Detection of Non Continguous Clones in Software using Program Slicing

Er. Richa Grover[1] Er. Narender Rana[2]

M.Tech in CSE[1] Astt. Proff. In C.S.E[2]

GITM, Kurukshetra University, INDIA

Abstract

Code duplication or copying a code fragment and then reuse by pasting with or without any modifcations is a well known code smell in software maintenance. Several studies show that about 5% to 20% of a software systems can contain duplicated code,which is basically the results of copying existing code fragments and using then by pasting with or without minor modifcations. Software cloning is copying of code fragment from one source code and using it in another source code. Cloning of software code has bad impact. One of the major problem of such duplicated fragments is that if a bug is find out in a code fragment, all the other fragments similar to it should need to check. For code clone detection, many detection techniques are there with different code intermediate representation techniques. This paper proposes a clone detection technique using program slicing with matching technique. The aim of the proposed approach is detection of three types of clone i.e. type 1 (exact match), type 2 (parameterized match) type 3 (near-miss match). The technique also detects non-contiguous clones**.**

Index Terms: Dead Codes, Matching Algorithms, Program Slicing,Software Clones.

1. Introduction

Clones are the results of copy-paste activities. Copying code fragments and then reuse by pasting with or without minor changes or adaptations are common activities in software development[1]. This type of reusing of existing code is called code cloning and the pasted code fragment (with or without modifications) is called a clone of the original. However, in a post-development phase, it is diffcult to say which fragment is original and which one is copied and therefore, fragments of code which are exactly the same as or similar to each other are called code clones.This process is called software cloning.

Cloning makes software difficult to maintain and produce the errors also.It propagates the error from one code to other cloned codes and has adverse affects on life cycle of software.Thus we need to remove the clones and prevent their introduction by monitoring the source code during its evolution.If clones exists in software,then there are various techniques for detection of code clones with different types of matching algorithms. In the paper, program slicing and comparison technique is utilized for detection of code clones. Program slicing is used to obtain an intermediate representation of the source code and then matching algorithm is applied among the obtained slices. Although program slicing is used for debugging a program, software maintenance, optimization, program analysis, information flow relations. Program slicing can also be used for software clone detection and it is very effective in detecting software clones because it detects reordered, non-contiguous clones and intertwined clones and it gives more precise result when used for code cloning. Program slice is an independent part of the program which does not affect the behavior of the remaining program. Therefore, program slicing removes irrelevant part of program and it gives better result. There are many clone detection approaches listed in researches like text-based, token-based, tree-based, graph-based, metric-based and hybrid based techniques which are widely used for code clone detection[1]. For each clone detection approach some tools are proposed and designed by various researchers. Here we are using token based clone detection matching technique for comparison of tokens obtained from program slices. Token based clone detection approach takes source code and converts them in lexemes/tokens. From sequence of tokens, token stream are formed. The heart of token

based matching approach is how to use syntax tree and syntax array. Some famous out of these tools are dup[6],[7] that uses token sequence and used them as syntax tree, CCFinder[6],[8] uses suffix tree matching techniques, CP-Miner[7],[9] uses frequent item-set mining, Koschke et al's tool [9] is based on parser and generated abstract syntax tree, Li and Thompson's tool[10] is based on AST and uses merging of token sequences and AST. Software clones are of four type's viz. Type 1, type 2, type 3, type 4 and non-contiguous clones. Non-contiguous code clones are clone in which source code is not sequentially arranged and is re-ordering of source code [10]-[12]. Therefore, non-contiguous code clones are not easily detected with simple code clone detection techniques. Program slicing is a way of detecting non-contiguous code clones.Remaining of this paper is organized as follows: Section 2 describes background about software clone detection, types of clone and applications of code clone detection. Section 3 describes program slicing algorithm about how program slices is extracted. Section 4 gives match detection technique and section 5 gives results and setion 6 gives future work and conclusion of this paper.

2. Background

In software development process, cloning of software code is becoming common in these days. Copying existing code fragments from a section of code and pasting it into another section of code is called code cloning. Cloning is a type of duplicity of an original form. But in post development phase, it is very difficult to find out which code fragment is original and which one is copied code fragment. The copied code is called a software clone and the process is called as software cloning [1],[2]. However software cloning is sometimes useful for developers because it may reduce the time for development. Code cloning is considered as a bad practice in software development process. Code cloning is not only difficult to maintain but also produces subtle errors. The surveys in the field of software clone detection show that the research in this field is going on increasing day by day [1]. But still there is no precise definition about code clone. Every research has its own definition but not a specific. In almost all the software, code cloning is done because with exact clones: it shrinks 14% the size of code fragment and with parameterized clones: it shrinks 61% of the code clone [1],[2]. About 20-30% of large software system consists of cloned code. Mostly code cloning happens due to open source software. In open source software, source code is provided; due to this any developer can easily use copy-paste and code cloning. With availability of source code, developers can easily cloning the functionality of code which is not textually similar.

Software clones:

According to different research studies, there are four basic types of clones. One study [1] shows that two types of similarities between two code fragments. Similarity of two code fragments is based on their similarity of program text and similarity of functionality. First three types are based on textual similarity and type four is based on functional similarity. The complexity of detecting software clones is going on increasing as we are going through from type 1 to type 4 clones.

Textual Similarity: Based on the textual similarity we distinguish the following types of clones:

Type I:Identical code fragments except for variations in whitespace (may be also variations in layout) and comments.

Type II:Structurally/syntactically identical fragments except for variations in identifiers, literals, types, layout and comments.

TypeIII:Copied fragments with further modifications. Statements can be changed,added or removed in addition to variations in identifiers, literals, types, layout and comments.

Functional Similarity: If the functionalities of the two code fragments are identical or similar i.e., they have similar pre and post conditions, we call them semantic clones and referred as *Type IV* clones.

Type IV: Two or more code fragments that perform the same computation but implemented through different syntactic variants.

There are various application and advantages of clones detection.We list some of those as follows:

Detects library candidate: code fragment that has been copied and reused multiple times in the system, then this fragment can be stored in a library, to announce its reuse potential officially.

Helps in program understanding: If the functionality of a cloned fragment is comprehended, it is possible to have an overall idea on the other files containing other similar copies of that fragments .

Helps aspect mining research: Detecting similar code is also required in aspect mining for detecting cross-cutting concerns. The code of cross-cutting concerns is typically duplicated over the entire application and could be identified with clone detection tools.

Finds usage patterns: If all the cloned fragments of a same source fragment can be detected, the functional usage patterns of that fragment can be discovered .

Detects malicious software: Clone detection techniques can be used in finding malicious software. By comparing one malicious software family to another, it is possible to find the evidence where parts of one software system match parts of another.

Detects plagiarism and copyright: Finding similar code may also useful in detecting plagiarism and copyright infringement.

Helps software evolution research: Clone detection techniques are successfully usedin software evolution analysis by looking at the dynamic nature of different clones in different version of system.

3. Program Slicing

For program slicing various techniques are given by researchers[4],[5].These techniques use different methods for data flow and control flow [3],[4]. Program slicing is a technique for aiding debugging and program comprehension by reducing complexity. The essence of program slicing is to remove statements from a program that do not affect the values of variables at a point of interest. There are many ways to do this (both variants of program slicing and different algorithms for achieving the same result).In this section we will give an overview of slicing.Program slice is a reduced program. Weiser stated in [5] that this reduced program should be executable. In the case that the program slice is executable then when the original program and the slice are executed the variables specified in the criterion should have the same values at the point of interest.The slicing criterion is denoted by (S, V) where 'S' is the statement or line number and 'V' is the variable in the program .Frank Tip states that slice is itself an executable program subset of the program whose behavior must be identical to the specified subset of the original program's[4]. We have used program slicing technique to identify the code clones in a program. Code clones are to be detected from the program source code. It may not be always practical to check the whole program which may contain thousands of lines of code to find the presence of code clones. In several situations, the program tester will be interested only in particular parts or function of the source program which is supposed to perform certain important tasks. In such a scenario, it is not advisable to analyze the program lines one by one as this will only cause unnecessary waste of time and effort. Program slicing is applied in such situations. Instead of analyzing the whole program, slicing converging the focus to some specific program parts. Sliced statements give the variable dependencies present in the program and also eliminate the need for unnecessary checking of the whole program.We have various categories for program slices On the basis of program behavior, program slicing can be static or dynamic slicing. On the basis of direction of flow in programs, program slicing is of two types, i.e., backward and forward slicing. On the basis of level of slicing, program slicing can be intraprocedural or interprocedural slicing.We have various steps for extracting the slices.

3.1 Steps for program slicing:

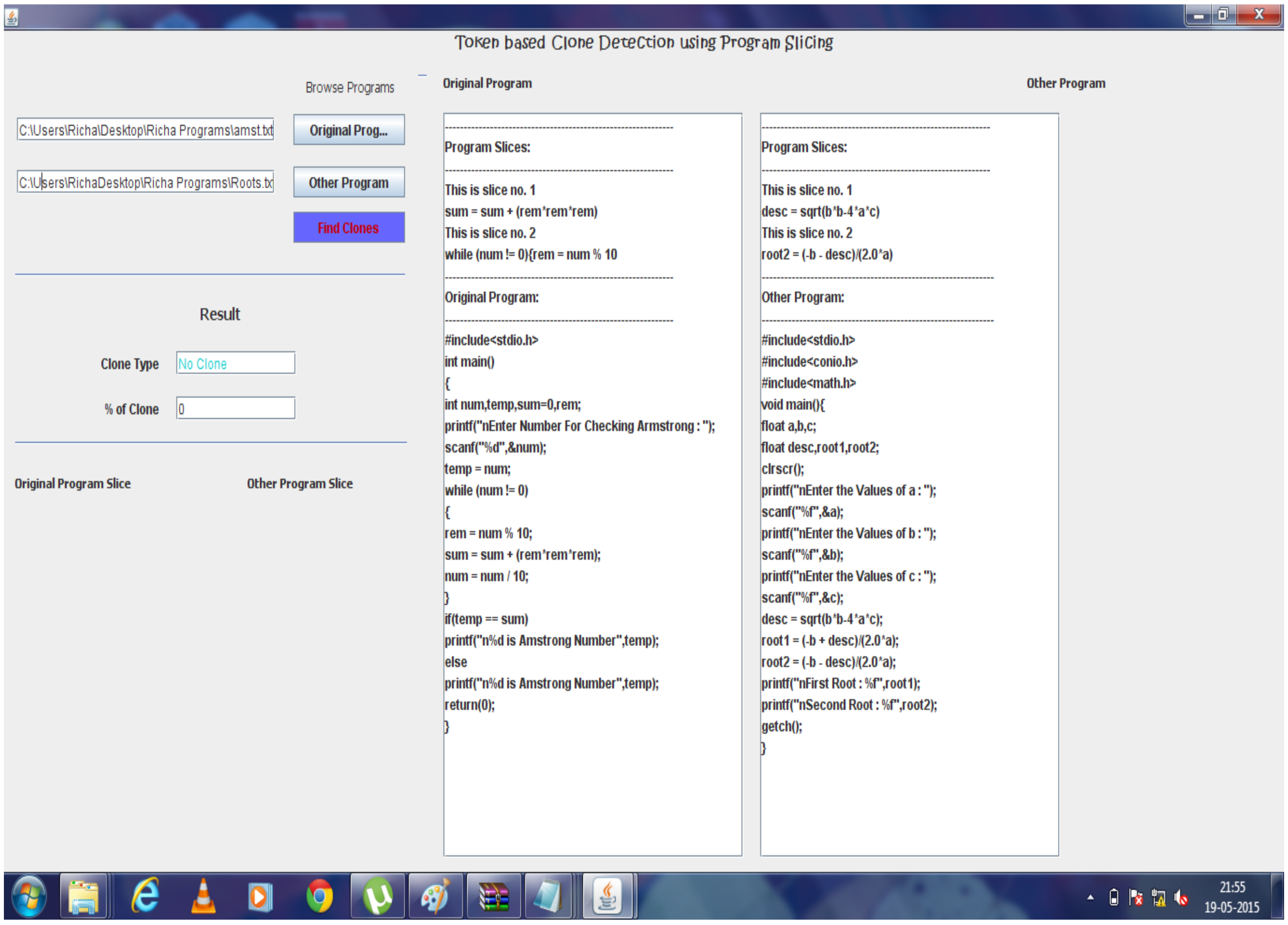
Firstly, source code is taken as input and all variable used in program are stored in a variable list. After this, program sections are found out by scanning the whole program, program sections i.e. conditional statements, loops and function and store them in a table. Then, step 3 and step 4 gives program slices with respect to control flow and data flow sequence. For step 3, Section can be tagged by making list with section name and starting and ending line in code. The list which is generated from above execution will have starting and ending line of every section. By using this list we can find the slice from that section only with boundary of starting and ending line. The lines of source code belonging to this section may also contain section as found in previous step. So there is need to check every section for tracing subsection. To do so, before proceeding further, we will repeat step similar to Step 2, 3, 4 and 5 until we reach to bottom of the source code tree. For dependency check, a variable is chosen from variable list and a check is made for program section i.e. for functions, loops. If match is found i.e. whole section is dependent on that variable, then that section is included to tag statements. Again section is checked for subsections and further inside each section, statements are checked for dependency of that variable. If match is found then statements for which variable dependency is present, merged to tag statement list. Tag statement list is a list which stores statements dependent corresponding to particular variable.

4. Match detection technique:

Various clone detection techniques are presented in various researches. With every technique presented have tools for detecting clones and gives their appropriate result. Four types of clones viz. type 1, type 2, type 3, and type 4 are detected by different technique.For any technique, the source representation and match detection technique are most important characteristics. The detection of code clones is mainly a two phase process: transformation and a comparison phase. Transformation phase is pre-processed phase in which removing any uninteresting parts like comments and blank lines. Intermediate representation is a way of extracting useful information based upon which comparison is done.Various intermediate representation/ transformation techniques are available viz. AST parse tree, Regularized token, call graph, vector space, PDG etc.[2] After choosing source code representation technique, match detection algorithms are applied. Clone detection techniques are classified into various types: text based, token based, tree based, graph based, metric based and hybrid based techniques. Program slices are extracted in first phase of process software clone detection. In second phase of process of detecting clones a match detection technique must be used for comparison. In textual comparison renaming of variables does not appears as similar match and results in the conflict of matching. So, we are using appropriate token based comparison technique. Tokens are smallest individual unit of a program. In token based comparison technique tokens are extracted from lexical analysis/parsing/transformation of complete source program. Then, this extracted sequence of tokens is scanned for detecting cloned subsequence of tokens. This technique is not robust against code change from text based technique. In our tokenization process, we are tokenizing variables of program slices into tokens and then slices with tokens of variables are compared. If match is found it results in software clones as according number of statements match of program slices and gives percentage and type of clones.

5. Results:

The approach for detecting clones using program slicing has been discussed in the 3rd section. This ssection will discuss the cloning results of the developed approach by use of program slicing. In designed GUI (Graphical User Interface), all steps are incorporated in single design frame so that it gives better result for understanding. Then, the factors responsible for detecting clones in diagrams of individual type are discussed. Finally, the cloning results are provided through the screenshots of developed approach.



C:\Users\Richa\Desktop\Richa Programs\amst.txt

C:\Users\RichaDesktop\Richa Programs\Roots.txt

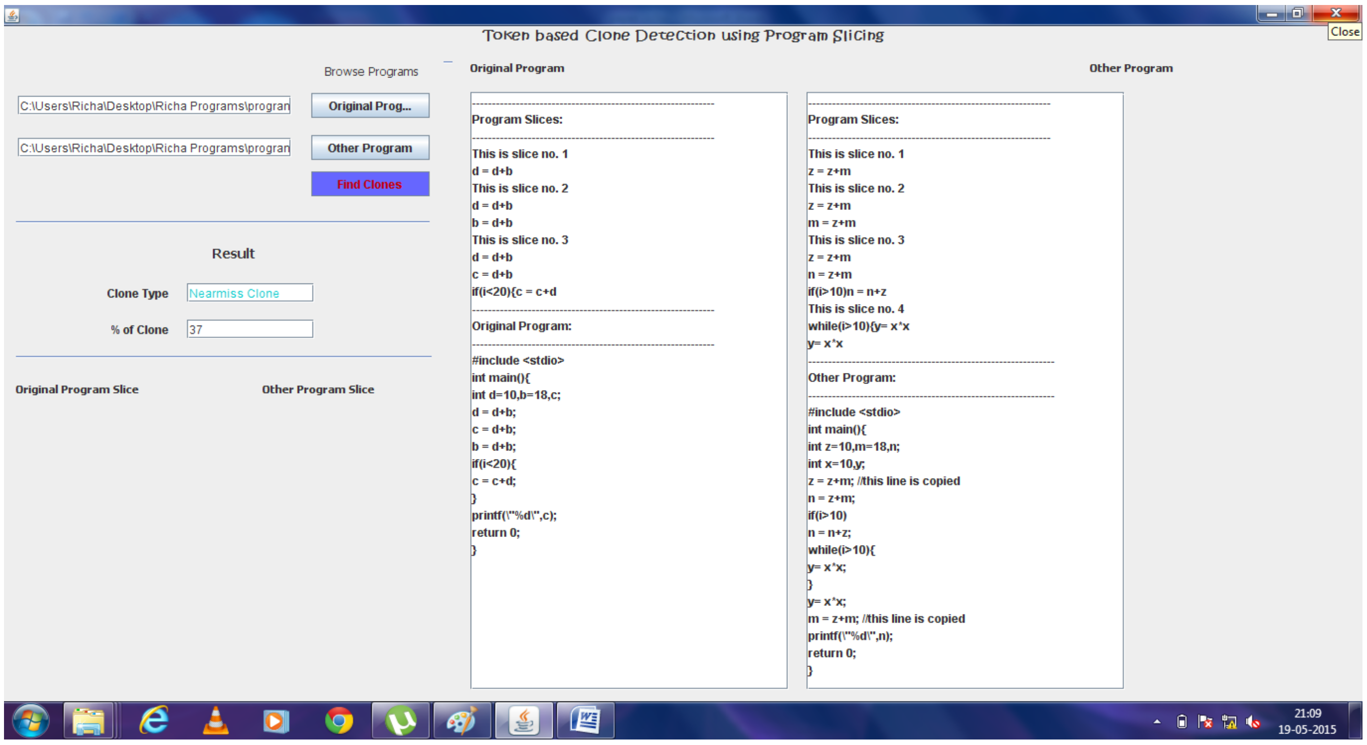
Clone Type: No Clone

% of Clone: 0

```
---------------------------------------------------------
Program Slices:
---------------------------------------------------------
This is slice no. 1
sum = sum + (rem*rem*rem)
This is slice no. 2
while (num != 0){rem = num % 10
---------------------------------------------------------
Original Program:
---------------------------------------------------------
#include<stdio.h>
int main()
{
int num,temp,sum=0,rem;
printf("nEnter Number For Checking Armstrong : ");
scanf("%d",&num);
temp = num;
while (num != 0)
{
rem = num % 10;
sum = sum + (rem*rem*rem);
num = num / 10;
}
if(temp == sum)
printf("n%d is Amstrong Number",temp);
else
printf("n%d is Amstrong Number",temp);
return(0);
}
```

```
---------------------------------------------------------
Program Slices:
---------------------------------------------------------
This is slice no. 1
desc = sqrt(b*b-4*a*c)
This is slice no. 2
root2 = (-b - desc)/(2.0*a)
---------------------------------------------------------
Other Program:
---------------------------------------------------------
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(){
float a,b,c;
float desc,root1,root2;
clrscr();
printf("nEnter the Values of a : ");
scanf("%f",&a);
printf("nEnter the Values of b : ");
scanf("%f",&b);
printf("nEnter the Values of c : ");
scanf("%f",&c);
desc = sqrt(b*b-4*a*c);
root1 = (-b + desc)/(2.0*a);
root2 = (-b - desc)/(2.0*a);
printf("nFirst Root : %f",root1);
printf("nSecond Root : %f",root2);
getch();
}
```

Fig1: No Clones



C:\Users\Richa\Desktop\Richa Programs\progran

C:\Users\Richa\Desktop\Richa Programs\progran

Result

Clone Type: Nearmiss Clone

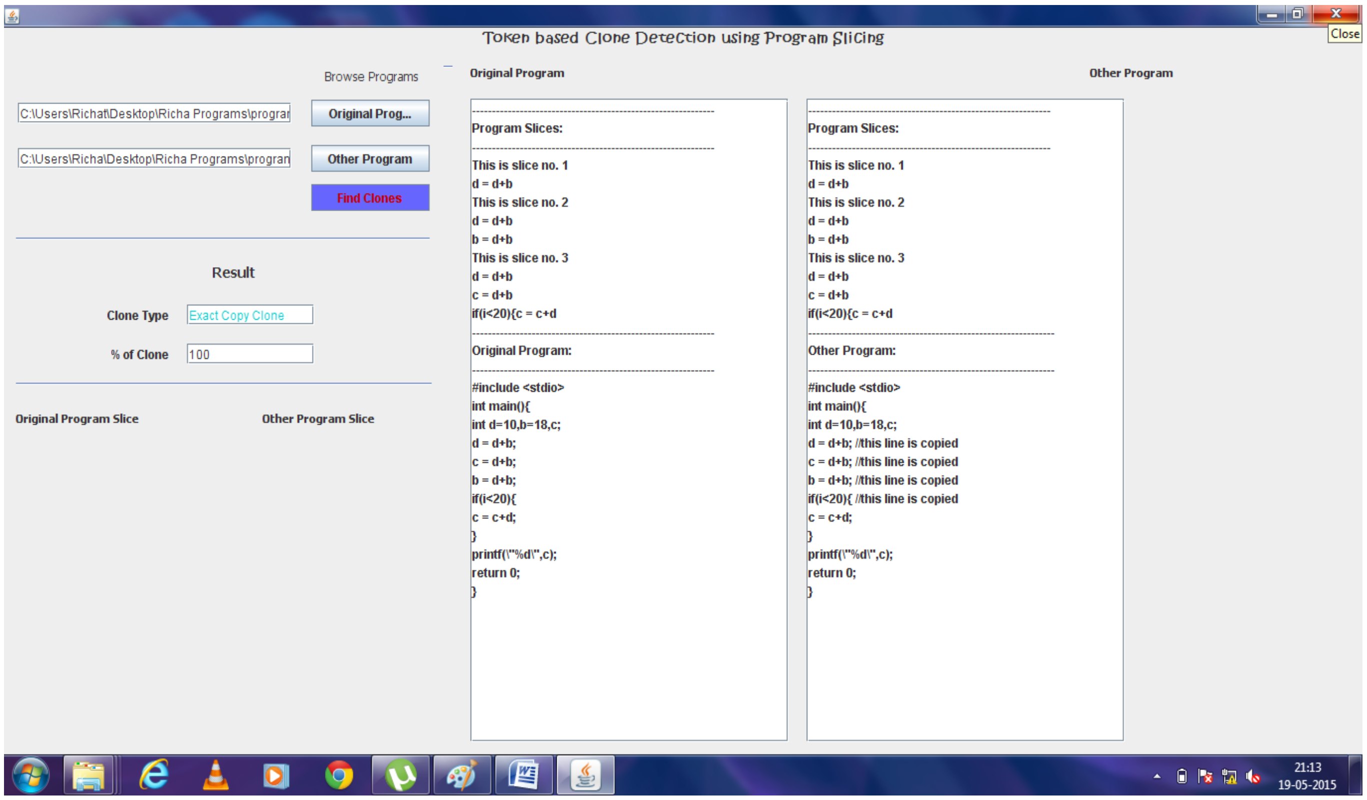
% of Clone: 37

```
-------------------------------------------------------------
Program Slices:
-------------------------------------------------------------
This is slice no. 1
d = d+b
This is slice no. 2
d = d+b
b = d+b
This is slice no. 3
d = d+b
c = d+b
if(i<20){c = c+d
-------------------------------------------------------------
Original Program:
-------------------------------------------------------------
#include <stdio>
int main(){
int d=10,b=18,c;
d = d+b;
c = d+b;
b = d+b;
if(i<20){
c = c+d;
}
printf(\"%d\",c);
return 0;
}
```

```
-------------------------------------------------------------
Program Slices:
-------------------------------------------------------------
This is slice no. 1
z = z+m
This is slice no. 2
z = z+m
m = z+m
This is slice no. 3
z = z+m
n = z+m
if(i>10)n = n+z
This is slice no. 4
while(i>10){y= x*x
y= x*x
-------------------------------------------------------------
Other Program:
-------------------------------------------------------------
#include <stdio>
int main(){
int z=10,m=18,n;
int x=10,y;
z = z+m; //this line is copied
n = z+m;
if(i>10)
n = n+z;
while(i>10){
y= x*x;
}
y= x*x;
m = z+m; //this line is copied
printf(\"%d\",n);
return 0;
}
```

21:09
19-05-2015

Fig2: Near-Miss Clones



C:\Users\Richat\Desktop\Richa Programs\prograi

C:\Users\Richa\Desktop\Richa Programs\progran

Result

Clone Type: Exact Copy Clone

% of Clone: 100

```
-------------------------------------------------------------
Program Slices:
-------------------------------------------------------------
This is slice no. 1
d = d+b
This is slice no. 2
d = d+b
b = d+b
This is slice no. 3
d = d+b
c = d+b
if(i<20){c = c+d
-------------------------------------------------------------
Original Program:
-------------------------------------------------------------
#include <stdio>
int main(){
int d=10,b=18,c;
d = d+b;
c = d+b;
b = d+b;
if(i<20){
c = c+d;
}
printf(\"%d\",c);
return 0;
}
```

```
-------------------------------------------------------------
Program Slices:
-------------------------------------------------------------
This is slice no. 1
d = d+b
This is slice no. 2
d = d+b
b = d+b
This is slice no. 3
d = d+b
c = d+b
if(i<20){c = c+d
-------------------------------------------------------------
Other Program:
-------------------------------------------------------------
#include <stdio>
int main(){
int d=10,b=18,c;
d = d+b; //this line is copied
c = d+b; //this line is copied
b = d+b; //this line is copied
if(i<20){ //this line is copied
c = c+d;
}
printf(\"%d\",c);
return 0;
}
```

21:13
19-05-2015

Fig3: Exact Clones

6. Conclusion and future work:

Software clone detection is a very broad and important research area to improve maintainability and quality of the system. A large number of clone detection techniques have been explored over the last two decades. After going through a vast study of the literature in this area, we came to a conclusion that a limited work is done in the area of determining intertwined code clones. By determining the intertwined clones, it is highly probable that each and every possible clone is successfully detected The first phase for this approach is to remove the dead code from souce code. But in future input source code file need to be modified and then any form of source code file can taken as input in that case there is no requirement of any dead code removal.This approach just detect the software clones and in future it may be possible to detect the malware code by checking it with machine code.This approach is applicable for detecting software clone in structured programming i.e.for 'c' programs, further it will be extended for object-oriented programs like 'c++' and 'java' programs.

7. References:

1. C.K. Roy, J.R. Cordy, A Survey on Software Clone Detection Research, Technical Report 2007-541, Queen's University at Kingston Ontario, Canada, 2007, p. 115.
2. Dhavlesh Rattan, Rajesh Bhatia, Maninder Singh, Software Clone Detection: A Systematic Review, Information and Software Technology 55 (2013) 1165-1199.
3. J.-F. Bergeretti and B.A. Carr´e. Information-flow and data-flow analysis of **while**-programs. ACM transactions on Programming Languages and Systems, 7(1):37–61, 1985.
4. A Survey of Program Slicing Techniques by Frank Tip.
5. Weiser, Mark. "Program Slicing." Proc. 5th Intl. Conference on Software Engineering, San Diego, California, IEEE Computer Society, March 1981, 439-449.
6. B. Baker, On finding duplication and near-duplication in large software systems, in: Proceedings of the 2nd working Conference on Reverse Engineering (WCRE'95), Toronto, Ontario, Canada, 1995, pp. 86–95.
7. S. Bellon, R. Koschke, G. Antoniol, J. Krinke, E. Merlo, Comparison and evaluation of clone detection tools, IEEE Transactions on Software Engineering 33 (9) (2007) 577–591.
8. T. Kamiya, S. Kusumoto, K. Inoue, CCFinder: a multi-linguistic token-based code clone detection system for large scale source code, IEEE Transactions on Software Engineering 28 (7) (2002) 654–670.
9. Z. Li, S. Lu, S. Myagmar, Y. Zhou, CP-Miner: finding copy–paste and related bugs in large-scale software code, IEEE Transactions on Software Engineering 32 (3) (2006) 176–192
10. R. Koschke, R. Falke, P. Frenzel, Clone detection using abstract syntax suffix trees, in: Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06), Benevento, Italy, 2006, pp. 253–262.
11. Y. Higo, S. Kusumoto, Code clone detection on specialized PDG's with heuristics, in: Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR'11), Oldenburg, Germany, 2011, pp. 75–84.
12. R. Komondoor, S. Horwitz, Using slicing to identify duplication in source code, in: Proceedings of the 8th International Symposium on Static Analysis (SAS'01), vol. LNCS 2126, Paris, France.