www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 05 Issue 9 Sep., 2016 Page No. 18221-18224

A Novel Metrics Based Technique for Code Clone Detection

Sushma[1], Jai Bhagwan[2]

[1]Scholar, Computer Science and Engineering Department,
Guru Jambheshwar University of Science & Technology
Hisar, Haryana, India
sushma.jangra0015@gmail.com

[2]Asstt. Professor, Computer Science and Engineering Department,
Guru Jambheshwar University of Science & Technology
Hisar, Haryana, India
jaitweet@gmail.com

Abstract: ***Nowadays, software development is a tricky and time-consuming task. In order to make the development easy, one uses the existing modules with or without a bit change. Modules which are used with or without changes are called as code clones. In several places in case of same or different software, a clone can be used for development purpose. Without having care, copy and paste code can lead to inefficiency smell. This bad smell can be detected by finding and refactoring code clones. A clone can be detected by several existing techniques. In this paper, we present a metric based technique for code clone detection. The existing technique is able to detect method level clones only. We proposed the technique to detect module level clones using several metrics. Our technique is useful to find out type1 and type2 clones at modules level in software systems. For implementation purpose, we have developed a tool in Java named as JSCCD (JS Code Clone Detector) which can detect clones between Java files.***

Keywords: Code Clones, Metrics, Clone Detection, JSCCD.

1. Introduction

In Software Engineering, reuse of code is very common thing as it is difficult to develop the software from scratch. So one has to do copy and paste of existing code while developing a new software. A clone is a code segment which is similar to another and copied from a specific location in a software module. Sometimes, a clone can disturb the program efficiency and make a system more complex. In software development, cloning is a usual practice and detection can be useful for maintenance and evolution. So, clones need to be detected in a software system written code [4][7]. In this paper, we first, describe clones type and their detection methods. In the coming sections, implementation and results are discussed. A clone can be of following types [10][11]:

a) Type 1: These are the exact copy of a programming segment (without whitespaces and comments). Let us consider the following code fragment:

```
int a=0, i;
for (i=0; i<10; i++)
{
   a=a+1;  //addition
}
```

An exact clone of the above code could be as given below:

```
int a=0, i; // variable declaration
for (i=0; i<10; i++)
{
   a=a+1;
}
```

(b) Type 2: Type 2 clones are having similar syntax but different identifier (variables, functions, class etc.) names may be different. Let us consider the following code fragment again:

```
int a=0, i;
for (i=0; i<10; i++)
{
   a=a+1;
}
```

A Type II clone for this fragment could be as shown below:

```
int x=0, j;
for(j=0; j<10; j++)
{
   x=x-1;
}
```

(c) Type 3: These are copied fragments and a statement can be added or deleted here. Consider the following code segment:

```
int a, b, c, n=13;
if (n>10)
{
   c=a*b;
}
```

```
else
{
   c=a+b;
}
```

We delete else part of the code shown above and then get the following:

```
int a, b, c, n;
if (n>10)
{
   c=a*b;
}
```

Now this code is considered in the category of Type 3 clones.

Type 4: The code fragments having similar functionality are called as type 4 clones. Consider the original code segment:

```
int factorial (int a)
{
   int j, f=1;
   for(j=1; j<=a; j++)
   {
      f=f*j;
   }
   return (f);
}
```

Type 4 clone for this fragment can be as follows:

```
int factorial (int a)
{
   if (a<=1)
      return 1;
   else
   return (a *factorial (a-1));
}
```

1.1 Clone Detection Methods

i) Text Based Method: In text-based method each line of source code is compared to another and if the line is matched then it is considered as clone [12].
ii) Token Based Method: In this case, the source code is converted into tokens and after token comparison; the clone is decided [12].
iii) AST Based Method: In this case, an Abstract Syntax Tree is generated from source code and sub-trees having similar structure are considered as clones [12].
iv) PGD Based Method: Here, a Program Dependency Graph is generated and after semantic analysis of source code, isomorphic sub graphs are considered as clones [12].
v) Metric Based Method: From a certain unit of source code, several metrics are calculated e.g. methods, class, package etc. of a software system; segments having similar metric values are calculated as code clones [12]. In the coming section, we describe the review of literature.

2. Literature Review

E Kodhai *et al.* [1] proposed a Light Weight Hybrid approach which is a combination of textual and metrics based techniques. This technique was designed to detect method level syntactic and semantic clones. A tool CloneManager was developed by authors for implementation of proposed approach. Using proposed technique all four types of clones were detected with improvement in Precision and Recall values.

G. R. Goda *et al.* [2] designed a novel clone detection technique which used metrics and template conversion to achieve the target. An analysis was done based on accuracy, precision, and recall and found that improved results came as compared to previous methods.

Y. Yuan *et al.* [3] proposed an approach named Boreas for code clone clusters detection. This approach is based on token based technique. Authors conducted experiments on JDK 7 and Linux kernel 2.6.38.6 and results were found efficient as compared to Deckard tool which was based on syntactic technique.

T. Kamiya *et al.* [4] proposed an approach based on the transformation of input source text and token by token comparison. For implementation purpose authors developed a tool called CCFinder which can detect clones in C, C++, COBOL, Java and other source files. Authors also proposed metrics to find out desired clones. After experiments it was found that the proposed tool worked fine as compared to many others which developed earlier.

F. Deissenboeck *et al.* [5] proposed an algorithm which is based on graph theory, so it can be applied to graphical data-flow languages and could detect clones on models based graph structures. This algorithm was applied in an industrial case study with MAN Nutzfahrzeuge using Matlab/ Simulink/ TargetLink models. This approach could work on large scale project (20,000 elements) and fond clones easily.

T. Ying *et al.* [6] developed a function clone generator extractor named cGen which can detect clones across multiple versions. The proposed technique can detect type-1 and type-2 function clone genealogies. Using this tool authors analyzed nine open source C projects and found that cGen worked efficiently across multiple versions.

E. Kodhai *et al.* [7] proposed a light weight technique which was a combination of metrics and simple textual analysis. The proposed method can detect clones at functions level. This technique performed better in term of higher amount of recall and found functional clones in C language.

W. Li *et al.* [8] proposed a metric vector-based code clone detection method using the function-calling tree. The method was tested in C language. Scientists performed the experiment with two programs and proposed approach was found more reliable as compared to traditional methods.

3. Proposed Method

We have studied various clone detection techniques. The existing method [7] can detect only function level clones. In order to detect module or file level clones, we have proposed a method by adding a few metrics. The pseudo code of this

DOI: 10.18535/ijecs/v5i9.72

Method is given in figure 1. The proposed method is based on software metrics which are following:

1) Effective Lines of Code
2) No. of Classes
3) No. of Methods
4) No. of return Statements
5) No. of if Blocks
6) No. of else Blocks
7) No. of while Loops
8) No. of for Loops
9) No. of File/ Module Level Variables

Whereas the existing technique uses seven metrics which are given below:

1) Effective Lines of Code
2) No. of Methods
3) No. of return Statement
4) No. of if Blocks
5) No. of for Loops
6) No. of while Loops
7) No. of Variables

```
Proposed Algorithm:

Input (Source, Destination)
BEGIN
    Clones=null, metricsSrc=null, metricsDst=null;
1. Calculation of metrics
    For i=0 to Source.EOF-1
        metricsSrc.add()
    For j=0 to Destination.EOF-1
        metricsDst.add()
    length=Max(metricsSrc, metricsDst)
2. Detection of Clones
    For k=1 to length
        if(metricsSrc==metricsDst)
        Clones.add()
    return Clones
END
```

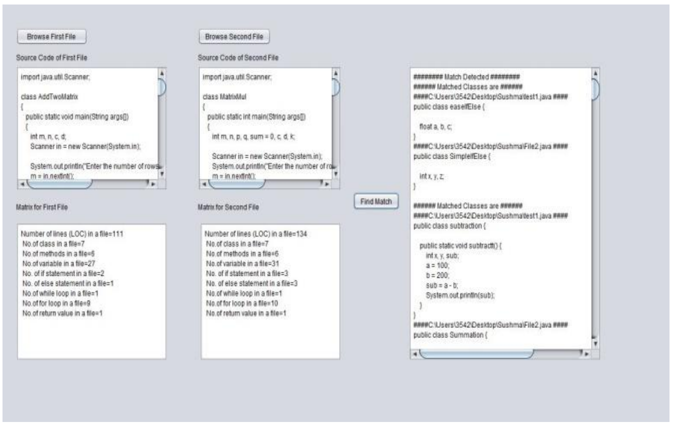
Figure 1. Proposed Algorithm

4. Implementation

For implementation purpose, we have developed a tool named JSCCD (JS Code Clone Detector) which is based on the algorithm shown in figure 1. The tool is developed in Java language and snapshot of the tool is shown in figure 2. Using this tool we performed experiments with existing [7] and proposed technique and found the clones which are shown in table 1. These experiments clearly show that our results are better than existing technique in terms of precision and recall values which are obtained by using the equations (1) and (2) described in section 4.1. The results comparison in the account of precision and recall are shown in figure 3 and 4 in section 4.1.

Table 1. Comparison of Clones Found

Clones Found	Using Existing Technique	Using Proposed Technique
Effected Lines	**74**	**83**
Classes	**0**	**5**
Methods	**4**	**1**
Loops Block	**7**	**7**
Condition Block	**3**	**5**
Variables	**16**	**16**



```
import java.util.Scanner;

class AddTwoMatrix
{
  public static void main(String args[])
  {
    int m, n, c, d;
    Scanner in = new Scanner(System.in);

    System.out.println("Enter the number of rows
    m = in.nextInt();
```

```
import java.util.Scanner;

class MatrixMul
{
  public static int main(String args[])
  {
    int m, n, p, q, sum = 0, c, d, k;

    Scanner in = new Scanner(System.in);
    System.out.println("Enter the number of ro
    m = in.nextInt();
```

```
Number of lines (LOC) in a file=111
 No.of class in a file=7
 No.of methods in a file=6
 No.of variable in a file=27
 No. of if statement in a file=2
 No. of else statement in a file=1
 No.of while loop in a file=1
 No.of for loop in a file=9
 No.of return value in a file=1
```

```
Number of lines (LOC) in a file=134
 No.of class in a file=7
 No.of methods in a file=6
 No.of variable in a file=31
 No. of if statement in a file=3
 No. of else statement in a file=3
 No.of while loop in a file=1
 No.of for loop in a file=10
 No.of return value in a file=1
```

```
######## Match Detected ########
###### Matched Classes are ######
####C:\Users\3542\Desktop\Sushma\test1.java ####
public class easeIfElse {

  float a, b, c;
}
####C:\Users\3542\Desktop\Sushma\File2.java ####
public class SimpleIfElse {

  int x, y, z;
}

###### Matched Classes are ######
####C:\Users\3542\Desktop\Sushma\test1.java ####
public class subtraction {

  public static void subtractt() {
    int x, y, sub;
    a = 100;
    b = 200;
    sub = a - b;
    System.out.println(sub);
  }
}
####C:\Users\3542\Desktop\Sushma\File2.java ####
public class Summation {
```

Figure 2. Clones and Metrics Detected by JSCCD Tool

4. 1 Performance Measure

For code clone detection, the accuracy of results can be considered by the combination of two parameters precision and recall. We can obtain the precision and recall by the equations given below [9]:

$$Precision\ (P) = \frac{\text{Number of Clones Correctly Found}}{Total\ Number\ of\ Clones\ Found} \quad (1)$$

$$Recall\ (R) = \frac{\text{Number of Clones Found Correct}}{Total\ Number\ of\ Clones\ in\ the\ Source\ Code} \quad (2)$$

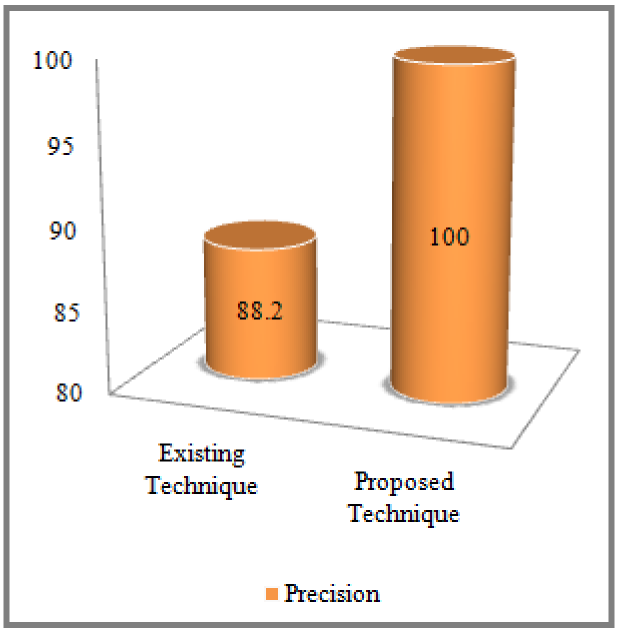


Figure 3. Comparison of Precision

DOI: 10.18535/ijecs/v5i9.72

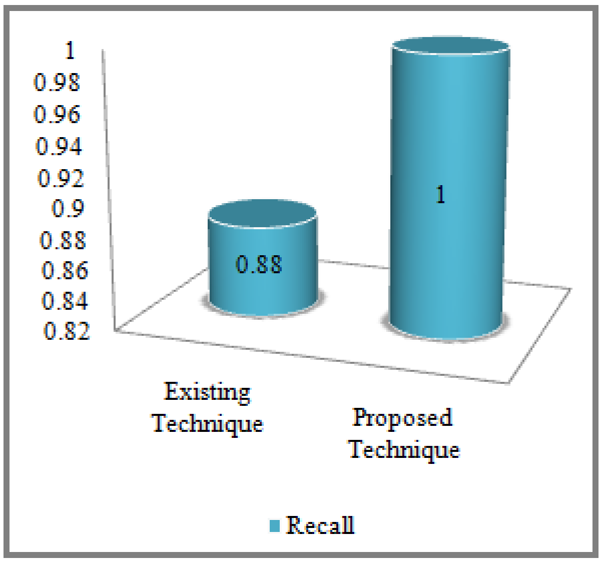


Figure 4. Comparison of Recall

5. Conclusion

Our paper proposes an enhanced metric based technique which is used to detect clones between Java files. The existing technique is able to detect method level clones only and the proposed technique detects module level clones using several metrics. Our technique is useful to find out type1 and type2 clones in Java based software systems. For implementation purpose, we have developed a tool in Java named as JSCCD (JS Code Clone Detector) which can detect clones with more accuracy in terms of Precision and Recall. After experiments, we found that using existing technique the precision value is found 88.2% and recall value 0.88 while the proposed method gives the precision 100% and recall 1.

For future work, the experiments can be performed at large scale. Our technique can be enhanced using soft computing methods for optimization purpose. The proposed technique can also be enhanced using a hybrid approach of two or more techniques for improvement of the results.

References

[1] E. Kodhai and S. Kanmani, “Method-level code clone detection through LWH(Light Weight Hybrid) approach,” Journal of Software Engineering Research and Development, pp. 1-29, 2014.

[2] G. R. Goda and A.Damodaram, “An Efficient Software Clone Detection System based on the Textual Comparison of Dynamic Methods and Metrics Computation,” International Journal of Computer Applications, Vol. 86, No 6, pp. 41-45, 2014.

[3] Y. Yuan and Y. Guo, “Boreas: An Accurate and Scalable Token-based Approach to Code Clone Detection,” *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, New York, pp. 286–289, 2012.

[4] T. Kamiya, S. Kusumoto and K. Inoue, “CCFinder: A multi-linguistic token based code clone detection system for large scale source code,” IEEE Transactions on Software Engineering, pp. 654-670, 2002.

[5] F. Deissenboeck and J. Girard, “Clone Detection in Automotive Model-Based Development,” ACM, pp. 10-18, 2008.

[6] T. Ying, Z. Li-ping, W. Chun-Hui and L. Dong-sheng, “Extract Function Clone Genealogies across Multiple Versions,” International Journal of Security and Its Applications, Vol. 9, No. 6, pp. 167-182, 2015.

[7] Kodhai. E, Kanmani. S, Kamatchi. A and Radhika. R, “Detection of Type-1 and Type-2 Clones Using Textual Analysis and Metrics,” ITC, IEEE, 2010.

[8] W. Li, D. Li, C. Qiu and J..Hou, “Efficient Metric Vector-Based Code Clone Detection Using Function-calling Tree,” International Journal of Hybrid Information Technology, Vol. 8, No.11, pp.139-150, 2015.

[9] A. kumar, C. R. K. Reddy and A. Govardhan, “An Efficient Method-Level Code Clone Detection Scheme Through Textual Analysis Using Metrics,” International Journal of Computer Engineering and Technology, Vol. 3, No. 1, pp. 273–288, 2012.

[10] D. M. Shawky and A. F. Ali, “An approach for assessing similarity metrics used in metric-based clone detection techniques,” *3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, Vol. 1, pp. 580–584, 2010.

[11] Kodhai.E, Perumal.A, and Kanmani.S, “Clone Detection using Textual and Metric Analysis to figure out all Types of Clones,” International Journal of Computer Communication and Information System,Vol 2, No1, pp. 99-103, 2010.

[12] Y. Higo, S. Kusumoto and K. Inoue, “A Metric-based Approach to Identifying Refactoring Opportunities for Merging Code Clones in a Java software system,” Journal of Software Maintenance And Evolution: Research And Practice, pp. 435-461, 2008.

Authors Profile

Sushma graduated with B.Tech and presently she is pursuing M.TECH (CSE) in Guru Jambheshwar University of Science & Technology, Hisar, India. Her area of interests includes Software Engineering.

Jai Bhagwan received the M.TECH degrees in Computer Science and Engineering from Maharishi Markandeshwar University, Mullana in 2011. After this, he stayed in Maharishi Markandeshwar University, Mullana on the post of Lecturer in Information Technology department for a period of one year.
Currently, he is working as Assistant Professor in the Computer Science & Engineering Department in Guru Jambheshwar University of Science & Technology, Hisar, India. He has more than 5 years of teaching experience. His areas of research are Cloud Computing and Software Engineering.