Hybrid approach for Detecting Code Clone by Metric and Token based comparison

Deepali
Assistant Professor
Guru Nanak College
Budhliada, India

Ankur Gupta
Software Engineer
Amadeus India Pvt. Ltd.
Banglore

Chirag Batra
Technology Lead
TCS
USA, New York

Abstract—In software development process, copying of existing code fragment and pasting them with or without modification is a frequent process. Clone means copy of an original form or duplicate. Software clone detection is important to reduce the software maintenance cost and to recognize the software system in a better way. There are many software code clone detection techniques such as text-based, token-based, Abstract Syntax tree based etc. and they are used to spot and finding the existence of clones in software system. One of the approaches to detect code clones is by analysis of different metrics for the programs. Another approach is token based comparisons of two programs to detect code clone. Our technique uses the combination of metrics and token based approach using hashing algorithm by analysis of two source codes in the process of finding clones among them and the percentage of cloning is reported as result.

Keywords— software clone detection, metric based comparison, token based approach using hashing algorithm, java byte code.

I. INTRODUCTION

In general, clones are set of identical segments of code in a software system, which has a bad impact in the system. In software development approach, duplicating previous code segments in different programs with or without modification is frequent process and that duplicated code is extremely difficult to maintain. The imitation in code is known as software clone and the phenomenon is known as software cloning. Code Cloning considered as a bad smell in software development approach, duplicating previous code segments in different programs with or without modification is frequent process and that duplicated code is extremely difficult to maintain. The imitation in code is known as software clone and the phenomenon is known as software cloning. Code Cloning considered as a bad smell in software development process.

Baker [11][2] has taken a simple program and concluded that program code can be reduced by 14% based on exact matches and 61% based on parameterized matches. Several studies states that about 5% to 20% of software contains duplicate program segment [1] . Thus, it becomes very important to find the clones in programs accurately and in an efficient manner. Cloning in software is really harmful as it becomes really difficult to maintain software and its evolution. The reason behind cloning can be intentional or unintentional. There is major shortcoming of duplication in code fragment that if there is a bug in code segment to be duplicated, then that bug will be propagated at different places and the bug is to be tested multiple times at different places of the code and that would definitely increase the maintenance cost. Cloning on large extent increases the size of a system and results in design problems such as missing inheritance and procedural abstraction. The modification cost carried out after delivering of software product is figured out to be 40% - 70% of the total costs during lifetime of a system [2].

There are many software clone detection techniques and tools that differ from each other on the basis of approach used by them to detect clones. Cloning between two program codes is recognized on the basis of textual similarity and functional similarity. These types of similarities define the clone type. Based on the textual similarity we define the type 1, type 2 and type 3 clones.

Type 1 (Exact clone):- Identical code fragments except for variations in white spaces, layout and comments.

Type 2 (Renamed/Parameterized Clone): Syntactically same code fragments except for changes in identifiers, literals, types, whitespaces, layouts and comments.

Type 3 (Near miss clone):- Copied with further modification such as “change, add or remove” statements in addition to changes in identifier, literal, types, whitespaces, layouts and comment.

Based on the functional similarity we define type 4 clones imitated as a semantic clone.

Type 4 (Semantic clone):- Code fragment which are functionally similar but not textually similar.

This paper describe the hybrid clone detection technique using metric based comparison and token based comparison approach using hashing algorithm on the java source code. The metric based comparison is strictly applied on the source code for which many tools are available to find out metrics for the source code program. Later on, we apply a hashing based algorithm to execute the token based comparison of the two programs to find more accurate clone results. This paper has 5 sections, section 2 describes the related work, section 3 describes the proposed work, section 4 describes the results, and section 5 describes the conclusion and future scope.

II. BACKGROUND

Software clone detection is mainly implemented for the reorganization, preservation, refactoring and reengineering of software [4] [1]. By detecting clones, the computation cost and complexity of the software system can be minimized. Because of these factors software clone detection is new and an important research area.

Following are the software clone detection approaches:-
A. Textual approach:
Two code fragments are compared with each other to find matched sequence of texts or strings. If match is found then code is a clone pair by the detection technique.

B. Token based approach:
It needs a parser or lexer to normalize the code in form of token. So every line of source code is transformed into tokens then comparison applied on intermediate representation of code. The sequences of lines are compared through different algorithms. This technique is slower than text based approach and it is more robust.

C. Abstract syntax tree approach:
The actual source code is parsed into abstract syntax tree (AST) or parse tree and traverse the tree for finding similar sub tree. If match is found for sub tree is termed as clone. AST based approach finds even better results than the text and token based approaches but it is very complex to create an abstract syntax tree and clone detection using AST is costly procedure on both time and memory.

D. Program dependency graph based approach:
It shows the control flow and data dependencies. When PDG is achieved from source code, graph isomorphism based comparison is applied to find match. For larger code it is very difficult to obtain PDG.

E. Metric based approach:
As an alternative of comparing two codes directly, metrics from source codes are obtained and these metrics are compared to detect clone. To calculate metrics, there are numerous softwares that can be used such as Columbus, Source Monitor, Datrix, MCD Finder etc. This technique is more scalable and accurate for big software system.

F. Hybrid based approach:
Hybrid based clone detection technique combines two or more than two different approaches to detect clones which increases its complexity but it is very active technique for detecting clones as compared to above discussed technique.

III. PROPOSED WORK
Figure 1 depicts the basic block diagram for the clone detection process. The proposed technique consists of two stages. First stage is a metric based comparison and Second stage is token based approach for clone detection. Firstly, the metrics are calculated as shown in figure 2 and then the comparison algorithm is applied on these metrics to detect clones.
The results of the calculated metrics of Java source code as shown in Figure 2 are transformed into an XML file (Figure 3).

The next step is to extract the metrics from the XML file into a database (Figure 4) that can later be used for comparison using JDBC. Once the metrics are extracted, they are compared for the two source programs to find out similarity among them and that similarity is reported as a result in the form of percentage of cloning (Figure 5).

**STAGE 2:**

After calculating metrics and comparing them, once it is identified that the extent of cloning is good enough to go to the next stage, then token-based approach will be applied on the two source programs to calculate more precise code clones. In the token-based approach, firstly we have to form an intermediate representation of code in which we interchange the identifiers and keywords with some pre-defined tokens. Once the intermediate representation of the two source codes is accomplished, then we have to apply the comparison algorithm. A hashing-based comparison algorithm has been developed to compare the two intermediate representations. The hash algorithm gives us good result with less time complexity because searching using hashing can be done in $O(n)$. Figure 6 shows the intermediate representation for the two source codes. Finally, the percentage of cloning is reported as a result as shown in Figure 8. In the next section, the algorithm for token-based comparison is discussed.

A. **Algorithm for implement hashing-based comparison**

G. Take two programs and remove all types of comments in the program depending on the language of interest.

H. Also removes tabs, and new line(s) and other blank spaces from the Java program.

I. Then we have to perform an intermediate representation of programs in which we substitute all identifiers and keywords by some tokens.
J. Then, a hash value is calculated for every statement of intermediate representation of program and will be stored in a list. Note that, this hash function must be implemented in a way, such that the hash value returned must be different for different statements, and it must be same for two same statements.

L. Finally, the contents of two lists are compared to find out similarity among two codes. The algorithm is given in the table shown below.

IV. RESULTS

| Table 1: Algorithm for Clone Detection
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: F1 (program in a text file)</td>
</tr>
<tr>
<td>Output: F2 (An intermediate representation for F1)</td>
</tr>
<tr>
<td>Begin:</td>
</tr>
<tr>
<td>Take two programs &amp; ignore all kinds of comments in the F1 depending on the language of interest and remove blank lines from F1.</td>
</tr>
<tr>
<td>Store all keywords with their corresponding tokens in a 2-D array to map them later for our intermediate representation of F1.</td>
</tr>
<tr>
<td>Read F1 != end</td>
</tr>
<tr>
<td>a. Read each character one by one and store it in an array until space or new line encounter.</td>
</tr>
<tr>
<td>b. If the character is from the set { (, ), : ; ; , . , { , } , = } ---- special characters.</td>
</tr>
<tr>
<td>Retrieve the string from the character array and trim it to remove spaces from string.</td>
</tr>
<tr>
<td>else, then store the character into the character array</td>
</tr>
<tr>
<td>c. Check whether the string is corresponding to any keyword</td>
</tr>
<tr>
<td>if yes, put the token corresponding to the string into the F2.</td>
</tr>
<tr>
<td>else, put “$&quot; for that string into F2.</td>
</tr>
<tr>
<td>Check whether the next character is ‘ ', 't' or ‘\n’</td>
</tr>
<tr>
<td>if yes, put the character in F2</td>
</tr>
<tr>
<td>else, put “$&quot; for that character into F2 as a special character.</td>
</tr>
<tr>
<td>End of file,</td>
</tr>
<tr>
<td>Read F2 != end,</td>
</tr>
<tr>
<td>For each statement,</td>
</tr>
<tr>
<td>Calculate a hash value and store that hash value in a list.</td>
</tr>
<tr>
<td>End of file,</td>
</tr>
</tbody>
</table>

K. This algorithm produces two lists containing the hash values for each statement of the intermediate representation of the two programs.

V. RELATED WORK

In last decade many algorithm are proposed on software clone detection technique and every algorithm has its own
advantage and disadvantage. This unit describes the summary and overview of recent research in the area of metric based software clone detection approach.

Y. Yuan et al. [19] proposed a count matrix based clone detection (CMCD) method, which is produced while counting the rate of frequencies of every variable in conditions specified by pre-determined counting condition. The projected technique is language-independent as it depends only on variable count. That is, if we have to count the rates of frequencies of variable in certain conditions with special standards, these standards are called as counting condition. Counting condition is used to select when the count should begin. The count matrix (CM) is a group of n count vectors (CV) and compares these Counting vectors with the help of Euclidean space. The variation between two vectors is calculated by the Euclidian Distance among them in the space, i.e.

\[
D(v1, v2) = \|v1 - v2\|^2 = \sum_{i=1}^{13} (v1i - v2i)^2
\]

The CMCD perform well in extracting count-based information and it is language independent. It supports to detect clone in large programs (> 1M LoC) also it has a abilities to perform well in scenario-based evaluation.

Vidhya et al. [20] proposed an emergent technique on java directories by using a metric based approach. The proposed system has been tested with two directories of JAVA files as input and the outcomes are produced based on the matching among files in directories. The percentage of the comparison is calculated by implementing the line by line comparison of the intermediate form of the files. This proposed technique merge both the textual based approach and metric based technique. Metric based approach is straightforward hence it is a light weight method. The textual based approach is the one which give high exactness. This proposed technique also helps to notice the directory level cloning that is not structurally correlated but functionally similar.

K. Raheja et al. [4] proposed another approach using metric based technique for clone detection on byte code. Firstly the metrics are calculated from MCD Finder (java based) tool and then comparison technique is applied on these metrics to detect clone. MCD Finder tool works only for the Java language and it is easy to use. The metric based technique is used to identify the potential clone which does not directly work on source code. The proposed technique can also be merge with other techniques like abstract syntax tree based and the program dependence graph based technique to make this a hybrid method to proficiently detect semantic clones.

Zhao Li et al. [3] proposed a technique, metric space based software clone detection by an iterative method. This technique transforms the main source code fragment into metric space member through the retrieved coordinate value, and then calculates similarity level through all members on their distance within the similar metric space. The nearer the two members are, the more similar they are, from code perception. As the distance between two members become lesser, then it means they are more similar and if the distance between them increases, it means they are less similar. This technique gives advantage like exactness and scalability with the help of metric space.

VI. CONCLUSION AND FUTURE WORK

The technique detects clones (type-1 and type-2) by metrics based approach for filtering code and after that it uses token based comparisons to detect code clone. The technique detects clones by hash algorithm in token based comparison to detect whether two clones really are clones of each other and it is also able to detect the type 3 clone near miss clone by using hash algorithm. The technique can also detect code plagiarism in student’s computer lab programs. In future this approach can be integrated with other approaches like abstract syntax tree based approach and the program dependence graph approach to make this a hybrid approach to efficiently detect semantic clones.

REFERENCES


