A Survey on Code Clone and Detecting Techniques

1R.Kavitha Kumar and 2G.Santhi

1Programmer, Department of Computer Science and Engineering, Pondicherry Engineering College, Pondicherry—605 011, India.
2Assistant Professor, Department of Information Technology, Pondicherry Engineering College, Pondicherry—605 011, India.

ARTICLE INFO
Article history:
Received 12 October 2014
Received in revised form 26 December 2014
Accepted 1 January 2015
Available online 25 February 2015

Keywords:
Code Clone Detecting techniques Lexical approach
Detecting tools

ABSTRACT
Code Clone is merely repetition of code in the software, is obtained by Copy and paste of code in the software with or without modifications. Its play a vital role in the area of software maintenance as well as software quality. Code clone Identification and refactoring code has grown as an active area in software engineering community in last decades. This paper provides an overview of existing research in the field of code identification.

INTRODUCTION
In software industry Systematic software reuse is a strategy for increasing productivity and improving quality, during code reuse the code cloning is take place in the form of cut-and-paste. Due to reuse practice software maintenance has become more complicated.

1. Code Clone:
According to (Bellon et al, 2007), two main kinds of similarity between code fragments. One is functionality based and another One is similarity code based. The first kind of clone is often the result of copying a code fragment and pasting into another location. In the following the types of clones based on both the textual (Types 1 to 3) (Bellon et al, 2007) (Mohammed Abdul Bari et al., 2011) and functional (Type 4) similarities are described:

Original code
Void sum(int n) {
For(int i=1;i<n;i++){
{ sum = sum+i; p=p*i; } }

Type-1: Identical code fragments except some changes in comments, whitespaces and layout.
Type -1
Void sum(int n) {
{ For (int i=1;i<n; i++)
{ sum = sum+i; p=p*i;
} }

Type-2: Syntactically same fragments except for variations in identifiers, literals, types, whitespace, layout and comments.
Type -2
Void sum(int n) {
{ For (int k=1;k<=n; k++)
{ s = s+k; y=y*k; 
} }

Corresponding Author: R. Kavitha Kumar, Programmer, Dept. Of Computer Science and Engineering, Pondicherry Engineering College, Pondicherry, 605011, India.
Type-3: Copied fragments with modifications in fragments such as include, removed or changed the statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments.

Type-3

```c
Void sum(int n)
{
    int i=1; i<=n; i++)
    { sum = sum+i; m=1; p=p*i; }
}
```

Type-4: More than one code fragments that perform the same way of computation but are implemented by different variants.

Type-4

```c
Void sum(int n)
{ i=1; n=5;
    While (i<=n)
    { sum = sum+i; p=p*i; i++;}
}
```

1.1 Reason For Code Clone:

There are many factors behind the code clone. In that main factor is

1. Development strategy.

Many Software industries force the employee to adopt the reuse strategy to reduce the development cost. Some programming language itself insists to reuse the code.

2. Maintenance Benefits:

   ➢ Avoiding unwanted design dependencies and avoid the risk in designing phase
   ➢ Ensuring - Ensure that the system will performance for real time programs is good and also robustness of the system time programs

3. Overcoming underlying limitation:

   ➢ Language limitation :Some programming language which does not support reuse concept in that case cloning is the only way to solve the same types of sub-problems
   ➢ Programmer’s limitation: In software industries depends upon the user requirement force the programmer to develop the code within time limit. Time limit is minimum means, in that situation the programmer use code clone to finish the code within the time bound.

4. Cloning by accident:

   ➢ Language paradigm
   ➢ Programmers working style: Programmer may use same strategy to solve the same kind of problem; unaware of other programmer’s problem solving strategy in that case by accident clone is occurring in the code. Programmer itself unintentionally using the same solution for the similar kind of problems.

1.2 Drawback Of Code Clone:

According to Krinke, Jens (Anil Kumar,G, et al., 2012), any changes in the code, consistency is necessary otherwise it will increase the maintenance cost as well as very difficult in maintenance phase. Any changes in the one part code simultaneously have to take care of other clone part also. Code clone make program huge and complex. Compile and run that huge program, it increases the software and hardware requirements.

Code clone is present in the structure programming means it violet the rule of the programming language. In object oriented programming language code clone violet the inheritance concept. It leads to bad design as well as impact on maintainability of programming language.

1.3 Code Clone Detecting Techniques:

Roy and cordy (Roy C.K, et al., 2008), classified the code clone detection techniques into four categories textual, lexical, syntactic, and semantic.

1.3.1 Textual Approaches:

The Textual approach uses no transformation of the source code before applying the comparison, and source code is used directly in the clone detecting process. Though text based approach is the efficient technique but it can detect type1 clone only (Surbhi Sonika, et al., 2014). This approach cannot be assured because it cannot detect the structural type of clones having different coding but same logic.
Johnson (Johnson, et al., 1993, Johnson, et al., 1994) adopted text based approach and uses “fingerprints” on substrings of source code. Roy and cordy(Roy C.K. et al., 2008, Roy C.K. et al., 2009) also adopted the text-based approach but in their work but exploits the benefits of tree-based structural analysis based on lightweight parsing to implement flexible pretty-printing, code normalization, source transformation and code filtering.

### 1.3.2 Lexical Approaches:

Token Based Approach needs a parser or lexical analyzer to normalize the code in the form of the tokens.

In Baker’s tool Dup(Baker.B, et al., 1995), lines of source files are first divided into tokens by a lexical analyzer. Tokens are split into parameter tokens (identifiers and literals) and non-parameter tokens, with the non-parameter tokens of a line summarized using a hashing function, and the parameter tokens are encoded using a position index for their occurrence in the line. This technique is more efficient as compared to the text based approach if blank spaces and comments are present in the source code. But it is not accurate because while conversion of source code in the token sequence various false positive may introduce in the code. Various tools are proposed for clone detection that is based on the token based approach. (Toshihiro Kamiah et al., 2012) proposed a token based approach tool named CCFinder but this technique requires a parser to transform the code into tokens.

### 1.3.3 Abstract Syntax Tree Based:

**Comparison:**

Abstract syntax tree based approach converts source code into an abstract syntax tree and traverse the tree for finding a similar sub tree. If similarity is found then the code for this sub tree is termed as clone (Yu, et al., 2008). The result obtained through this comparison is quite efficient but it is very difficult and complex to create an abstract syntax tree and the scalability is also not good.

(Funaro et al., 2010) proposed a hybrid technique using Abstract Syntax Tree (AST) to identify clone candidates and textual methods to discard false positives.

### 1.3.4 Program Dependency Graph:

**Comparison:**

Program Dependence Graph shows control flow and data dependencies. Once the PDG is obtained from the source code, an isomorphic graph comparison (Koschke, et al., 2006) is applied to find the clones, and the original code slices represented by a sub graph which are returned as a clone. This approach is more efficient because they detect both semantic and syntactic clones. But the drawback with this approach is that for large software it is very complex to obtain the program dependence graph and the cost is also very high.

### 1.3.5 Metric Based Comparison:

This approach calculates the metrics from source code and uses these metrics to measure clones in software. Rather than working on source code directly this approach use metrics to detect the clones (Anil Kumar, et al., 2012). Many tools are available for calculating metrics of source code. Columbus is the tool which calculates metrics that are useful in detecting clones, but this tool does not work for Java programs. And the tool available for the calculation of Java code metrics is Source Monitor but the metrics provided by this tool are not so efficient in providing the result for detection of clones. Other tools that are available for calculating Java code metrics are very complex like Datirix which are designed for extending the quality of Java code (Jean, et et al., 1999). In the presented work a tool for metric based clone detection is proposed. The metrics calculated by this tool are useful for detecting clones in the Java software (Jean, et al., 1999) and it is easy to use too.

(Fabio Calefato et al., 2004) described how a semi automated approach could be used to identify cloned functions within scripting code of web applications. The approach was based on the automatic selection of potential function clones and the visual inspection of selected script functions. The results obtained from the clone analysis of four web applications showed that the semi automated approach was both effective and efficient at identifying function clones in web applications, and could be applied to prevent clone from spreading or to remove redundant scripting code.

(Stephane Ducasse et al., 2006) investigated a number of simple variants of string- based clone detection that normalize differences due to common editing operations, and assessed the quality of clone detection for very different case studies. Their results confirmed that the inexpensive clone detection technique generally achieved high recall and acceptable precision. Overzealous normalization of the code before comparison, however, could result in an unacceptable numbers of false positives.

(C. Kapser et al., 2006) presented an in-depth case study of cloning in a large software system that is in wide use, the Apache web server; they provided insights into cloning as it exists in this system, and they demonstrated techniques to manage and make effective use of the large result sets of clone detection tools. In their case study, they found several interesting types of cloning occurrences, such as “cloning hotspots”, where a
single subsystem comprising only 17% of the system code contained 38.8% of the clones. They also founded several examples of cloning behavior that were beneficial to the development of the system, in particular cloning as a way to add experimental functionality.

(Chanchal K. Roy et al., 2009) provided a qualitative comparison and evaluation of the current state-of-the-art in clone detection techniques and tools, and organized the large amount of information into a coherent conceptual framework. They began with background concepts, a generic clone detection process and an overall taxonomy of current techniques and tools. They then classified, compared and evaluated the techniques and tools in two different dimensions. Finally, they provided examples of how one might use the results of that study to choose the most appropriate clone detection tool or technique in the context of a particular set of goals and constraints.

(Robert Tibshirani et al., 2007) applied the fused lasso method to the “hot-spot” detection problem in comparative genomic hybridization (CGH) data. The CGH signal was approximated by a piecewise function that has relatively sparse areas with nonzero values. Hence, the method was useful for determining which areas of the signal were likely to be nonzero.

(Mohammed Abdul Bari et al., 2011) discussed the concept of code cloning, presented overcall taxonomy of current techniques and tools, and classified evolution tools in two different format as static code clone and dynamic code cloning, that together presented with program analysis, secondly as a solution the static code was divided into four parts as T1, T2, T3, T4, to finally develop a process to detect and remove code cloning.

1.4 Clone detecting tools:

<table>
<thead>
<tr>
<th>Tools Name</th>
<th>Approach</th>
<th>Supporting Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dup</td>
<td>Text Based</td>
<td>C, C++, Java</td>
</tr>
<tr>
<td>Clones</td>
<td>Token/Suffix tree</td>
<td>C, C++, Java, VB</td>
</tr>
<tr>
<td>cscope</td>
<td>Token/Suffix tree</td>
<td>C, C++</td>
</tr>
<tr>
<td>JPlag</td>
<td>Token</td>
<td>C, C++, Java</td>
</tr>
<tr>
<td>CloneDr</td>
<td>AST/ Tree matching</td>
<td>C, C++, JAVA, COBOL</td>
</tr>
<tr>
<td>Konto’s Tool</td>
<td>Comparison of metrics values and dynamic programming approach</td>
<td>C</td>
</tr>
</tbody>
</table>

2. Conclusion:

Clone detection is an very interesting area and the lot of research work is going on to detect code clone and removing from the code, In this paper , a survey on the area of clone types, detecting method and tools and also discussed open issues is for further research . This study will help user to choose the avenues for research.

REFERENCES


Liu, C., Chen, J. Han, P.S. Yu, 2006. GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis, Conf. On Knowledge Discovery and Data Mining, 872-881.


