Java Clone Finder: Cost Efficient Syntax Tree Based Code Clone Detection in Java

Shaikh Farooque Ahmed[1] Sawatantra Chauhan[2] Prashant Upadhyay[3] Mohammed Ahmed[4]
[1,2,3]Student [4]Assistance Professor
[1,2,3,4]Department of Computer Engineering
[1,2,3,4]MHSSCOE, Maharashtra, India

Abstract— Software development life cycle does not end with the development of software. The maintenance of software follows the development of software. Usually a large amount of money is spent on the maintenance purpose which needs to be reduced. A software system might contain a lot of duplicate fragments of code which might mean duplicate bugs. Also, duplicate code fragments might make software design sloppy. Thus there is a need to reduce or eliminate the code clones for reducing the maintenance cost. This paper proposes JAVA CLONE FINDER, which is a software capable of detecting code clones in the source code .It works by creating abstract syntax tree of the source code. It does this by fetching the syntax tree from eclipse compiler and then it displays the common syntax tree via some appropriate GUI. JAVA CLONE FINDER is a easy to use and a cost efficient solution for software maintenance activities.

Key words: Code Clone, Eclipse, Maintenance, Cost Efficient, Java

I. Introduction

Software systems must be durable, reliable and long lasting. For these properties to exist, the maintenance of the software system must be proper. Software maintenance activities mostly occur after the deployment of software. Previous studies have shown that more than 60% of software cost is spent on maintenance of the software. For most software systems, maintenance cost is more than the development cost. Software maintenance is defined as the totality of activities that are required to provide cost effective support to a software system. Thus, the cost of maintenance must be decreased in order to make the software system more successful.

With the evolution of software systems, a large amount of code tends to get repeated. The developers usually ignore these repeated code fragments. These code clones are a result of different programming styles, copy/paste programming etc. and they degrade the code quality. Also, a certain code fragment might contain a defect and by finding duplicate code fragments, the defects at all occurrences can be fixed.

Studies have shown that 7% to 23% of software systems contain duplicate code fragments. Detection of these code clones with JAVA CLONE FINDER can drastically improve the maintainability of the software and thereby lessen the maintenance cost.

II. Scope

For a software project to be successful, it necessary to satisfy all the requirements of the user and make the user satisfied. Therefore we describe the scope of the project which should be accomplished within a given deadline. If it software system has all the mentioned requirements then the system will be considered a success. Code clones can be classified into four different types. The first type of code clone is an exact copy of a code fragment with white spaces and comments as exception. The type two of code clones, are syntactically equal but contain small modifications such as renaming methods and variables. The third type of code clones consist of copied code fragments with slight syntactical differences. The fourth type of code clone consist of functionally equivalent but implementation wise different code fragments.

JAVA CLONE FINDER can detect type-1 and type-2 code clones.

III. Existing System

Some code clone detection methods use text based searching and matching while some are based on lexical analysis and token matching .A few code clone detection techniques and their drawbacks are mentioned below.

A. Line Based Technique:

This technique can only detect code clones by comparing source code on a line and before the comparison, the tabs and white spaces are eliminated. This is an old method and it has a very low detection accuracy and is not suited for different styles of programming. For example-even if '{' position of if or while loop changes it cannot detect the code clone and also it cannot detect code clones having different variable names.

B. Metric Based Techniques:

Here, the source code is divided into different functional units and metrics are defined for each unit. Then the units having similar metric value are defined as code clones. The drawback of this technique is its poor performance due to the high sensitivity of metrics to small changes which can cause the similar units not to get detected.

C. Token Based Techniques:

This technique is based on simple string matching, it was the first approach of code clone detection using sequence matching algorithms. The source code is firstly tokenized to give a sequence of tokens as input to the string matching algorithms. After tokenization process, token sequences are then compared to find the similar sub-sequence as code clones. The drawback of this technique is that it does not offer scalability.

D. PDG (Program Dependency Graph) Based Techniques:

Here, control and data flow dependency of a function is represented by a program dependency graph and code clones can be identified as isomorphic sub-graphs. The accuracy of clone detection is very high but the drawback is that it requires complex computations which are difficult to apply to large software systems.

IV. Proposed System

In the proposed system "JAVA CLONE FINDER" is able to detect duplicate code fragments in a java source code using the eclipse platform.

A. *The Operation of the Proposed System:*

When the java source code is compiled by an eclipse compiler, the abstract syntax tree of corresponding source code is generated by an eclipse parser. JAVA CLONE FINDER then fetches this syntax trees from the eclipse compiler and these abstract syntax trees are compared with each other thereby producing common trees. With respect to the common trees the corresponding result is shown to user.

B. *Details of the Hardware and Software:*

1) *Hardware Requirements:*

- Pentium-IV or above processor with clock speed 1.6 HZ
- 1GB RAM
- 10 GB hard disk space

2) *Software Requirements:*

- Eclipse
- Windows Operating System
- Prefuse Tree Map

V. Methodology

This methodology consists of input module which is our source code, in which we have to detect common code which gets parsed by eclipse compiler thereby generating abstract syntax trees, the code clone algorithm is then applied on these abstract syntax trees by which we get common tree which gets displayed by a visualiser. The visualiser is a component which displays the common code in a way that makes it easy for a user to understand and distinguish codes. Lastly the PrefuseTreeMap is used for displaying the code clone which uses swing as graphical user interface API. As eclipse is implemented in SWT TreeMapViewer is used as a glue-component which enables integration between eclipse architecture and external tool PrefuseTreeMap.

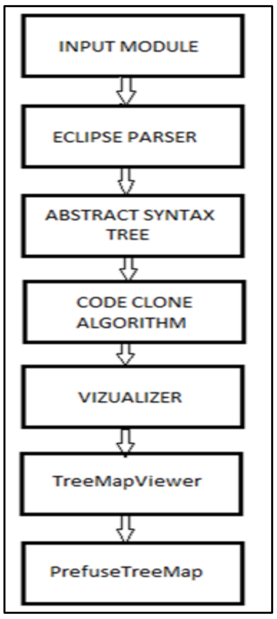
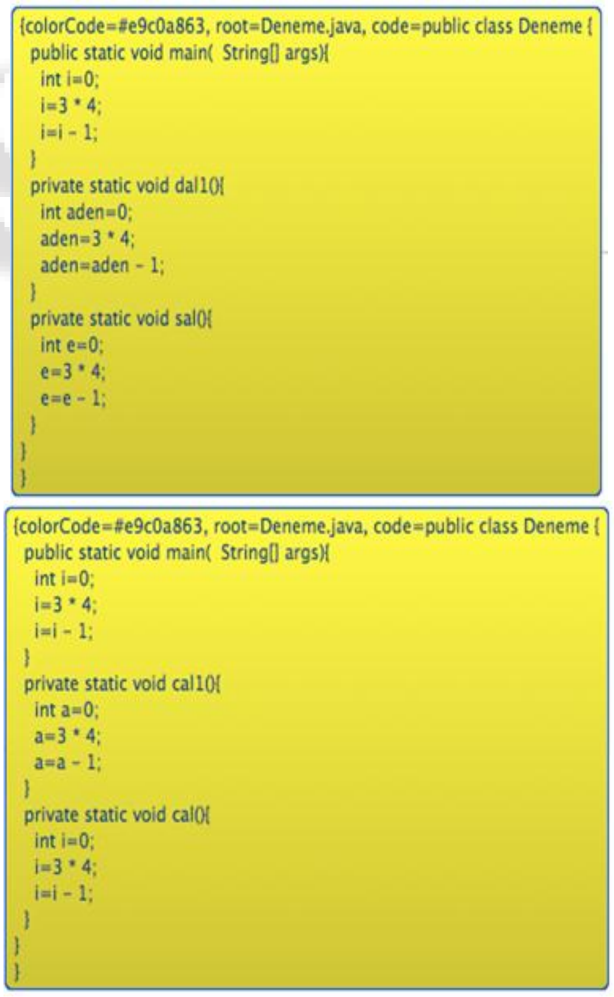


Fig. 1:

VI. Experiments and Results

We are able to achieve the goal of detecting common code by making this tool JAVA CLONE FINDER. However, there is no empirical data present about the speed and performance of JAVA CLONE FINDER yet, because of multi-thread concurrent programming model, but due to the use of hash algorithm in normalizing software code the creation of common clone information is usually done in short time frames. However, external tools for visualizing code clones may cause some performance problems due to following two reasons:

1) The first reason that might cause some performance issues is the external tree map viewer which creates xml formations of given data set interface. This is an expensive operation for large java projects and takes a lot of processing time to complete. The user of the system might have latencies during the launch operation of this integrated external tree map viewer.
2) The second reason is the external PrefuseTreeMapis originally implemented in Swing and integrated into Eclipse with the help of SWT to AWT Bridge which may cause some inconsistencies. On the other hand PrefuseTreeMap presents useful information on detected software clones.



```
{colorCode=#e9c0a863, root=Deneme.java, code=public class Deneme {
  public static void main( String[] args){
    int i=0;
    i=3 * 4;
    i=i - 1;
  }
  private static void dal1(){
    int aden=0;
    aden=3 * 4;
    aden=aden - 1;
  }
  private static void sal(){
    int e=0;
    e=3 * 4;
    e=e - 1;
  }
}
}
```

```
{colorCode=#e9c0a863, root=Deneme.java, code=public class Deneme {
  public static void main( String[] args){
    int i=0;
    i=3 * 4;
    i=i - 1;
  }
  private static void cal1(){
    int a=0;
    a=3 * 4;
    a=a - 1;
  }
  private static void cal(){
    int i=0;
    i=3 * 4;
    i=i - 1;
  }
}
}
```

Fig. 2: Gives A Simplified Example View For Given Code Clones Of Type-2 Classes.

VII. ADVANTAGES

- Easy to operate.
- Provides results quickly.
- Has a user friendly interface.
- No external component for compiling purpose is required.

VIII. CONCLUSION

JAVA CLONE FINDER is software which can used to reduce the efforts in maintenance of software systems by finding duplicate code fragments with relatively high accuracy as compared to previously used software systems, thereby reducing the maintenance cost of software systems.

IX. ACKNOWLEDGEMENT

We are thankful to Mohammed Haji Saboo Siddik College of Engineering, for providing us the opportunity to do constructive work. We are also thankful to anonymous reviewers for their constructive suggestions.

REFERENCES

[1] Baxter, I.D. et al., 1998. Clone detection using abstract syntax trees.
[2] Brooks, F., 1975.Addison-Wesley Pub. Co.
[3] Eclipse.org, 2009a. About the Eclipse Foundation.
[4] Eclipse.org, 2009b. Eclipse Java development tools (JDT).
[5] Eclipse.org, 2009c. PDE.
[6] Friedman, D., 2008. Essentials of programming languages 3rd ed.
[7] Grubb, P., 2003. Software maintenance: concepts and practice 2nd ed.
[8] Horch, J., 2003. Practical guide to software quality management 2nd ed.
[9] ISO/IEC, 2008. ISO12207 Systems and software engineering -- software life cycle processes, Geneva: ISO/IEC-IEEE.
[10] ISO/IEC, 2006. ISO14764 Software Engineering -- Software Lifecycle Processes --Maintenance, New York, NY: Institute of Electrical and Electronics Engineers.
[11] Lehman, M., 1985. Program evolution: processes of software change.
[12] Li, Z. et al., 2006. CP-Miner: Finding copy-paste and related bugs in large-scale software code.