments are taken as same. • The type first of clone is the outcome of the activities of copy and paste. ✚Type I: Similar code fragments except for changes in the layout, white space and comments. ✚Type II: Syntactically identical fragments apart from literals, whitespace, identifiers variations, comments and layouts. ✚Type III: Copied fragments than later modifications like removed or added statements, variations in literals, identifiers, whitespace, comments and layouts. ✚Type IV: Two and more code fragments that defines the similar process but are executed by diverse syntactic variants.

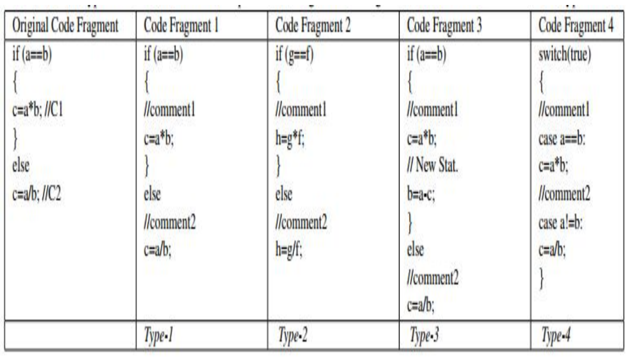


```
Fragment 1:
...
for (int i=1; i<n; i++) {
    sum = sum + i;
}
if (sum <0 ) {
  sum =  n - sum;
}
...
```

```
Fragment 2:
...
for (int i=1; i<n; i++) {
    sum = sum + i;
}
if (sum < 0 ) {
    sum = n - sum;
}
while ( sum <  n ) {
  sum  =  n / sum ;
}
...
```

```
Fragment 3:
...
if (result  < 0 ) {
    result = m - result;
}
while (result < m ) {
  result  = m / result
}
...
```

Figure 1: Clone pair and clone class



Original Code Fragment	Code Fragment 1	Code Fragment 2	Code Fragment 3	Code Fragment 4
if (a==b) { c=a*b; //C1 } else c=a/b; //C2	if (a==b) { //comment1 c=a*b; } else //comment2 c=a/b;	if (g==f) { //comment1 h=g*f; } else //comment2 h=g/f;	if (a==b) { //comment1 c=a*b; // New Stat. b=a-c; } else //comment2 c=a/b;	switch(true) { //comment1 case a==b: c=a*b; //comment2 case a!=b: c=a/b; }
	Type-1	*Type-2*	*Type-3*	*Type-4*

Figure 2: Code clones Types

Above figure represents an original code and the four clones. An original code is given and the fragments are mentioned w.r.t to each clone type [4].

3. CLONE DETECTION TECHNIQUES

This section defines the techniques for code clone detection [5] [6]:

i. Textual Approach (Text Based technique)

This approach has no source code transformation before the comparison being drawn on both sides. In variety of cases, the original source code is utilized as it is presented in the process of clone detection. For example, NICAD, SDD, Simian 1 and so on.

ii. Lexical Approach (Token Based technique)

This technique initially converts the source code in the lexical sequence, known as tokens by utilizing compiler style lexical analysis. The sequence later scans the not required duplication of token sequence by means of original code that is resulted as clones. These types of approaches are normally more resilient for small variations in the code. It is defined as spacing, formatting and renaming which is different as compare to textual techniques. For example CCFinder, Dup, CPMiner and so on.

iii. Syntactic Approach

This approach utilizes a parser for converting a source program in abstract syntax trees or parse trees that can be processed by using structural metrics or tree matching for finding the clones. For example: Deckard, Clone Dr and CloneDigger and so on.

iv. Semantic Approach

This approach has been developed by utilizing the static program because it gives the in-depth data as compare to the syntactic similarity. In different approaches, the provided approach is given in the form of PDG (Program dependency graph) or in the form of statements or expressions but the edges shows the data or control dependencies. For example, GPLAG, Duplix and so on.

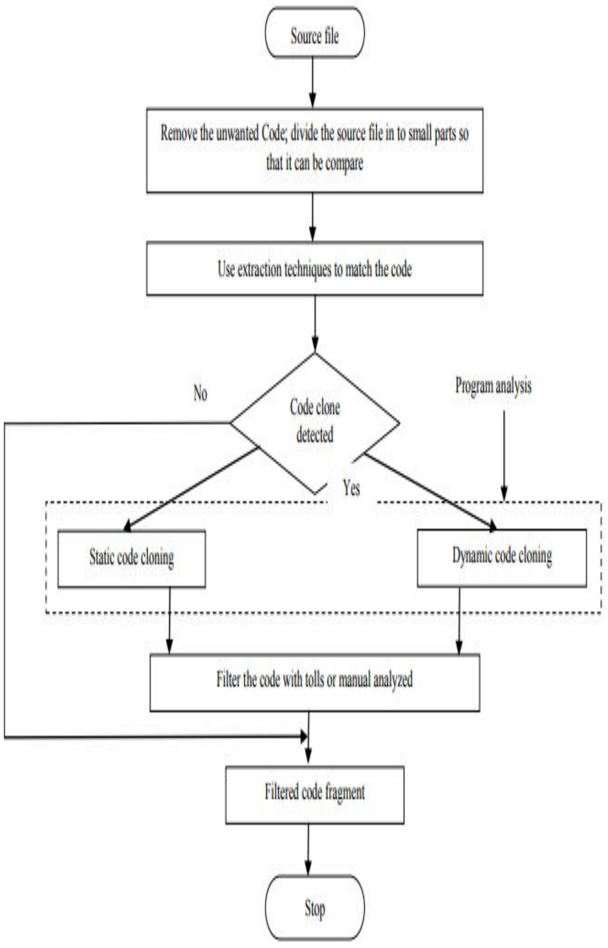

Source file
Remove the unwanted Code; divide the source file in to small parts so that it can be compare
Use extraction techniques to match the code
No
Code clone detected
Program analysis
Yes
Static code cloning
Dynamic code cloning
Filter the code with tolls or manual analyzed
Filtered code fragment
Stop

Figure 3: Process of clone detection

Above figure is representing that a code detector is trying to find the pieces of code that has more similarities in the system source text. The comparison of each clone is done by the detector and the tool support is needed for identifying unique clone.

Figure 3 is defining the general steps in the process of code clone detection.

Table 2: Clone Code Classification and techniques

	Text based	Token based	AST based	PDG based
Category	Textual	Textual	Semantic	Semantic
Supported Clone	Type I	Type I, II	Type I, II, III	Type I, II, III
Complexity	O (n)	O (n)	O (n)	O(n^2)
n Meaning	Lines of code	Number of token	Node of AST	Node of PDG

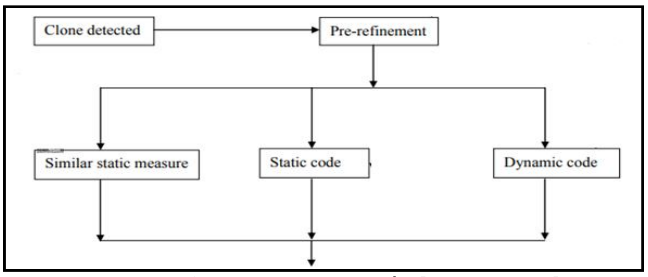


Figure 4: Process of Clone removal

Figure 4 is defining the code clone removal process. After the detection of clone, the conceptual view is seen only [7]. If it is same as the clone which is being detected, than same method will be implied. If the code is static, because it is language independent than a novel method will be applied and if the code is dynamic than the execution is done with the help of tools [8].

4. TECHNIQUES OF CODE CLONE DETECTION FOR DIFFERENT PROPERTIES

Below table defines the comparison of code clone detection techniques by means of different properties [8]. The comparison has been drawn for text based, token based, tree base, PDG based and Metric based techniques on the basis of representation, transformation, comparison dependent, computational complexity and refactoring opportunities and dependency language properties [9].

Table 3: Comparative analysis of code clone detection techniques w.r.t varied properties

Properties	Text based	Token based	Tree based	PDG based	Metric based
Representation	Normalized source code	In the representation of tokens	Shown in the abstract syntax tree form	Considered as program dependency graph set	Metrics value set
Transformation	Eliminates comments and whitespace	Token is produced from the source code	AST is produced from the source code	It is produced from the source code	For finding the metrics value, AST is produced from the source code
Comparison dependent	Token of line	Token	Node of tree	Program dependency graph node	Metrics value
Computational complexity	Algorithm dependent	Linear	Quadratic	Quadratic	Linear
Refactoring opportunities	Better for exact matches	Some post processing required	Better for refactoring as it finds syntactic clones	Better for refactoring	Manual inspection is needed
Dependency language	Ease for adaptability	It requires a lexer but no syntactic knowledge is needed	Parser is needed	Edge syntactic knowledge and PDG is needed	Parser is needed

5. CONCLUSION

Code clone detection is known as a live problem in industry with a lot of work on removing and detecting the clones from the software. But the problem of detection and removal has not been resolved yet. The tools of code clone have to be integrated in standard IDE for having widespread adoption. This paper mainly focuses on describing the detection techniques with the methods of code clone method. The process of removal is also discussed. The code clone detection plays a vital role in the research of software evolution in which the attributes of similar code entity are observed for number of versions. It is being concluded that the clones are the significant aspects for software evolution. If the system has to be evolved than its clones should make consistent variations.

References

[1]. Roy, Chanchal K., James R. Cordy, and Rainer Koschke. "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach." Science of computer programming 74.7 (2009): 470-495.
[2]. Rattan, Dhavleesh, Rajesh Bhatia, and Maninder Singh. "Software clone detection: A systematic review." Information and Software Technology 55.7 (2013): 1165-1199.
[3]. Pate, Jeremy R., Robert Tairas, and Nicholas A. Kraft. "Clone evolution: a systematic review." Journal of software: Evolution and Process 25.3 (2013): 261-283.
[4]. Roy, Chanchal K., Minhaz F. Zibran, and Rainer Koschke. "The vision of software clone management: Past, present, and future (keynote paper)." Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. IEEE, 2014.
[5]. Pate, Jeremy R., Robert Tairas, and Nicholas A. Kraft. "Clone evolution: a systematic review." Journal of software: Evolution and Process 25.3 (2013): 261-283.
[6]. Kim, Miryung, et al. "An empirical study of code clone genealogies." ACM SIGSOFT Software Engineering Notes. Vol. 30. No. 5. ACM, 2005.
[7]. Baxter, Ira D., et al. "Clone detection using abstract syntax trees." Software Maintenance, 1998. Proceedings., International Conference on. IEEE, 1998.
[8]. 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.
[9]. Bellon, Stefan, et al. "Comparison and evaluation of clone detection tools." IEEE Transactions on software engineering 33.9 (2007).
[10]. Basit, Hamid Abdul, and Stan Jarzabek. "Efficient token based clone detection with flexible tokenization." Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM, 2007.
[11]. Göde, Nils, and Rainer Koschke. "Incremental clone detection." Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on. IEEE, 2009.
[12]. Schulze, Sandro, and Martin Kuhlemann. "Advanced analysis for code clone removal." Proceedings des Workshops der GI-Fachgruppe Software Reengineering (SRE), erschienen in den GI Softwaretechnik-Trends 29 (2). 2009.