Advances in Computer Science and Information Technology (ACSIT)
Print ISSN: 2393-9907; Online ISSN: 2393-9915; Volume 2, Number 8; April-June, 2015 pp. 18-21

http://www.krishisanskriti.org/acsit.html

# Evaluation of different Parameters to Detect the Code Similarity

**Amandeep Kaur[1] and Harpreet Kaur[2]**

*[1,2]Department of Computer Engineering Punjabi University Patiala*
*E-mail: [1]sharleenballu@gmail.com, [2]khasria.harpreet@gmail.com*

**Abstract—***Code duplication or code clone plays an important role in software engineering .Code cloning is known as copying the code fragments and then reuse it by pasting with or without modifications .The code cloning make the software programming easier but it effects on the maintenance of the software system. It also leads to the bad quality of software system. To detect the similar code the various technique are used like Text- based, Tree-based, PDG-based and Metric-based. Most previous work on clone detection has focused on finding identical clones, and insertion/deletion/modification clones. However it is often important to find code similarity percentage of two files. In this paper, therefore we calculated different parameters to quantify code similarity.*

*In this paper we used number of parameters 1.Euclidean distance 2.Co-relation 3.Chi-square 4.Cosine similarity 5.Proportional similarities which are evaluated to detect the code similarity. The used approach transforms the source program into tokens and then parameters are calculated on grid formed from detected Tokens .All the parameters detect the similarity very well but it has been observed that Euclidean distance is best suitable to detect the similarity among all evaluated parameters. Case study of c program was evaluated.*

**Keywords***: Code Clone, Parameters, Similarity, Token*

## 1. INTRODUCTION

In software development, programmers frequently reuse the code fragments. Copying the original code and then paste it with or without modifications. These code fragments which are identical or similar are called code clones. Clones are shown to be harmful in software maintenance phase and evaluation. According to the previous research about 7%-23% code in software is cloned. Code cloning is problematic in software maintenance phase, as it increases error in the software [7, 8]. If the error find in one code, we have to check the cloned code also to correct error. It increases the maintenance cost and effort. It also degrades the code quality. It also leads to the bad design of the software due to the presence of bugs. Hence we need to detect the clones of the code and the detection process is known as clone detection. [5, 6, 8]

For the detection of clones, number of techniques has been proposed. The Text-based approach is simple and easier to detect the clone. In this the two code fragments are compared with each other to find the same text/strings. Token-based in which the source code is broken into tokens and then on these tokens operations are applied to detect the clones. In Tree-based approach source program is parsed into the sub-trees, then the matching process is applied on the sub-trees and similar trees are returned as cloned. Program Dependency Graph (PDG) contains the control flow and data flow information of program and hence carries the semantic information. In this technique, on the basis of data flow information graphs are obtained from the source program and by using the matching algorithm similar sub-graphs are retuned as clones. Metric-based approaches calculate the different matrices and compare them instead of comparing the code directly. These all the previously used techniques detect the types of code clones but do not give the percentage of similarity between two codes. [7, 8, 9]

In this paper, an approach for evaluating the different parameters to calculate the similarity between two programs is proposed. Rather than detecting the Types of code clones, number of parameters are used to find the similar values in numeric form. In this proposed approach, Token-based technique is applied. The source codes are broken into the token, then token list is prepared on which these parameters are applied. The proposed approach is efficient to calculate the similarity percentage between the original and cloned code.

This paper includes five sections. Section 2 describes the related terminologies which are used in proposed work. Section 3 gives the details of the proposed approach. Section 4 gives the results analysis from the proposed work. Section 5 gives conclusion and directions for the future work.

## 2. RELATED TERMINOLOGY

*1. Euclidean distance*

It calculates the distance between two points. Euclidean distance is used to calculate the original and cloned tokens. The Euclidean distance between them is calculated as:-

$$D(u, v) = ||u-v||_2 = \sqrt{\sum_{i=0}^{k} (u-v)^2}$$

In Euclidean distance similarity of two codes is related to their lengths (means detected tokens). That if the counted tokens in the two codes are very long, their distances are likely to be longer, but they might be similar as well. If their distances are small then they are similar and similarity percentage is evaluated according to their obtained values.

*2. Co-relation*

It refers to any of a broad class of statistical relationships involving the dependencies. It gives the relationship between the two codes. Co-relation function is used to find the dependencies between two codes. It is the measure of similarity of two codes. It gives the values between the 1 and -1.Co- relation between the two codes given by:-

$$R= 1/ (n-1) \sum (u- \bar{u}/ s_u) (v-\bar{v}/s_v)$$

The positive value indicates the greater association between the two codes means their codes are similar. The negative value or 0 indicate that there is no relationship between the two codes and they are dissimilar.

*3. Chi-square*

It designed to arrange the numeric values and values are counted and divided into categories.

Chi-Square test is also used to detect the similarity. Chi-square basically calculates the difference between the original and cloned code. In this original code is taken as the observed value and cloned code is taken as the expected value. Chi-square is calculated as:-

$$C= diff/sum = (diff=sum (sum ((u-v). ^2)))$$

$$Sum = sum (sum (u + v))$$

The value lies between 1 and 0. Value 1 show that both code are similar but value less than or equal 0.5 show the dissimilarity between the codes.

*4. Cosine –similarity function*

It is the measure of similarity between two vectors.Cosine-similarity function is a measure of similarity between the two vectors by measuring the cosine angle between them. If the value is greater than codes are similar otherwise they are dissimilar. The result lies between the 1 and 0. Cosine similarity formula is given by:-

$$CosSim = \sum_{i=1}^{k} u_i \times v_i / (\sqrt{\sum_{i=1}^{k} u_i} \times \sqrt{\sum_{i=1}^{k} v_i})$$

*5. Proportional –similarity function*

This function is modified to prevent the incorrect zero similarity. Proportional –similarity function is used to compare the token marks (keywords, punctuators etc.). The function is modified to prevent the incorrect the zero similarity. Given two occurrence counts u and v (u>=v). Their proportional similarity is defined as

$$ProSim = (1/(u+1)) + (1/(v+1))$$

## 3. PROPOSED APPROACH

A. In the token based approach code broken (transformed) into the tokens. The numeric values are calculated to find the similarity percentage of the two codes.

Fig. 1. Show the flowchart of the proposed approach. It consists of four steps which are given in detail in this section.

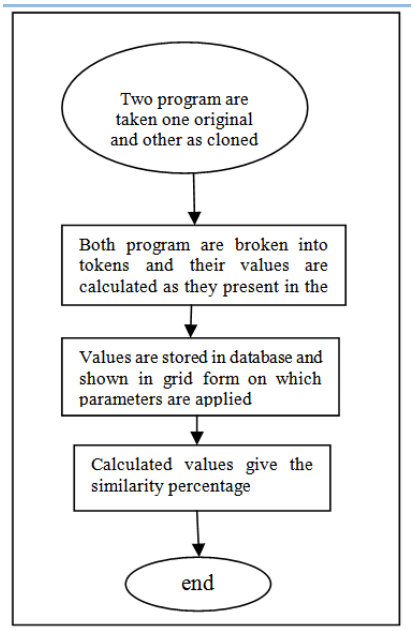


**Fig. 1: Flowchart of the proposed approach**

*B. Steps followed in the Proposed Approach*

1. *Source code selection.*

In the source code the two programs are taken on which token based technique is applied. One code is taken as the original and another one is taken as cloned. The cloned or copied code is taken is three different ways 1. By deleting the comments, blank lines, layouts. 2. By changing the variables names, constants, class, methods or so on. 3. By adding or deleting some statements in the original code.

*2. Generation of token list*

The both files are broken into the tokens and a numeric values are calculated as the token present in both programs. Token list comprises of keywords, operators, punctuators, identifiers, literals.

Keyword contains the (1) decision statements (2) control statements (3) nested loops etc. Operator defined by the (1) addition/ subtraction /multiplication/division. (2) Greater then, less then equal to and many more. Punctuators defined by (1) array subscript, braces, asterisk, colons, semicolons. Identifiers and Literals values are also calculated.

*3. Storage of values and Selection of parameters*

The calculated token values are stored in database and also stored in grid form. The various parameters are selected for the calculation of similarity. Now these parameters are calculated on grid formed from detected tokens.

*4. Similarity percentage*

These parameters give the similarity of the original code and cloned code. Similarity is lies between 0 and 1.

*C. Parameters used in similarity calculation*

The following parameters are used in the proposed approach for similarity calculation. These parameters are detailed with their mathematical used formula.

*C. Calculating The Similarity*

The similarity of two programs is calculated on the grid formed from the detected tokens. We broke both the program into token and sort them according to their used frequencies in both the programs. Then used frequencies are put into one list of one program. On these values parameters are applied.

In the proposed method we use the frequencies differently and calculate them separately. Token list contains the keywords, operators, punctuators, identifiers, and literals separately of both the files. In this approach we applied all parameters differently on each token. Keyword list is compared with the keyword and parameters are calculated. Similarly others are also calculated in same way. The two programs are same written with same logic but different with some statements, identifiers, method used. We compares them to find the similarity between them.

```
                break;
        case '5': return 0;
        case '6': system("cls");
                break;
        default : system("cls");
                printf("Please enter one of the options");
                printf("(1/2/3/4/5/6) to continue \n ");
                break;
      }
   }
   return 0;
}

/*Function to display available options in this application*/

void display_options()
{
    printf("\n1. Create new account \n");
    printf("2. Cash Deposit \n");
    printf("3. Cash withdrawl \n");
    printf("4. Account information \n");
    printf("5. Log out \n");
    printf("6. Clear the screen and display available ");
    printf("options \n\n");
}

/* Function to create new account */

void Create_new_account()
{
    char bank_name[20];
    char bank_branch[20];
    char acc_holder_name[30];
    int acc_number;
    char acc_holder_address[100];
    float available_balance = 0;
    fflush(stdin);
    printf("\nEnter the bank name              : ");
    scanf("%s", &bank_name);
    printf("\nEnter the bank branch            : ");
    scanf("%s", &bank_branch);
    printf("\nEnter the account holder name    : ");
    scanf("%s", &acc_holder_name);
    printf("\nEnter the account number(1 to 10): ");
```

**Program 1**

```
void calculator_operations()
{
    //system("cls");  use system function to clear
    //screen instead of clrscr();
    printf("\n            Welcome to C calculator \n\n");

    printf("******* Press 'Q' or 'q' to quit ");
    printf("the program ********\n");
    printf("***** Press 'H' or 'h' to display ");
    printf("below options *****\n\n");
    printf("Enter 'C' or 'c' to clear the screen and");
    printf(" display available option \n\n");

    printf("Enter + symbol for Addition \n");
    printf("Enter - symbol for Subtraction \n");
    printf("Enter * symbol for Multiplication \n");
    printf("Enter / symbol for Division \n");
    printf("Enter ? symbol for Modulus\n");
    printf("Enter ^ symbol for Power \n");
    printf("Enter ! symbol for Factorial \n\n");
}
{
void addition()
{
    int n, total=0, k=0, number;
    printf("\nEnter the number of elements you want to add:");
    scanf("%d",&n);
    printf("Please enter %d numbers one by one: \n",n);
    while(k<n)
    {
        scanf("%d",&number);
        total=total+number;
        k=k+1;
    }
    printf("Sum of %d numbers = %d \n",n,total);
}

void subtraction()
{
    int a, b, c = 0;
    printf("\nPlease enter first number  : ");
```

**Program 2**

These both programs are compared with each other to find the similarities.

**Table 1: Similarity between keywords.**

| Parameters name | Calculated values | Case study |
|---|---|---|
| Euclidean distance | 0 | Keywords present in both c programs |
| Co-relation | 1 | |
| Chi-square | 0.734 | |
| Cosine Similarity | 0 | |
| Proportional Similarity | 1 | |

**Table 2: Similarity between Operators.**

| Parameters name | Calculated values | Case study |
|---|---|---|
| Euclidean distance | 1.56 | operators present in both c programs |
| Co-relation | 1 | |
| Chi-square | 0.534 | |
| Cosine Similarity | 0 | |
| Proportional Similarity | 0 | |

**Table 3: Similarity between punctuators**

| Parameters name | Calculated values | Case study |
|---|---|---|
| Euclidean distance | 1.234 | Punctuators present in both c programs |
| Co-relation | 1 | |
| Chi-square | 0.534 | |
| Cosine Similarity | 1 | |
| Proportional Similarity | 0 | |

**Table 4: Similarity between identifiers**

| Parameters name | Calculated values | Case study |
|---|---|---|
| Euclidean distance | 0 | identifiers present in both c programs |
| Co-relation | 1 | |
| Chi-square | 0.122 | |
| Cosine Similarity | 1 | |
| Proportional Similarity | 1 | |

**Table 5: Similarity between literals.**

| Parameters name | Calculated values | Case study |
|---|---|---|
| Euclidean distance | 1.34 | literals present in both c programs |
| Co-relation | 0 | |
| Chi-square | 0.23 | |
| Cosine Similarity | 0 | |
| Proportional Similarity | 0 | |

The above five Table gives the similarity between tokens after applying the parameters.

## 4. ANALYSIS FROM RESULTS

The results are calculated from the proposed approach shown above in tables. The similarity calculated from the five parameters shown differently. The keywords similarity table shows that keywords are slightly taken similar in both programs.

The operators are differently used in both programs. The

Punctuator values are almost similar. Use identifiers taken very much similar in both programs. Literals values are taken differently in both programs.

From the above discussion we can say that there is codes are similar to each other, as the parameters values lies between the 0 and 1.

**Table 6: Analysis from the obtained results**

| Parameters name | Calculated values | Case study |
|---|---|---|
| Euclidean distance | 1.34 | Total tokens present in both programs |
| Co-relation | 1 | |
| Chi-square | 0.567 | |
| Cosine Similarity | 0 | |
| Proportional Similarity | 1 | |

The above table shows the total calculated similarity between both the programs, on total token frequencies used in the program.

## 5. CONCLUSION AND FUTURE WORK

Different parameters are used like Euclidean distance, Co-relation, Chi-square, Cosine similarity, Proportional similarities which are evaluated to detect the code similarity. From analysis and results we can say that about 60% to 70% both the codes are similar. All the parameters calculate the similarity very well. But it has been observed that the Euclidean distance is best parameter to calculate the similarity percentage.

In future this approach can be applied to the large source code. Also, similarity can be applied on the different Types of code clones. On the basis of similarity precision and recall values can also be detected.

## REFERENCES

[1] Mai Iwamoto, Shunsuke Oshima, Dep. of Electronic and Information Engineering Kumamoto National College of Technology 2627 Hirayama-Shinmachi, Yatsushiro, Kumamoto, Japan.

[2] Lifang Han, Baojiang Cui,RuZhang, Zhongxian Li School of Computer BUPBeijing, China, 2010 International Conference on Multimedia Information Networking and Security.

[3] Yang Yuan and Yao Guo Key Lab of High-Confidence Software Technologies Department of Computer Science, School of EECS, Peking University Beijing 100871, P. R. China.

[4] Hiroaki Murakami, Keisuke Hotta, Yoshiki Higo, Hiroshi Igaki and Shinji Kusumoto Graduate School of Information Science and Technology, Osaka University, 1-5, Yamadaoka, Suita, Osaka, 565-0871, Japan.

[5] Geetika Bansal, Rajkumar Tekchandani Computer Science and Engineering Dept. Thapar University Patiala, India.

[6] Florian Deissenboeck, Benjamin Hummel, Elmar Juergens Institut für Informatik, Technische Universität München Garching b. München, Germany {deissenb,hummelb,juergens}@in.tum.de.

[7] Filip Van Rysselberghe Serge Demeyer Lab On Re-Engineering University Of Antwerp Middelheimlaan 1, B 2020 Antwerpen.

[8] Elmar Juergens, Florian Deissenboeck, Benjamin Hummel, Stefan Wagner Institut f¨ur Informatik, Technische Universit¨at M¨unchen Boltzmannstr. 3, 85748 Garching b. M¨unchen, Germany fjuergens,deissenb,hummelb,wagnerstg@in.tum.de.

[9] Tibor Bakota, Rudolf Ferenc and Tibor Gyim´othy University of Szeged, Department of Software Engineering {bakotat|ferenc|gyimi}@inf.u-szeged.hu.

[10] Chanchal Kumar Roy and James R. CordySeptember 26, 2007Technical Report No. 2007-541 School of ComputingQueen's University at Kingston Ontario, Canada

[11] Mark Gabel Lingxiao Jiang Zhendong Su Department of Computer Science University of California, Davis {mggabel,lxjiang,su}@ucdavis.edu

[12] FABIO CALEFATO, FILIPPO LANUBILE, TERESA MALLARDO Dipartimento di Informatica, University.

[13] Tariq Muhammad, Minhaz F. Zibran, Yosuke Yamamoto, Chanchal K. Roy Department of Computer Science, University of Saskatchawan, Saskatoon, SK, Canada