SPL-1 Project Report, 2019

SORUCE CODE PLAGIARISM DETECTION

SE 305: Software Project Lab 1

Submitted by

Shakh Mohibul Alam

BSSE Roll No. : 1004

BSSE Session: 2017-18

Supervised by

Md. Saeed Siddik

Designation: Lecturer

Institute of Information Technology

Institute of Information Technology

University of Dhaka

[29-05-2019]

Table of Contents

1. Introduction

Parker and Hamblen define source code plagiarism as “a program which has been produce from another program with a small number of routine transformations.” Source code plagiarism can vary from copy-pasting small amounts of program source code to copying large chunks of source code and masking everything with some techniques to disguise copied program.

Source-code plagiarism detection in programming, concerns the identification of source-code files that contain similar and/or identical source-code fragments. Based on the analysis of the characteristics and defects of the existing program code similarity detection system, a method of source code similarity detection. Code segments usually occurs due to replication from one place and then rewrite them in to another section of code with or without variations/changes are software cloning and the copied code is called clone.

Various researchers have reported more than 20–59% code replication. The problem with such copied code is that an error detected in the original must be checked in every copy for the same bug. Moreover, the copied code expansions the effort to be done when augmenting the code .

However, the code quality analysis (improved quality code), replication identification, virus recognition, facet mining, and bug exposure are the other software engineering tasks which require the mining of semantically or syntactically identical code segment to facilitate clone detection significant for software analysis .

1.1. Background Study

1.1.1 CODE CLONE

Code clone is a computer programming term for a sequence of source code that occurs more than once. It can be located either within a program or across different programs owned or maintained by the same entity. Cloning unnecessarily increases program size.

It requires the developers to find and update several fragments while changing any module. This section contains the related topics which should be known before introducing code clone.

1.1.2 Code Fragment

A Code Fragment (CF) is any sequence of code lines (with or without comments). Clone is detected using comparison between the fragments in a source code. It can be of any types of code, for example function definition, begin-end block, or sequence of statements. A CF is identified by its file name and begin-end line numbers in the original code base.

1.1.3 Source code based plagiarism detection techniques

1.Lexical Similarities
2.Parse Tree Similarities
3. Program Dependence Graphs
4. Metrics

1.1.3.1 Lexical Similarities

Converts source code into a stream of lexical tokens from which compiler extract meaning from the source. During the lexical analysis phase, the source code undergoes a series of transformation . Some of these transformations, such as the identification of reserved words, identifiers are beneficial for plagiarism detection.

Consider the following two snippets of C Code:

Code-1

```
int A[]={1,2,3,4};
int i;
for ( i=0 ; i<4 ; i++  )
{
    A[i]=A[i]+1;
}
```

```
nt B[]={1,2,3,4};
int i;
for ( i=0 ; i<4 ; i++  )
{
    B[i]=B[i]+1;
}
```

Lexical Stream of two C code are :

int_declearation_int_declearation_loop_operation_end_of_loop

int_declearation_int_declearation_loop_operation_end_of_loop

Both the C snippets will have the exact lexical stream .So , this two code is copied .

1.1.3.2 Parse Tree Similarities

The parse tree or derivation tree built from the lexical for a program also exhibits structure for a given program A compiler, during the compilation process builds a parse tree which represents the program. The parse tree will have the same structure for both the snippet of code as the lexical streams are same. An algorithm for detecting plagiarism using this method would first, parse each program. Next, for each pair of parse trees, it attempts to find as many common sub trees as possible. Use this number as a measure of similarity between the two programs.

1.1.3.3 Plagiarism detection by similarity analysis using software metrics.

- Software metrics are:
- Number of function calls
- Number of used or defined local variables
- Number of used or defined non-local variables
- Number of parameters
- Number of statements
- Number of branches
- Number of loops

Each fragment characterized by a set of features measured by metrics Metrics computation requires the parsing of source code to identifying interesting fragments Metrics are simple to calculate and can be compared quickly False positives: two fragments with the same scores on a set of metrics may do entirely different things.

1.1.3.4 Program Dependence Graphs(PDG)

PDG is a graph representation of the source Code .It is a directed, labeled graph which represents the data and the control dependencies within one procedure .Basic statements like variable declarations, assignments, and procedure calls are represented by vertices in PDGs. It depicts how the data flows between statements and how statements are controlled by other statements. The data and control dependencies between statements are represented by edges between vertices in PDGs .Data and control dependencies are plotted in solid and dashed lines respectively.

1.1.4 The types of copying/cloning source code are:

- **Type-1:** Identical code fragments except for variations in whitespace, layout and comments. Figure 1 shows this type of code clone.

Orginal Code :

```
void sumProd(int n)
{
		float sum=0.0;
		float prod=1.0;
		for(int i=0 ; i<=n ; i++)
		{
			sum+=i;
			prod=prod*i;
			foo(sum,prod);
		}
}
```

Copied Code :

```
 void sumProd(int n)
{
		float sum=0.0;
		float prod=1.0;
		for(int i=0 ; i<=n ; i++)
		{
			sum+=i;
			prod=prod*i;
			foo(sum,prod);
		}
}
```

- **Type-2:** Syntactically identical fragments except for variations in identifiers, literals, types, whitespace, layout and comments. Figure 2 shows this type of code clone.

Orginal Code :

```
	void sumProd(int n)
	{
			float sum=0.0;
			float prod=1.0;
			for(int i=0 ; i<=n ; i++)
			{
				sum+=i;
				prod=prod*i;
				foo(sum , prod);
			}
	}
```

Copied Code :

```
  void sumProd(int n)
 {
		float s=0.0;
		float p=1.0;
		for(int j=0 ; j<=n ; j++)
		{
			sum+=j;
			prod=prod*j;
			foo(s , p);
		}
 }
```

Type-3: Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments. Fig 3 shows this type of code clone.

Original Code :

```
void sumProd(int n)
{
		float sum=0.0;
		float prod=1.0;
		for(int i=0 ; i<=n ; i++)
		{
			sum+=i;
			prod=prod*i;
			foo(sum ,prod);
		}
}
```

Copied Code :

```
void sumProd(int n)
{
		float sum=0.0;
		float prod=1.0;
		for(int i=0 ; i<=n ; i++)
		{
			sum+=i;
			prod=prod*i;
			foo(sum );
		}
}
```

- **Type-4:** Two or more code fragments that perform the same computation but are implemented by different syntactic variants. Fig 4 shows this type of code clone.

Orginal Code:

```
void sumProd(int n)
{
		float sum=0.0;
		float prod=1.0;
		for(int i=0 ; i<=n ; i++)
		{
			sum+=i;
			prod=prod*i;
			foo(sum ,prod);
		}
}
```

Copied Code :

```
void sumProd(int n)
{
		float sum=0.0;
		float prod=1.0;
		int i=0;
```

```
        while(i<=n)
        {
                sum+=i;
                prod=prod*i;
        foo(sum,prod);
                i++;
        }
}
```

1.2. Challenges

I faced so many problem to implement this project . I did not about the clone detection techniques and clone type . I did not know necessary algorithms (as example edit-distance). Then I learned necessary algorithms and clone detection techniques and clone type.

2. Project Overview

In this project , I apply various code clone detection techniques to detect clone between two C Code code. I apply lexical analysis ,Software Metrics , abstruct syntax tree based plagiarism detection techniques to detect plagiarism between two C Code .By applying these techniques , I fully detect type-1 and type-2 code clone. In Some of case of type-3 clone ,my code handle it. Anyone can detect plagiarism between a C source file and a folder in which many of c code are stored by using my project code and create an csv file .

3. User Manual

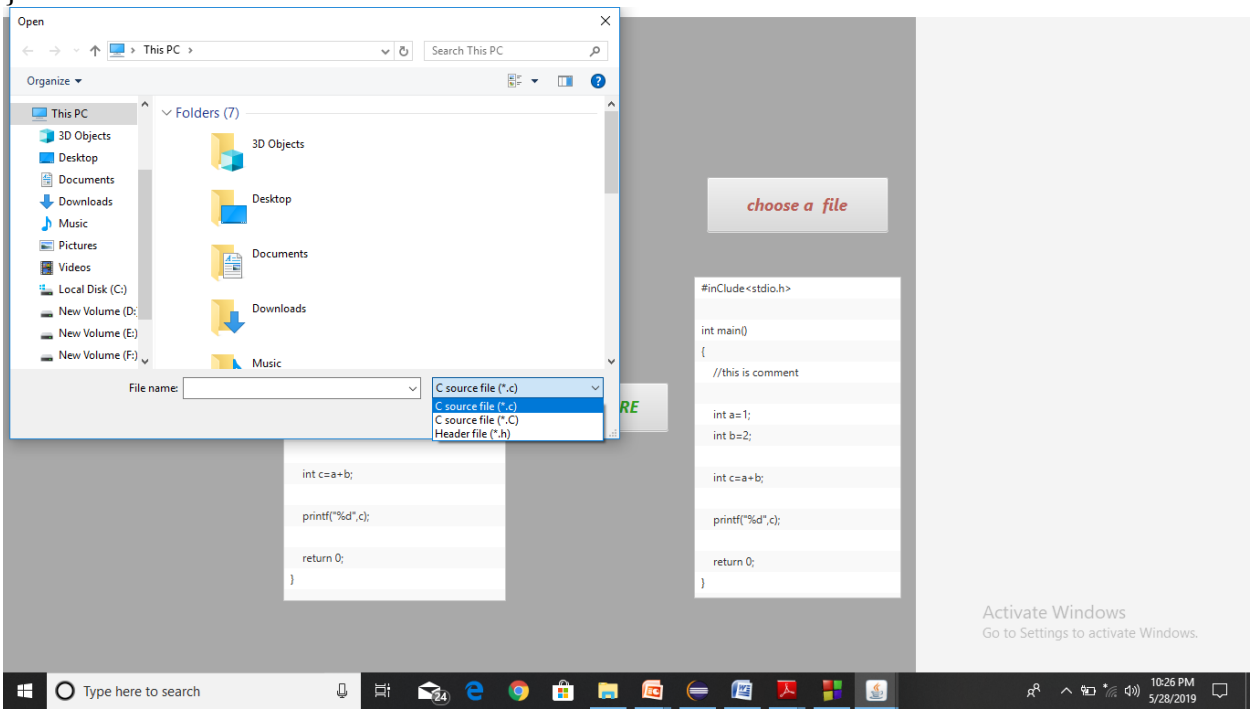
When the input two code are same : (type-1 Clone)

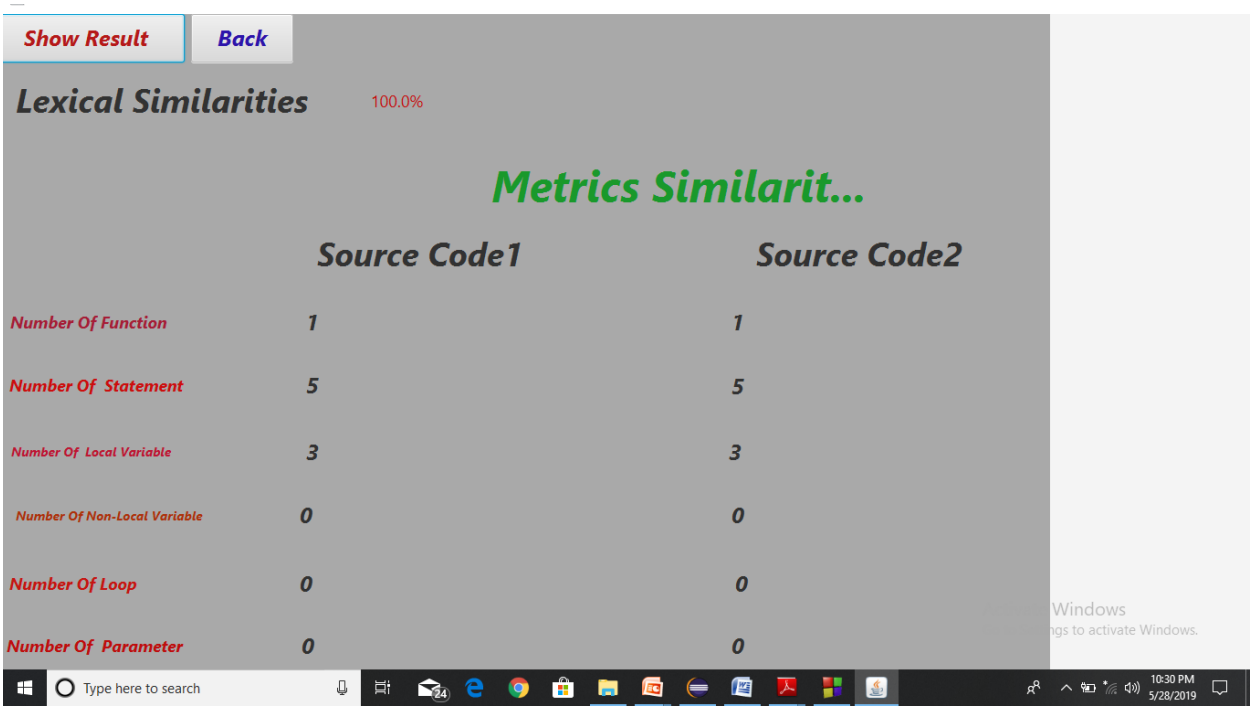
Code a:

```
#include<stdio.h>

int main()
{
   //this is comment

   int a=1;
   int b=2;

   int c=a+b;

   printf("%d",c);

   return 0;
}
```

Code b:

```
#include<stdio.h>

int main()
{
   //this is comment

   int a=1;
   int b=2;

   int c=a+b;

   printf("%d",c);

   return 0;
}
```





When the variable's name are changed and more comment are added(type-2 Clone) :

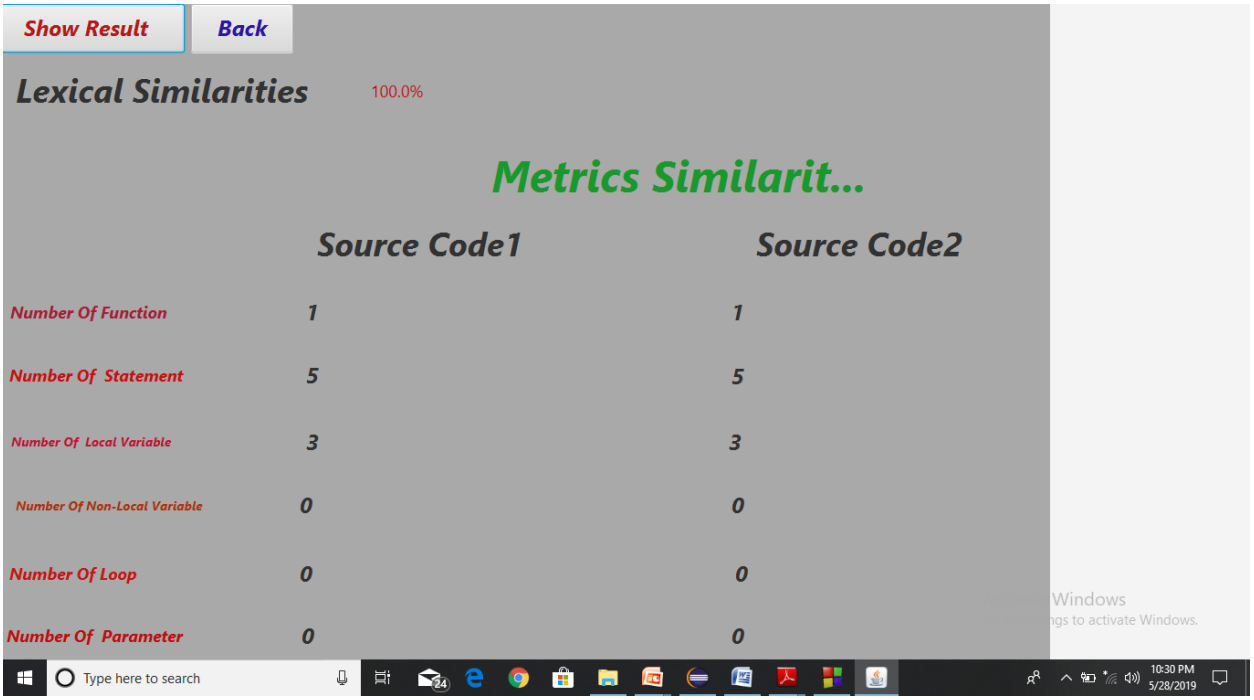
Code A :

```
#include<stdio.h>

int main()
{
   //this is comment

   int a=1;
   int b=2;

   int c=a+b;

   printf("%d",c);

   return 0;
}
```

Code B:

```
#inClude<stdio.h>

int main()
{
```

```
    //this is comment

    int x=1;
    int y=2;

    /*


        this is multiline comment


    */

    int z=a+b;

    printf("%d",c);

    return 0;
}
```



4.Conclusion

The code-clone detection is an emerging issue in the software ecosystem which degrades the software's comprehensibility as well as maintainability. Therefore, its analysis and detection is necessary for improving the quality, structure and design of the software system. This evaluations are not only intended for experts in clone detection, but also intended for potential new users and builders of clone detection-based tools and applications. It is hoped that it may also assist in identifying remaining open research questions, avenues for future research, and interesting combinations of techniques.

5.Appendix

I apply these plagiarism detection techniques to know about these techniques and code analysis . My Supervisor also suggest me to do this project.

6.Reference

[1] S. Bellon, R. Koschke, G. Antoniol, J. Krinke and E. Merlo, Comparison and Evaluation of Clone Detection Tools, Transactions on Software Engineering, 2007, pp. 33(9):577-591.

[2] M. Gabel, L. Jiang and Z. Su, Scalable Detection of Semantic Clones, in: Proceedings of the 30th International Conference on Software Engineering, ICSE 2008, pp. 321-330.

[3] B. Baker, A Program for Identifying Duplicated Code, in: Proceedings of Computing Science and Statistics: 24th Symposium on the Interface, 1992, Vol. 24:4957, 24:49-57.

[4] T. Kamiya, S. Kusumoto and K. Inoue, CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code, IEEE Transactions on Software Engineering, 2002, 28(7):654-670.

[5] V. Wahler, D. Seipel, J. Gudenberg and G. Fischer, Clone Detection in Source Code by Frequent Itemset Techniques, in: Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation, SCAM 2004, pp. 128-135.

[6] W. Yang, Identifying Syntactic Differences between Two Programs, Software Practice and Experience, 1991, 21(7):739