Code Clone Detection Using Hybrid Approach

Sayali Sharad Patil
Computer Engineering Department
SSBT's College of Engineering and Technology, Bambhori
Jalgaon, India

Sachin Santosh Chaudhari
Computer Engineering Department
SSBT's College of Engineering and Technology, Bambhori
Jalgaon, India

Ashwini Mukunda Sonawane
Computer Engineering Department
SSBT's College of Engineering and Technology, Bambhori
Jalgaon, India

Sonal Siddharth Salunke
Computer Engineering Department
SSBT's College of Engineering and Technology, Bambhori
Jalgaon, India

Makarand Ramakant Bhole
Computer Engineering Department
SSBT's College of Engineering and Technology, Bambhori
Jalgaon, India

***Abstract*—Many researchers have look over distinct techniques to detect duplicate code in programs exceeding thousand lines of code. These techniques have drawback of finding either the structural or functional clones. Code clones are the duplicated code that degrade the software quality and hence increase maintenance value. Detection of code clone in software system is extremely necessary to improve design structure and quality of software product. The proposed lightweight weight hybrid approach uses textual comparison and template conversion for detection of method level syntactical and semantic clones in C file and functional clones in C and Java file.**

***Keywords*— Clone detection, Functional clones, Textual analysis**

I. INTRODUCTION

Copying code fragments and then reusing them through the paste option with or without any or some minor modification and adaptation is called Code Cloning and the pasted code chunk is called a clone [3]. In the software system copied code chunk and code clones are considered as bad smell of the software. It is observed that code clone has bad effect on the maintenance of the software system [4]. To get rid of clones from the software systems is very necessary and quite beneficial. These clones are syntactically or semantically similar. Several studies show that it is hard to detect system which contains the code clones as compared to other software system which does not contain any clone. Cloning May increases the bug possibility, if any bug is found in the code and that code is reused by copying and pasting then that bug is also found in that pasted code portion. For fixing the bug all these code chunk should be detected. Code clones are basically of four types, where the first three Type I, Type II, Type III are textual and last one Type IV is functional [11].

II. BACKGROUND

A. Reasons of Code Duplication

Here are many reasons for code duplication. Reuse of code design and logic is the main cause of code duplication. Sometimes there is a need to combine two similar system having same functionalities to create a new one which result duplication of code even both of the system are developed by different teams or peoples[8]. One of the major reason of code duplication is the time limit assigned to developers. Developers find the easy and simple solutions of the problem due to time limit. They find the similar code related to their project and they just copy and paste the existing code [2].

B. Types of Clone

Based on functionalities and program text, two code fragments are said to be similar. The primary kind of clone are primarily the results of copy and paste activities. Within the following type of clone's type-I, type-II and type-III clones are mainly based on the textual similarity and type-IV clones are mainly based on the functional similarity [11].

- Type-I Clone: Type-I clone is an exact same copy without modifications apart from whitespace and comments [3]. In type I clone, two code fragments are similar to one another. However, there can be some variations in white space, comments and layouts. The clone pair (a, b) is of type-1 which have exactly the same code except the alignment, space and comment [12].

Source Code(a)	Type-1 Clone(b)

int main()
{
int a=1;
int b=a+9;
return b;
}
	int main()
	{
	int a=1;
	int b=a+9;
	return b;
	}

- Type II Clone: Type-2 is a syntactically same copy; apart from some changes in variable name, data type, identifier name, etc. The clone pair (a, c) is of type-2 which have minor variations in function names and parameters.

Source Code(a)	Type-2 Clone(c)

int main()
{
int a=1;
int b=a+8;
return b;
}
	int fun2()
	{
	int s=1;
	int t=s+8;
	return t;
	}

- Type III Clone: Type-3 is a copied chunk with additional changes. Statements can be modified, added or removed in addition to variations in identifiers, literals, types, layout and comments [3]. The clone pair (a, d) is of type-3 with additional statements in code, as they need not be functionally similar.

Source Code(a)	Type-3 Clone(d)

int main()
{
int a=1;
int b=a+8;
return b;
}
	int fun2()
	{
	int a=1;
	int c=a+5;
	c=a++;
	return c;
	}

- Type IV Clone: Two code chunks that perform a same calculation but implemented through completely distinct syntactical variations are type-4 clone. The clone pair (a, e) is of type-4 clones with no similarity in code, but the output of the functions are same [12].

Source Code(a)	Type-4 Clone(e)

int main()
{
int a=2;
int b=a+9;
return b;
}
	int add()
	{
	int n=10;
	return ++n;
	}

The results of the code clone detection are presented as clone pairs and clone clusters.

Clone Pair (CP): Clone Pair is pair of code portions/fragments that are similar or similar to each other [12].

Clone Cluster: Clone Cluster is the union of all clone pairs that have code portions in common [12]

III. LITERATURE SURVEY

There has been over a decade of analysis within the area of software clone. Clone detection analysis has proved that software systems have 9%-17% of duplicated code [7]. Thummalapenta indicated that, in [12] most of the cases, clones are modified systematically and for the remaining inconsistently modified cases, clones undergo independent evolution. Effective code clone detection will support perfective maintenance. Comparison and analysis of code clone detection techniques are administrated by Bellon, Koschke and Roy and Cordy [6]. A clone detection method is typically done by changing the source code into another type that's handled by an algorithmic program to detect the clones [7]. Token-based technique use a similar sequence matching algorithmic program. However, its accuracy isn't that adequate because the normalization, and also token conversion method may bring false positive clones in result set [12]. Several of the clone detection approaches have used Abstract Syntax Tree (AST) and suffix tree illustration of a program to search out clones. A number of the clone detection techniques use an AST that's generated by a preexisting parser [12]. Baker describes one amongst the earliest applications of suffix trees for the clone detection method [3]. An algorithmic program based on feature-vector computation over AST was applied by Lee to detect similar clones. However, all of them use parsing, which ends in heavy-weighted approach. Text-based techniques are investigated by comparing two code fragments with one another to search out longest common subsequences of same strings to find clones [6]. Although these techniques find clones they are not low in precision values. Metric-based techniques establish a group of appropriate metrics to find a specific kind of clone. By a quantitative assessment of the metric values within the ASCII text file, the clone detection is finished. Marco Funaro proposed Hybrid technique [7]. A proposed a hybrid technique using Abstract Syntax Tree to identify clone candidates and textual methods to discard false positives. Leitao additionally proposed a hybrid approach with the combination AST and PDG [4]. Each approaches use parsing which ends in heavy-weight. As text-based techniques preserve higher recall, metrics-based techniques preserve higher precision and each of them are light-weight, a hybrid technique with the combination of textual analysis and comparison, is experimenting for the detection of all four types of clones [12].

IV. EXISTING TECHNIQUES

In the literature many kinds of clone detection techniques are given. For the analysis purposes most of the techniques are used, whereas some of them are used for commercial purpose. Following are some existing techniques for code clone detection.

A. Text-Based Techniques

In the text based technique the source code chunks are considered as sequence of line [5]. Once removing the various comments, whitespace by applying the various transformations the code fragment are compared with one another. Once the two code fragment are found to like one another to some extent they are referred to as clone pair or clone pairs form the clone class [6]. Text based technique is efficient technique however it will find only Type I clones. Text based approach can't find the structural type of clone having a similar logic however different coding [9]. Within the text based approach following transformations are applied on source code.

1) Comments Removal: Within the code fragment ignore all the comments.

2) White Space Removal: Within the code fragment removes all the tabs and blank lines.

Though text based approach will find only type 1 clone. This method cannot detect the structural type of clones having identical logic however completely distinct coding.

B. *Token-based Techniques*

In the token-based technique, initial sequence of tokens is generated from the source code. For changing the source code into tokens it needs a lexer[5]. Lexer convert the source code into tokens then the various transformation are performed by adding, changing or deleting some tokens. For finding the duplicated code, the code is scanned. Therefore the code chunks representing the duplicated code returned as clones. Token based technique is able to find only type I, type II clone [11].

C. *Tree-baseed Techniques*

In the tree-based approach from the source code a parse tree or an abstract syntax tree is created. This method creates sub trees instead of making tokens from each statements [7]. The code then said to be code clone if the sub trees match. With the help of parser of a language similar sub trees are searched within the tree using tree matching algorithm or structural metrics then the code of similar sub trees are returned as clone pairs[10]. Abstract syntax tree have the entire data concerning the code. The result obtained from this method is kind of efficient however to create a abstract syntax tree is difficult for a large code and therefore the scalability is also not good [12].

D. *PDG-based Techniques*

Program Dependency Graph (PDG) technique is more efficient then tree based technique. Program dependency graph shows data flow and control flow information [7]. First the program dependency graph is obtained from the source code then to search out the similar sub graphs or clones many type of sub graph matching algorithm are applied and returned as clones. This method will find each semantic and syntactical clones however just in case of large code to get the program dependency graph is incredibly tough [6].

E. *Metric-based Techniques*

In Metrics based Technique initial differing kinds of metrics of the code like number of lines and number of functions are calculated and compare these metrics to search out the clones. Metrics based technique doesn't compare code directly [4]. To search out the code clones many style of code metrics are utilized by clone detection techniques. Most of the time, for calculating the various type of metrics the source code is converted into abstract syntax tree or program information graph [7]. Metrics are calculated from the name, layout, control flow and expression of the functions [10].

V. PROPOSED APPROACH

All the benefits and drawbacks of various approaches mentioned in above sections that clearly show that although several techniques but still none is able to search out the clones properly. Thus we tend to propose a hybrid technique that is able to search out more number of true positives. This approach will find all the clones within the system regardless of their place and will show to the programmer [3]. So that after or during the development of the code the programmer itself can determine the chunks that contain the clone and may decide whether or not to get rid of the clone or it's a good smell[12]. Within the proposed approach, two code chunks can be compared. Firstly, the preprocessing is applied on files. Preprocessing involves removing of comments and white spaces. A LWH (Light Weight Hybrid) approach has been proposed with a combination of textual comparison and template conversion. As there is no need for external parsing, this approach is of light weight [8]. Moreover, a model has been arrived to find syntactical and semantic clones which is able to cover all four types of clones [12]. The proposed LWH approach is able to find method clones in C projects and function level clones in C and Java projects.

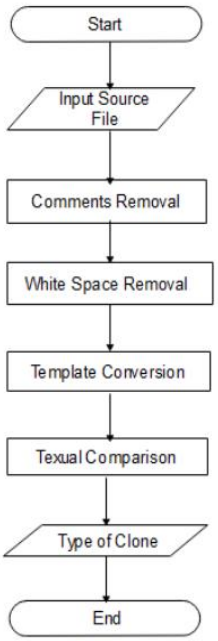

Fig. 1. Flowchart for Code Clone Detection using Hybrid Approach.

VI. ADVANTAGES

Code clones are the duplicated code which degrade the software quality and hence increase maintenance cost. Detection of code clone in software system is very necessary to enhance design, structure and quality of software product. Code clone duplication has several benefits within the development of software project. Some of them are as follows.

- Detects library candidate: Code fragment proves its usability by coping and reusing multiple times within the system which will be incorporated in a library and announce its reuse potential officially [13].
- Understanding Program: It is possible to get an overall idea of alternative files containing similar copies of the fragment, if the functionality of a cloned fragment is understood [13].
- Helps aspect mining research: Detecting code clone is also necessary in aspect mining to find cross-cutting issues [13].
- Detects malicious software: To find malicious software system clone detection techniques will play an important role [10]. By comparing one malicious software system to another, it's possible to search out the evidence where match parts of another [13] parts of the one software system.

- Helps detecting plagiarism copyright content: Finding similar code may additionally helpful in detecting plagiarism and copyright infringement [13].
- Software evolution: Clone detection techniques are successfully used in software system evolution analysis by looking at the dynamic nature of different clones in numerous versions of a system [13].

VII. DISADVANTAGES

Apart from advantages of code clones, it has severe impact on the standard, reusability and maintainability of a software. The following are the list of some drawbacks of having cloned code in an exceedingly system.

- Increased probability of bug propagation: If a code chunk contains a bug and that segment is reused by coping and pasting without or with minor changes, the bug of the original chunk may remain in all the pasted chunk in the system and therefore, the possibility of bug propagation may increase significantly in the system [13].
- Increased probability of introducing a new bug: In many cases, only the structure of the duplicated chunk is reused with the developer's responsibility of adapting the code to the current need. This process can be error prone and may discover new bugs in the system [13].
- Increased probability of bad design: Cloning may also introduce bad design, lack of good inheritance structure or abstraction [5]. Consequently, it becomes difficult to reuse part of the implementation in future projects. It also badly impacts on the maintainability of the software [13]
- Increased difficulty in system upgradation: Because of duplicated code in the system, one needs additional time and attention to understand the existing cloned implementation and concerns to be adapted, and therefore, it becomes difficult to add new functionalities in the system, or even to change existing ones [13].
- Increased maintenance cost: If a cloned code chunk is found to be contained a bug, all of its similar counterparts should be investigated for correcting the bug in question as there is no guarantee that this bug has been already eliminated from other similar parts at the time of reusing or during maintenance [13].

CONCLUSION

A copy and paste activity which is done by developer is the main reason of code cloning [3]. It looks like a simple and effective method, but these copy and paste actions are not documented which create a bad effect on the software quality. The proposed approach is mainly design to overcome drawback of existing techniques to detect clone. The proposed approach is able to detect all four types of clones accurately. The hybrid approach uses combination of template conversion and textual comparison to detect syntactic and semantic levels of clones.

REFERENCES

[1] Hummel B Al-Batran B, Schatz B. "Semantic clone detection for model-based development of embedded systems". Institute of computer science pages 258-272,2011

[2] Mandeep Singh Sandhu Amandeep Kaur. "Software code clone detection model using hybrid approach". IJCT, 3(2), October 2012.

[3] B. Baker. "A program for identifying duplicated code, in: Proceedings of computing science and statistics: 24th symposium on the interface". International Journal of Computer Applications, 24:49–57, 1992.

[4] Madan Lal Manpreet Kaur. "Code clone detection using function based similarities and metrics". International Journal of Emerging Research in Management and Technology, 4(7):156–159, Jully 2015.

[5] Dr. Shahanawaj Ahamad Mohammed Abdul Bari. "Code cloning: The analysis, detection and removal". International Journal of Computer Applications, 20(17), April 2011.

[6] Adamov R. "Literature review on software metrics". Institute of computer science, 1987.

[7] Chanchal Kumar Roy and James R. Cordy. "A survey on software clone detection re-search". Technical Report, page 541, September 2007.

[8] Kodhai. E Rubala Sivakumar. "Code clones detection in websites using hybrid approach". IJCA, 48(13), June 2012.

[9] Yogita Sharma. "Hybrid technique for object oriented software clone detection". Masters Thesis submitted at Lulea University of Technology, 2011.

[10] "Finding Clones with Dup: Analysis of an Experiment". IEEE transactions on software engineering. 2007, pages 608–621, September 33.

[11] Mohammed, Rowyda, Amal Elsayed, and Mostafa-Sami Mostafa. "Clone Detection Using DIFF Algorithm For Aspect Mining", International Journal of Advanced Computer Science and Applications, 2012.

[12] Kodhai, Egambaram, and Selvadurai Kanmani. "Method-level code clone detection through LWH (Light Weight Hybrid) approach", Journal of Software Engineering Research and Development, 2014.

[13] Balwinder Kumar, Dr. Satwinder Singh, " Code Clone Detection and Analysis Using Software Metrics and Neural Network-A Literature Review", International Journal of Computer Science Trends and Technology (IJCST) – Volume 3 Issue 2, Mar-Apr 2015.