Research Methodology on Code Clone Detection with Refactoring Using Textual and Metrics Analysis in Software

Dr.G.Anil Kumar
Sr. Assistant Professor CSE MGIT Hyderabad T.S. India

Abstract The act of cut, copy and pasting code fragments and making minor, non–functional alterations, is a major problem for large, industrial software systems. It leads to duplicated code fragments known as code clones. The major consequence of cloning in the code is that it makes the maintenance process difficult. Finding out the reused fragment of code in any application is usually called as code clone detection. In the process of software maintenance, evolution clones are considered to be harmful because it increases the complexity of the system. From the evolution of the clone detection, it provides improved results and decreases the complexity of the system for better maintenance.

Through the state of art in code cloning, one can understand clone detection process is mainly focused on detection of a line after line or detection based on tokenization. This technique makes the system complex and takes long time to process the source code to find the clones in it. If a code fragment is not an exact copy, but the functionality shows that it is similar to another code fragment, then current clone detection system unable to find out such type of clones.

The proposed research model for detection of clone approach shows that the detection process is easier and it has produced efficient results. This approach is a process of combining textual approach and metric analysis of the given source code for detection of all four types of clones presented in a given set of code fragment in java source code. All the detected clone pairs are grouped together to form clone clusters and they are stored in files. All the detected clones can be automatically refactored if it is required by the programmer.

Different semantics have been formulated and the values of these semantics have been used in the process of clone detection. These metrics along with textual analysis provide a very less complexity in figuring out the clones and provide accurate results.

Efficiency of the technique is measured in terms of Precision and Recall values. The results of the proposed method are compared with the bench mark tools like Clone DR, CCFinder and other techniques. The analysis of the experimental results shows that Precision and Recall values are improved and they are better than the previous techniques.

Keywords — Code Clone, Clone detection, refactoring, metrics, textual analysis,

I. INTRODUCTION, OVERVIEW, CONCLUSION OF RESEARCH WORK AND FUTURE ENHANCEMENTS

Through the state of art in code cloning, one can understand clone detection process is mainly focused on detection of a line after line or detection based on tokenization. This technique makes the system complex and takes long time to process the source code to find the clones in it. If a code fragment is not an exact copy, but the functionality shows that it is similar to another code fragment, then current clone detection system unable to find out such type of clones.

The proposed research model for detection of clone approach shows that the detection process is easier and it has produced efficient results. This approach is a process of combining textual approach and metric analysis of the given source code for detection of all four types of clones presented in a given set of code fragment in java source code. All the detected clone pairs are grouped together to form clone clusters and they are stored in files. All the detected clones can be

ISSN: 2348 – 8387  www.internationaljournalssrg.org  Page 24
automatically refactored if it is required by the programmer.

Different semantics have been formulated and the values of these semantics have been used in the process of clone detection. These metrics along with textual analysis provide a very less complexity in figuring out the clones and provide accurate results.

Efficiency of the technique is measured in terms of Precision and Recall values. The results of the proposed method are compared with the benchmark tools like Clone DR, CCFinder and other techniques. The analysis of the experimental results shows that Precision and Recall values are improved and they are better than the previous techniques. CONCLUSIONS

Present Work:

Software maintenance is a very important phase of the development life cycle. Reducing the software maintenance overhead is an activity that makes the software industry more comfortable. When it is compared with any other product the software product customers expect more of maintenance of the product because they feel it is flexible (i.e. it is simply writing few instructions). So, changes can be accommodated easily at any time. But software engineering literature says it is a myth.

Software clone is more dangerous in large software systems. Most of the times cloning happens due to copy and paste activity only. Almost every developer thinks to save developing time, so he uses this activity because developing code from the scratch takes more time. Sometimes time constraints force developers to turn towards cloning. Some maintenance engineers accidentally produce these clones. Although it seems to be an effective and simple solution to the developer’s problems, usually these cloning activities are documented and it leads to number of negative effects on the quality of the software. It increases the total number of lines of code of the system and lines of code that needs to be maintained.

Clone detection is an ongoing research area and the existing literature is overwhelmed in detecting and eliminating clones from software systems. The literature presented in literature survey topic of this research work gave several dimensions of code cloning. Many existing methods and tools have been compared and discussed. It is very important to identify the clones present in the code, at the same time there should be some solution proposed to this problem. The existing refactoring methods can give solutions to the code cloning problems. Refactoring has been used effectively in the proposed method.

In this work, a light-weight method has been proposed to identify functional clones. This method uses the computation of several metrics in combination with simple textual analysis technique. The usage of metrics with existing exponential rate of comparison overhead of the other methods is reduced to minimum number of comparisons. This is possible by early analysis of potential clones and applying comparisons only on code fragments that are identified as clones in this analysis. Since the string matching/textual comparison is performed over the short listed candidates, a higher amount of recall could be obtained.

The Proposed work is divided into two stages. The first one is selection of potential clones and the second one is comparison of potential clones. The proposed technique detects exact clones on the basis of metric match and then by text match. Potential clones are compared line-by-line to determine whether two potential clones really are clones of each other. The experiments proved that this method can do better than existing systems in finding and classifying the clones in JAVA. The Precision and Recall values that are obtained describe the efficiency of the work proposed. It has been proved that Precision 98% and Recall 96% is achievable in code cloning. In addition it also identifies the functional clones.

Future Enhancements

Though the proposed technique is working efficiently for Programming languages like JAVA, it can be extended to find clones in multiple languages. When it comes to identify only type I, type II and type III clones this method can identify clones in almost all object oriented programming languages. Research work can be extended not only to find the clones but also to remove the actual clones. Though refactoring process has been used, it can be fully automated and implemented so that no human intervention is required.

The proposed method is experimented on medium sized software applications only. These applications are of 10 to 15 KLOC only. Experiments on large scale systems can be conducted to observe efficiency of the method. The parameters for the efficiency are taken only in the form of precision and recall values. It also can be extended to scalability, portability and robustness etc.

II. APPENDIX

Programs of the case study are presented earlier. This is a class which is taken from a package where only few methods were present. These classes are taken because the methods which are present in these classes are useful to demonstrate the code duplication.

Program 1

```java
import java.util.*;
import java.lang.*;
```

public class TestFileOne{
    int p,q=1,r;
    double VALUE; /* the number which we
    need to find factorial */

    public int factorial(int n)   { /* factorial
        function using recursive function */
        if(n == 0)   {
            return 1;
        } else {
            return n * factorial(n-1);
        }
    }

    public int gcdOne(int a, int b) {
        while (b != 0) {
            if (a > b) {
                a = a - b;
            } else {
                b = b - a;
            }
        }
        return a;
    }

    public int mul(int a, int b){
        int n = 0, p=0;
        p=p+1;
        for(int i = 0; i < b; i++) {
            n += a;
        }
        return n;
    }

    public int factorial1 ( int VALUE ){ /* factorial
        using for loop */
        for (p=1; p<=VALUE; p++)
            q = q*p;
        return q;
    }
}

This program is similar to the program 1 with few changes in it

Program 2

import java.util.*;
import java.lang.*;

public class TestFileTwo {
    public int factorial2 (int n){ /* factorial using
        recursive function */
        if(n == 0)   {
            return 1;
        } else {
            return n * factorial2(n-1);
        }
    }

    public int gcdTwo(int c, int d) {
        while (d != 0) {
            if (c > d) {
                c = c - d;
            } else {
                d = d - c;
            }
        }
        return c;
    }

    public double mulTwo(double a, long b)
    {
        double n = 0.0;
        for(long i = 0l; i < b; i++)
            n += a;
        return n;
    }
}

REFERENCES


