# **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 costeffective 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.

	Code Clone   Home   Token String   Pattern Mining   Clone Instance														
	Welcome in Token String														
Fir	Find Token String														
0	#	1	include	57	<	3	constants	56		26	constants	56	>	5	Г
1	#	1	include	57	<	3	constants	56		26	constants	56	>	5	T
2	#	1	include	57	<	3	constants	56		26	constants	56	>	5	T
3	#	1	include	57	<	3	constants	56		26	constants	56	>	5	T
4	class	72	constants	56											
5	{	8	[												
6	int	61	constants	56	;	11									
7	char	65	constants	56	[	6	constants	56	constants	56	]	7	;	11	i
8	int	61	constants	56	;	11									
9	char	65	constants	56	;	11									
10	public	67	:	22											

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.

0	1	57	3	56	26	56	5
1	1	57	3	56	26	56	5
2	1	57	3	56	26	56	5
3	1	57	3	56	26	56	5
4	72	56					
5	8						
6	61	56	11				
7	65	56	6	56	56	7	11
8	61	56	11				
9	65	56	11				
10	67	22					

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.

Code Clone   Home   Token String   Pattern Mining   Clone Instance	<u>e</u>
Welcame To Dettern Mining	
vveicome to Pattern Mining	
Pattern Counting	
Repeting Line No	
0,1,2,3	1 57 3 56 26 56 5
4	72 56
5,17,29,47,59,63,76,83,93,106,118,123,130,138,143,150,152,163,169,175,177,194,199,208,210,222,227,237,244,255,268,277,29	28
6,8,246,279	61 56 11
7	65 56 6 56 56 7 11
9,77,245,278	65 56 11
10	67 22

Figure4: Pattern Mining Process

#	include < constants ,	constants > 4
class	constants 1	
{	33	
int	constants; 4	
char	constants [ constants consta	nts ] ; 1
char	constants; 4	
public	: 1	
void	function ( );	12
int	function ( );	1
}	; 1	

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.

	Code Clone	<u>Home</u>   <u>Token String</u>   <u>Pattern Mining</u>   <u>Clone Instance</u>
	Welcome	e in Program., View Program Repeated Code Patterns Clone Behavior Report
pos	osprogline	
0	) #include <fstream.h></fstream.h>	
1	#include <iomanip.h></iomanip.h>	
2	#include <stdio.h></stdio.h>	
5	#Include <conio.n></conio.n>	
-+ -		
6	int rollino:	
7	char name[50]:	
8	3 int pmarks:	
9	char grade;	
10	0 public:	
11	1 void getdata();	
12	2 void showdata();	
13	3 void showTabular();	
14	4 int retrolino();	
15	5 };	
16	.6 void student::calculate()	
17	7 { 2	due : concerdent/C 0:
18	s per=(p_marks+cmarks+mmarks+email 9 if(per>=60)	rks+csmarks)/5.0;
20	20 grade=A:	
21	1 else if(per>=50)	
22	2 grade=B;	
23	13 else if(per>=33)	
24	/4 grade=C;	
25	25 else	
27	7 }	
28	8 void student::getdata()	
29	9 {	
30	0 cout<<"\nEnter The roll number of st	udent ";
31	1 cin>>rolino;	ne ".
33	2 cout<< (mineriter the name of stude) (3 dets(name);	к ;
24	4 sout < "InEnter The marke in physics	out of 100 . ".
35	<ul> <li>soucce granter memarks in physics</li> <li>icin&gt;&gt;pmarks;</li> </ul>	
36	6 cout<<"\nEnter The marks in chemist	y out of 100 : ";
37	7 cin>>cmarks;	
38	8 cout<<"\nEnter The marks in maths o	ut of 100 : ";
39 40	o cont<<"\nEnter The marks in english	out of 100 : ":
41	1 cin>>emarks;	
42	2 cout<<"\nEnter The marks in compute	r science out of 100 : ";
43	3 cin>>csmarks;	
44 45	+ calculate();	
46	6 void student::showdata()	
47	7 {	
48	8 cout<<"\nRoll number of student : "<	<rolino;< th=""></rolino;<>
49	cout<< \nivame of student : "< <name 0 cout&lt;&lt; \nivame of student : "&lt;<name 0 cout&lt;&lt; \nivame of student : "&lt;<name 0 cout&lt;</name </name </name 	z; ke
51	1 cout<<"\nMarks in Chemistry : "< <m< th=""><th>narks;</th></m<>	narks;
52	2 cout<<"\nMarks in Maths : "< <mmarks< th=""><th>(\$;</th></mmarks<>	(\$;
53	3 cout<<"\nMarks in English : "< <emar< th=""><th>ks;</th></emar<>	ks;
54	H Icout<<"\nMarks in Computer Science	: < <csmarks;< th=""></csmarks;<>



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:

No. 1.1       Status       Opengiad Case back       Status	207 : while(inFile.read((char *) &st, sizeof(student)))	]	157 : inFile.close();			
30. additional and (rate = 1 add, (rate = 1) add, (rate = 1) add)         opening add (rate = 1) add (rate =	208 : {		 158 : if(flag==0)			
Dist         Opening and Chaing Bodet         Dist         D	236 : while(inFile.read((char *) &st, sizeof(student)))		185 : found=1:			
1         gendesic         1         gendesic         1         file	237 : (	Opening And Closing Bracket	186 · 1			
1         1			107 . )	Contains if Or Else Statement		
1         10 closed (1)			187 : }			
1         1			188 : File.close();			
1         Holder Green is D         P1			189 : if(found==0)			
1         Hitchieristandy 20         Filteristandy 20	0 : #include <fstream.h> </fstream.h>		177 : {			
1         file/bit/stdfin         File/bit/stdfin </td <td>1 : #include<iomanip.h></iomanip.h></td> <td></td> <td>178 : st.showdata(); </td> <td></td>	1 : #include <iomanip.h></iomanip.h>		178 : st.showdata(); 			
Bender Files         Part of the file	2 : #include <stdio.h></stdio.h>		<pre>207 : while(inFile.read((char *) &amp;st, sizeof(student)))</pre>			
1         1000000000000000000000000000000000000		Header Files	208 : {	Function		
1         1	3 : #include <conto.h></conto.h>		<pre>236 : while(inFile.read((char *) &amp;st, sizeof(student)))</pre>			
201       gradesh.         21.       gradesh. <td></td> <td></td> <td>237 : {</td> <td>_</td>			237 : {	_		
Bit Control (1)         Bit Contro			238 : st.showtabular();	·		
01         62<		-	34 : cout< "\nEnter The marks in physics out of 100 : ";</td <td></td>			
1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1	20 : grade=a;		35 : cin>>pmarks;			
2: : : : : : : : : : : : : : : : : : :	21 : else if(per>=50)		36 : cout<<"\nEnter The marks in chemistry out of 100 : ";			
1: tabe if (per=23)       Arthmetic Operation       3: cost ("Matter the watks is nation of of 00: 0: 1","       1         2: cost=0: ("Matter the watks is nation of of 00: 0: 1","       9: cost ("Matter the watks is nation of of 00: 0: 1","       1         2: cost=0: ("Matter the watks is nation of of 00: 0: 1","       9: cost ("Matter the watks is nation of 00: 0: 1","       1         2: cost=0: ("Matter the watks is nation of 00: 0: 1","       9: cost ("Matter the watks is nation of 00: 0: 1","       1         2: cost=0: ("Matter the watks is nation of 00: 0: 1","       9: cost ("Matter the watks is nation of 00: 0: 1","       1         2: cost=0: ("Matter the watks is nation of 00: 0: 1","       10: cost ("Matter the watks is nation of 00: 0: 1","       1         2: cost=0: ("Matter the watks is nation of 00: 0: 1","       10: cost ("Matter the watks is nation of 0: 0: 0: 1","       1         2: cost=0: ("Matter the watks is nation of 0: 0: 0: 1","       10: cost ("Matter the watks is nation of 0: 0: 0: 1","       1         2: cost=0: ("Matter the watks is nation of 0: 0: 0: 0: 0: 0: 0: 0: 0: 0: 0: 0: 0:	22 : grade=B;		37 : cin>>cmarks;	Input Output statements		
H = grade-C;       32 : : : : : : : : : : : : : : : : : : :	23 : else if(per>=33)	Arithmetic Operation	38 : cout<<"\nEnter The marks in maths out of 100 : ";			
01 : cost <	24 : grade=C;		39 : cin>>mmarks;			
24 1 [       24 1 [			40 : cout<<"\nEnter The marks in english out of 100 : ";			
24: (       44.7 (         35: char ch;       56: char ch;         24: i far two;       56: char ch;         27: char ch;       57: char ch;         27: char ch;       56: char ch;         28: char ch;       56: char ch;         29: char ch;       56: char ch;         21: char ch;       56: char ch;         21: char ch;       57: char ch;         21: char ch;       57: char ch;         21: char ch;       58: char ch;         2			243 : void result()			
25 : char ch:       26 : char ch:       27 : char ch:       28 : char ch:<	244 : {		244 : {			
24: int row;       24: int row; <td< td=""><td></td><td></td><td>245 : char ch;</td><td></td></td<>			245 : char ch;			
21       1.1       202       c.ase 3       202<	216 · int mo:		246 : int rno;	Variable Dedenation		
21 - 1       23 - 1 + max       24 - 1 + max       25 - 1 + max       25 - 1 + max       26 - 1 + max       26 - 1 + max       27 + max       28 - 1 +			 262 : case 3 :	variable Declaration		
279 int mm;       244 defailt cont <	2/8 : cnar cn;	Variable Declaration	 263 : break:			
7::       (         71::       char ch;         136::       )         137::       wold displaysp(int n)         138::       (         14::       youd displaysp(int n)         16::       youd autobackt();         16::       youd autobackt();         16::       youd student::::::::::::::::::::::::::::::::::::	279 : int num; 		264 : default:cont<<"\a":	×		
17 : char ch;       1         136 : 1       1         137 : void displaysp(int n)       1         138 : (       6         140 : gottch();       6         151 : 1       1         152 : void displaysp(int n)       1         153 : (       1         154 : (       1         155 : (       1         157 : void displaysp();       1         153 : (       1         155 : (       1         155 : (       1         155 : cont<	76 : {		11 : void getdata();			
156 : 1	77 : char ch;		12 : void showdata():			
137 : void displaysp(int n)	136 : }		13 · woid chor#shulse/\.			
138 : (       160 : getch();       161 : 1)       162 : wold molifystudent(int n)       162 : wold molifystudent(int n)       162 : wold student::getdata();       162 : wold student::getdata();       162 : wold student::getdata();       100 : default :cout<	137 : void displaysp(int n)			-		
160 : getch();       Opening And Closing Bracket       67 : void displayal1();         151 : )       68 : void displaysp();       69 : void nodifystudent();         163 : (1       100 : default :cont<	138 : {		<pre>bb : Vold Writestüdent();</pre>	Function		
161::)       68:: roid displaysp();         162:: void andifystudent(int n)       69:: roid modifystudent();         163::(       100:: default :cout<("\a";)	160 : getch();	Opening And Closing Bracket	67 : void displayall(); 			
162 : void modifystudent(int n)       69 : void modifystudent();       163         163 : {       100 : default :cout       101 : }         27 : 1       101 : }       102 : default :cout<	161 : }		68 : void displaysp(); 			
163 : {       100 : default :cont<<"\a";	162 : void modifystudent(int n)		69 : void modifystudent();			
27: }       101: }         28: void student:::getdata()       252: cim>ch;         29: {       253: cirscr();         45: }       0pening And Closing Bracket         46: void student:::showdata()       254: switch(ch)         47: {       255: {         50: cont<	163 : {		100 : default :cout<<"\a";			
28 : void student:::getdata()       252 : cin>ch;         29 : {	27 : }	^	101 : }			
29 : {       253 : clrscr();       Input Output statements         45 : ; }       0pening And Closing Bracket       253 : clrscr();       Input Output statements         46 : roid student::showdata()       255 : {       255 : {       255 : {         47 : {       256 : case 1 :classresult();       256 : case 1 :classresult();       104 : }       105 : roid writestudent()       105 : roid writestudent()       106 : {       106 : {       106 : {       107 : student st;       106 : {       107 : student st;       108 : ofstream outFile;       108 : ofstream outFile;       108 : ofstream outFile;       115 : geth();       116 : }       0pening And Closing Bracket       0pening And Closing Bracket	28 : void student::getdata()		252 : cin>>ch;			
45: }       Opening And Closing Bracket       254: switch(ch)         46: void student::showdata()       255: {         47: {       256: case 1:classresult();         57: }       104: }         50: cont<<**\mMarks in Chemistry: *	29 : {		253 : clrscr();	Input Output statements		
46 : void student::showdata()       255 : {       255 : {       255 : {       255 : {       256 : case 1 : classresult();       256 : case 1 : classresult();       257 : }       256 : cose 1 : classresult();       257 : ]       104 : }       105 : void writestudent()       105 : void writestudent()       105 : void writestudent()       106 : {       105 : void writestudent()       106 : {       107 : student st;       106 : {       107 : student st;       107 : student st;       107 : student st;       108 : ofstream outFile;       108 : ofstream outFile;       115 : getch();       115 : getch();       116 : }       116 : }       116 : }       116 : }       116 : }       116 : }       116 : }       116 : }       116 : }       116 : }       116 : }       116 : }       116 : }       116 : }       116 : ] <t< td=""><td>45 : }</td><td>Opening And Closing Bracket</td><td>254 : switch(ch)</td><td></td></t<>	45 : }	Opening And Closing Bracket	254 : switch(ch)			
47 : {       256 : case 1 : classresult();       2         57 : }       104 : }       105 : void writestudent();       105 : void writestudent();         51 : cout<<*/whatks in Chemistry : " <cmarks;< td="">       105 : void writestudent();       106 : {       107 : student st;         53 : cout&lt;&lt;*/whatks in English : "<cemarks;< td="">       107 : student st;       107 : student st;       107 : student st;       108 : ofstream outFile;       108 : ofstream outFile;       115 : getch();       116 : }       116 : }       116 : )       116 : ]       116 : ]<td>46 : void student::showdata()</td><td></td><td>255 : {</td><td></td></cemarks;<></cmarks;<>	46 : void student::showdata()		255 : {			
57 : }         50 : cout       []]         50 : cout       []]         51 : cout       []]         52 : cout       []]         53 : cout       []]         53 : cout       []]         54 : cout       []]         54 : cout       []]         54 : cout       []]         55 : []]       []]         56 : []]       []]         57 : []]       []]         58 : cout       []]         59 : []]       []]         50 : []]       []]         51 : []]       []]         52 : []]       []]         53 : []]       []]         54 : []]       []]         54 : []]       []]]         54 : []]       []]]         55 : []]       []]]         56 : []]       []]]         57 : []]       []]]         58 : []]       []]]]         59 : []]       []]]]         50 : []]       []]]]]         50 : []]       []]]]]         51 : []]]       []]]]]         52 : []]]]       []]]]]]]         53 : []]]]]       []]]]]]]]]]         54 :	47 : {		256 : case 1 :classresult();			
50 : cout       ::	57 : }		104 : }	\$		
51 : cout	50 : cout<<"\nHarks in Physics : "< <pmarks; </pmarks; 	]	105 : void writestudent()			
52 : cout       in Maths : " <mmarks;< td="">         53 : cout       input Output statements         54 : cout       input Output statements         108 : ofstream outFile;       108 : ofstream outFile;         115 : getch();       116 : )</mmarks;<>	51 : cout<<"\nHarks in Chemistry : "< <cmarks;< td=""><td></td><td>106 : {</td><td></td></cmarks;<>		106 : {			
53 : cout       Input Output statements         54 : cout       Input Output statements         108 : ofstream outFile;         115 : getch();         116 : )	52 : cout<<"\nMarks in Maths : "< <munarks;< td=""><td></td><td>107 · student st</td><td></td></munarks;<>		107 · student st			
54 : cout       108 : ofstream outrile;	53 : cout<<"\nMarks in English : "< <emarks;< td=""><td>Input Output statements</td><td>100</td><td>Opening And Closing Bracket</td></emarks;<>	Input Output statements	100	Opening And Closing Bracket		
115 : getch(); 116 : }	54 : cout<<"\nMarks in Computer Science : "< <csmarks; </csmarks; 		100 . OIStream OUTTIE;			
116 : }			115 : getch(); 			
			116 : }			

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.



### 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.

C.	Input File	Long	Number	Draai	Decell
51	input rife	Lang	Nulliber	Flect	Recall
No	Name	uage	of	sion	
$\sum_{i=1}^{n}$			Tokens		
1	Student	С	2655	1	0.94
	Record				
	System				
2	Snake	С	1474	1	1
	Game				
3	Telephone	Срр	3460	1	0.90
	Billing				
	System				
4	Supermarket	Срр	2244	1	1
	-				
5	Address	Java	4905	1	0.88
	Book				

Table1: Clone behavior results of single input file.

Figure7 shows behavior graph for single input file.



Sr	Input	Lang	Number	Number	Preci	Rec
No	Folder	uage	of files	of	sion	all
	Name		included	Tokens		
1	Library	С	6	12265	1	0.90
	Manage					
	ment					
2	Depart	С	4	5129	1	0.85
	ment					
	Store					
3	Video	Срр	4	6584	1	0.85
	Store					
	System					
4	Student	Срр	5	4482	1	1
	Report					
	System					
5	Rapid	Java	7	5750	1	0.95
	Roll					
	Game					

Table2: Clone behavior results of multiple input files.

Figure8 shows behavior graph for multiple input files.



# 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.