Detection and Behavior Identification of Higher-Level Clones in Software

Swarupa S. Bongale, Prof. K. B. Manwade

D. Y. Patil College of Engg. & Tech., Shivaji University Kolhapur, India
Ashokrao Mane Group of Institutions, Vathar tarf Vadgaon, Shivaji University, Kolhapur, India

Abstract

A clone is called similar code patterns or similar code fragments occur in software systems. Software product consists through software life cycle process. Code cloning creates problem during the maintenance phase of software development process. It effects on software code size, cost and implementation time. Propose a technique that includes: 1) Detecting higher-level similarities code patterns. 2) Identifying the clone behaviour. 3) Generate clone report. 4) Performing analytical study to measure the precision and recall of the technique.

1. Introduction

Software life cycle consist of different phases, maintenance phase play an important role because software maintenance cost contributes total development cost. The working efficiency of the software reflects its quality and strength, but the actual skelton of software is its written code. This research basically focuses over the clone components of software. Similar program structures are called code clones, commonly found in software systems. Software clones may increase or decrease the cost, size and complexity of software maintenance. Cloning is active area of research, with multiple clone detection techniques has been proposed in the literature [3], [4], [1], [6]. Duplication may complicate the changes in software. Any missing can leads to update. Existing researches suggest that the code clone or duplicated code is one of the main factors that degrades the design and the structure of software and lowers the software quality such as readability, changeability and maintainability. Recent research has provided evidence that it may not always be practical, feasible, or cost-effective to eliminate certain clone groups. Copying and pasting source code is common practice, also known as software reuse. When programmers copy, paste, and then modify source code, the once-identical code fragments (code clones) can become indistinguishable as the software evolves over time. It is believed that identical or similar code fragments in source code, also known as code clones, have an impact on software maintenance. The limitation of considering only simple clones is known in the field [7]. Some clone detection tools are reported to simple clone in a huge number of ways. Another way is to detect clones of larger granularity than simple clones [1], [7].

Clone behavior, is the behavior of found clone instances. Detecting clones and identifying clones behavior helps in reducing the source code as well as to remove the unnecessary clones.

2. Detection and Behavior Identification Process of Clones

A data mining technique, pattern mining algorithm used to detect code clone. As an input give a single source file in .txt format or give a folder that contains multiple source files in .txt formats. We can give input file in c, cpp or java language.

Following, figure1 shows clone detection and behavior identification process.

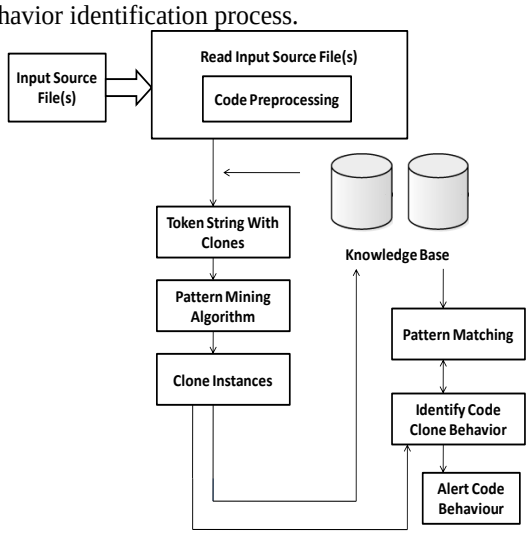


Figure1: clone detection and behavior identification process.

For whole processing, as an input here given, a single Student Report System Project source file. Following is the procedure to find clones and their behaviors:

Step 1: Perform the Code Pre-processing.
Step 2: Find out Token String with Clones.
Step 3: Apply Pattern Mining algorithm to find out repeated code patterns.
Step 4: Identify clone instance behavior.

1) Code Preprocessing:

As an input given a single source file or folder is read. Apply a simple tokenization scheme, reference [9], a single large token string is generated from the input source file(s). Here, propose a customizable tokenization strategy. In this scheme, a separate integer ID is assigned to each token found in the source code. Figure2 shows code preprocessing.

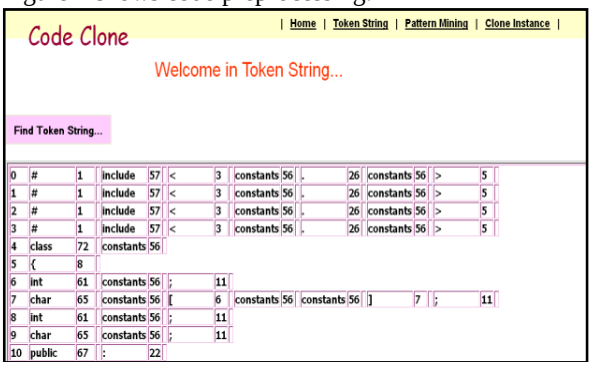


Figure2: Code Preprocessing

2) Token String with clones:

After the code preprocessing, identical segments of these id's are reported as clones. The classification of tokens is totally customizable. For example, if the user does not want to differentiate between the types {int, short, long, float, double}, we can have the different ID to represent every member of the above set of types. In this way, all those code fragments that differ only in the type of certain variables become exact replicas of each other in the token string. Figure3 shows repeated token ids.

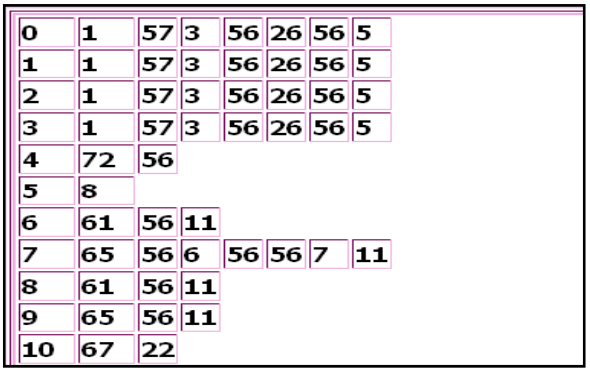


Figure3: Token String with clones

3) Pattern Mining:

Pattern mining is a naive approach is to discover repetitive patterns in the input. However, there can be many repetitive patterns discovered and a pattern can be embedded in another pattern. We detect every consecutive repetitive pattern and merge them (by deleting all occurrences except for the first one) from small length to large length. It shows, repeating line numbers and their related pattern only once. Also shown count of which pattern is how many times repeated.

Pattern mining algorithm: Given: Input source file(s).
Step1: Computes possible pattern length and return maximum pattern length for all patterns in the list.
Step2: Starting from smallest pattern length that looks for first pattern in the list.
Step3: Starting pattern compare with next occurrence of pattern, if match founds returns true.
Step4: The algorithm continues to find more matches of patterns until the end of the list has encountered.
Step5: If a pattern is detected, the algorithm modifies the list by deleting all occurrences of the pattern except for the first one.
Step6: Finally, recomputed the possible pattern length for each pattern in the modified list, reinitializes the variables to be ready for a new repetitive pattern and continues the comparisons for any repetitive patterns in the given list of patterns.

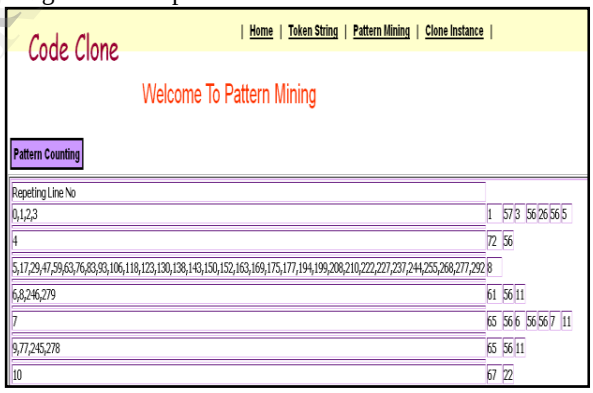


Figure4: Pattern Mining Process

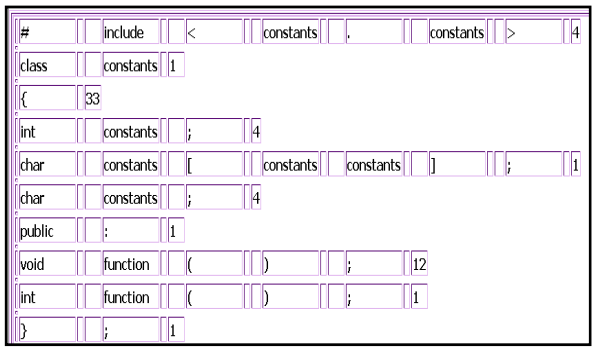


Figure5: Repeated Line Count

4) Clone Instance:

A clone relation holds between two code portions if they are the same sequences. For a given clone relation, a pair of code portions is called clone pair if the clone relation holds between the portions. An equivalence class of clone relation is called clone class. That is, a clone class is a maximal set of code portions in which a clone relation holds between any pair of code portions.

The found clone instances from input source file(s) are highlighted in different color. Figure6 shows highlighted clone instances in different color.

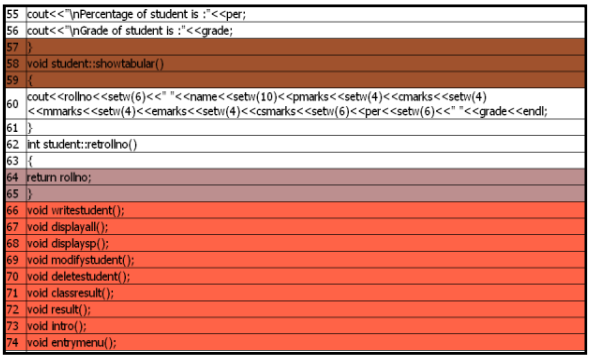


Figure6: Higher-Level Similarity Clones

5) Clone Behavior:

Once found the similar code patterns identify behavior of it. Behavior identification is useful to understand what types of patterns are repeated in given input file. So it makes easy to reduce the code size.

There are different programming structures in programming language like classes, functions, structures, control statements, file operations, input output statements and so on. Here, match the patterns of these programming structures to identify the code clone instance behavior. For example if patterns contains cout or cin , printf or scanf statements then show the behavior as input output statements.

We are matching the following patterns shows code clone behavior in the software:

- Class
- Function
- Structure
- Opening and Closing Brackets
- Header Files
- Variable Declaration
- Contains if or else statements
- Input output statements
- Looping Statements
- File Operations
- Access Specifiers
- Graphics Functions
- Clear screen
- Arithmetic operations
- Try and catch block
- Go to X and Y
- Case break in switch
- Ending of program

Clones Behaviors for Student Report System project file are shown as bellows:

```
207 : while(inFile.read((char *) &st, sizeof(student)))
208 : {
236 : while(inFile.read((char *) &st, sizeof(student)))
237 : {
```

Opening And Closing Bracket

```
0 : #include<fstream.h>
1 : #include<iomanip.h>
2 : #include<stdio.h>
3 : #include<conio.h>
```

Header Files

```
20 : grade=A;
21 : else if(per>=50)
22 : grade=B;
23 : else if(per>=33)
24 : grade=C;
```

Arithmetic Operation

```
244 : {
245 : char ch;
246 : int rno;
278 : char ch;
279 : int num;
76 :  {
77 :  char ch;
```

Variable Declaration

```
136 : }
137 : void displaysp(int n)
138 : {
160 : getch();
161 : }
162 : void modifystudent(int n)
163 : {
```

Opening And Closing Bracket

```
27 : }
28 : void student::getdata()
29 : {
45 : }
46 : void student::showdata()
47 : {
57 : }
```

Opening And Closing Bracket

```
50 : cout<<"\nMarks in Physics : "<<pmarks;
51 : cout<<"\nMarks in Chemistry : "<<cmarks;
52 : cout<<"\nMarks in Maths : "<<mmarks;
53 : cout<<"\nMarks in English : "<<emarks;
54 : cout<<"\nMarks in Computer Science :"<<csmarks;
```

Input Output statements

```
157 : inFile.close();
158 : if(flag==0)
185 : found=1;
186 : }
187 : }
188 : File.close();
189 : if(found==0)
```

Contains if Or Else Statement

```
177 : {
178 : st.showdata();
207 : while(inFile.read((char *) &st, sizeof(student)))
208 : {
236 : while(inFile.read((char *) &st, sizeof(student)))
237 : {
238 : st.showtabular();
```

Function

```
34 : cout<<"\nEnter The marks in physics out of 100 : ";
35 : cin>>pmarks;
36 : cout<<"\nEnter The marks in chemistry out of 100 : ";
37 : cin>>cmarks;
38 : cout<<"\nEnter The marks in maths out of 100 : ";
39 : cin>>mmarks;
40 : cout<<"\nEnter The marks in english out of 100 : ";
```

Input Output statements

```
243 : void result()
244 : {
245 : char ch;
246 : int rno;
262 : case 3 :
263 : break;
264 : default:cout<<"\a";
```

Variable Declaration

```
11 : void getdata();
12 : void showdata();
13 : void showTabular();
66 : void writestudent();
67 : void displayall();
68 : void displaysp();
69 : void modifystudent();
```

Function

```
100 : default :cout<<"\a";
101 : }
252 : cin>>ch;
253 : clrscr();
254 : switch(ch)
255 : {
256 : case 1 :classresult();
```

Input Output statements

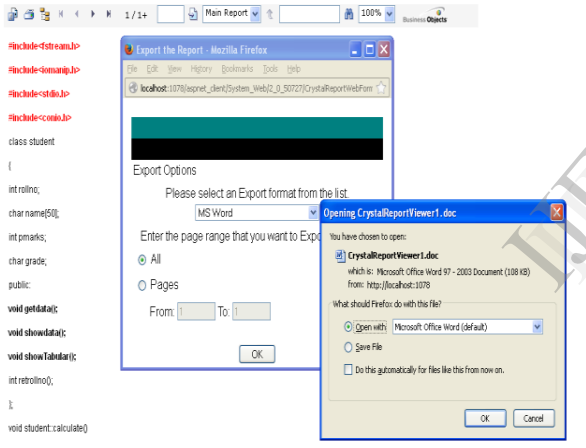
```
104 : }
105 : void writestudent()
106 : {
107 : student st;
108 : ofstream outFile;
115 : getch();
116 : }
```

Opening And Closing Bracket

It shows that, Student Report System project contains 14 behaviors of different types. For example, block3 contains arithmetic operator (=), showing behavior: Arithmetic Operations, block2 contains #include statements, showing behavior: Header Files, block4 contains variables, showing behavior: Variable Declaration, block12 contains functions, showing behavior: Function, block8 contains if statement, showing behavior: Contains if or else statement.

3. Clone Report

It generates a clone report which shows highlighted color clone instances available in a project and saves a project file in a different file format like MS-Word, pdf, rtf etc. When a user wants to reopen this saved file, he/she easily found that similar code fragments in a given input project file. So, no need to run the project every time to find similar code patterns. Following figure shows clone report of example, Student Report System project.




```
per=(p_marks+cmarks+mmarks+emarks+csmarks)/5.0;
if(per>=60)
grade=A;
else if(per>=50)
grade=B;
else if(per>=33)
grade=C;
else
grade=F;
}
void student::getdata()
{
cout<<"\nEnter The roll number of student ";
cin>>rollno;
cout<<"\n\nEnter The Name of student ";
gets(name);
```

4. Experimental Results

The experiments are done on different input files. Precision and recall are the two basic measures used to calculate the result accuracy of the system. Precision denotes the probability that a randomly chosen candidate clone group is relevant. Recall denotes the probability that a relevant clone group, chosen from the hypothetical set of all relevant clone groups, is contained in a detection result. We calculate the precision and recall in terms of single input file and multiple input files.

Clone Detection Result: Our System founds all higher-level similarity clones. So, precision is 1 and recall is 1, in case of clone detection.

Clone Behavior Result: Here found the system generated total number of clone behaviours and out of them correct number of clone behaviours. From that calculate the precision and recall.

Table1: Clone behavior results of single input file.

Sr No	Input File Name	Language	Number of Tokens	Precision	Recall
1	Student Record System	C	2655	1	0.94
2	Snake Game	C	1474	1	1
3	Telephone Billing System	Cpp	3460	1	0.90
4	Supermarket	Cpp	2244	1	1
5	Address Book	Java	4905	1	0.88

Figure7 shows behavior graph for single input file.

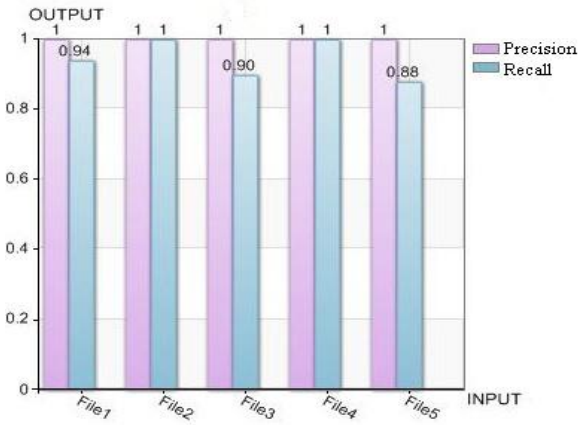


Figure7: Single file behaviors graph

Table2: Clone behavior results of multiple input files.

Sr No	Input Folder Name	Language	Number of files included	Number of Tokens	Precision	Recall
1	Library Management	C	6	12265	1	0.90
2	Department Store	C	4	5129	1	0.85
3	Video Store System	Cpp	4	6584	1	0.85
4	Student Report System	Cpp	5	4482	1	1
5	Rapid Roll Game	Java	7	5750	1	0.95

Figure8 shows behavior graph for multiple input files.

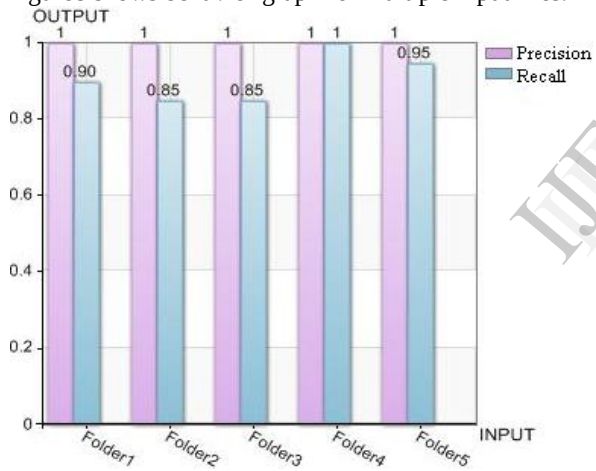


Figure8: Multiple files behaviors graph

5. Conclusion

Cloning is active area of research in software development process. A software development process is a creativity of software through different phases. In software development process, maintenance phase play an important role because maintenance cost contributes total development cost. Software reuse reduces software development and maintenance costs in the process of creating software systems. Reusable modules reduce the implementation time. The use of existing components is done basically with the activity of copy and paste. Cloning is the unnecessary duplication of data whether it is at design level or at coding level. Software clones may increase or decrease the cost, size and complexity of software maintenance. Clone detection and their behavior identification are useful to reduce total software development cost and software implementation time. In future, try to reduce the code size by removing unnecessary clones. Clone detection and clone behavior identification is useful in code optimization.

6. References

[1] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: Multi-Linguistic Token-Based Code Clone Detection System for Large Scale Source Code", IEEE Trans. Software Eng.,vol. 28,no. 7,pp. 654-670, July 2002.

[2] Cory J. Kapser and Michael W. Godfrey, "Cloning Considered Harmful" Considered Harmful: Patterns of Cloning in Software", Software Architecture Group (SWAG) avid R. Cheriton School of Computer Science, University of Waterloo.

[3] B.S. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems," Proc. Second Working Conf. Reverse Eng.,pp. 86-95, 1995.

[4] I.D. Baxter, A. Yahin, L. Moura, M.S. Anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees," Proc. IEEE Int'l Conf. Software Maintenance, pp. 368-377, 1998.

[5] R. Koschke, R. Falke, and P. Frenzel, "Clone Detection Using Abstract Syntax Suffix Trees," Proc. 13th Working Conf. Reverse Eng., pp. 253-262, 2006.

[6] A. Walenstein, A. Lakhotia, and R. Koschke, "The Second International Workshop Detection of Software Clones: Workshop Report," SIGSOFT Software Eng. Notes, vol. 29, no. 2, pp. 1-5, Mar.2004.

[7] A. De Lucia, G. Scanniello, and G. Tortora, "Identifying Clones in Dynamic Web Sites Using Similarity Thresholds," Proc. Int'l Conf. Enterprise Information Systems, pp. 391-396, 2004.

[8] G. Grahne and J. Zhu, "Efficiently Using Prefix-Trees in Mining Frequent Itemsets," Proc. First IEEE ICDM Workshop Frequent Itemset Mining Implementations, Nov. 2003.

[9] Swarupa S. Bongale, Prof. K.B.Manwade, Prof. G.A.Patil, "An Efficient Data Mining Approach for Complex Clone Detection in Software", International Journal of Advanced Research in Computer Science and Software Engineering, volume 3 issue 5, ISSN: 2277 128X May 2013.