ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

Techniques and Algorithm for Clone Detection and Analysis

Snehal R. Jadhav[1], Sachin B. Wakurdekar[2]

M. Tech Student, Department of Computer Technology, BVDUCOE, Pune, India[1]

Assistant Professor, Department of Computer Technology, BVDUCOE, Pune, India[2]

ABSTRACT: For maintenance and development of source code, several studies have exposed that the replica of a code or code clone in software technology are possibly risky. While this is the severe difficulty within software industry, throughout refactoring, there is small bit support for removing software clones. A large demanding difficulty is association and inclusion the replicated code, particularly later starting introduction to the software clone they are going through the numerous alterations in them. This paper presents a novel algorithm in which a couple of clone is mechanically reviewed and exclusive of altering the agenda performance that clone couple is re-factored securely. The differentiations shown in the clones are studied in this approach and those are securely parameterized with no incidence of any side cause. Novel of the gain of this approach is that the insignificant computational expenditure. Lastly, a large-scale experiential study has been carried out on above a million clone couples noticed and this discovery is completed by four dissimilar clone detection tools. This has been conducted in nine open source projects for supplying how re-factorability is exaggerated by dissimilar clone assets and tool arrangement selections.

KEYWORDS: Code duplication, Software clone management, Clone refactoring, Re-factorability assessment, Empirical study.

I. Introduction

Within several software industries, the complex problem has been acknowledged that the source code replication is grave difficulty. Code replication has awful result on the software product protection as well as development [1], [2]. In most recent few years dissimilar study areas have industrial numerous practices which are capable to discover and examine the replicated code [3]. along with latest study is concentrated on the clone managing actions [4], which comprise detecting clones in the past of a project, examining the reliability of alterations to the clones, renew additionally clone set as the assignment develop, as well as assigning the priority to the refactoring of the clones. In the accumulation to beyond improvement effort, result of the replicated code on protection attempt as well as rate, error-proneness because of incompatible changes, software faults, change-proneness, and modify dissemination has been examined empirically by means of numerous studies. There is a requirement of software or hardware tools which can be mechanically investigate software clones to verify whether they can be securely re-factored with no altering the activities of the source code. One of the vital nevertheless lost characteristics from the clone organization is re-factorability examination. Whenever the programmers are responsive in investigating refactoring prospects for replicated code it can be used to clean the clones which can be openly re-factored. This is the technique by which maintainers can concentrate on the different sections of the source code which can instantly advantage from the refactoring, so hence thus reasons development in the defensibility. The paper encompasses an novel algorithm which takes two clone fragments as a input that are noticed from the several tool plus it pertains next three stages in order to verify that they can be re-factored with no several part causes or not.

Here we detect code cloning using code attributes which are given below:

- Number of line counts.
- Number of bracket counts.
- Number of import counts.

Stage 1: this stage discovers source code fragments within the same nesting arrangements inside those input clones that provide as possible as refactoring chances. If it divides general nesting arrangement then it considers that two code fragments could be incorporated, plus hence re-factored.

Stage 2: In this stage, algorithm discovers amapping among the comments of the source code fragments which enhance number of mapped declarations as well as reduces number of dissimilarity among the mapped declarations through discovering the investigate gap of substitute mapping results.

Stage 3: In third stage, the dissimilarity inside the reviews declarations which were noticed in the earlier stages is studied beside the set of prerequisites. This producer is completed in order to decide whether it can be parameterized with no altering the program performance or not.

The rest of this paper is ordered as follows: In section 2 we will explore the two core program structure. Section 3 defines problem statement of proposed system. Section 4 explores motivation behind clone detection. Section 5 define main objective of proposed system. Section 6 defines the proposed system with two different inputs for assessing the refactorability. We draw a conclusion in section 7.

II. RELATED WORK

Following are the two core program structures that are used in this approach:

1. Program Structure Tree.
2. Program Dependence Graph.

1. Program Structure Tree

This Program Structure Tree [5] was established by Johnson et al. as a hierarchical demonstration of program formation and this formation is stands on solo-entry solo-exit (SESE) areas of this control flow graph. The nesting association of SESE regions and chains of sequentially composed SESE regions are captured by essentially PST.

2. Program Dependence Graph

The Program Dependence Graph [6] is a directed graph which includes of different type of edges. In this PDG, the nodes indicate the declarations of a task or a scheme, plus the edges indicate manage as well as data flow dependencies among comments. In this way PDG illustration is used in two ways. In first way, the composite variables are introduced which represents the state of objects which are referred in body of method and it also creates data dependencies for these variables. In second way, two more types of edges are added in the PDG, which are helpful in the examination of preconditions. These two types of edges are: anti-dependencies and output dependencies.

III. MOTIVATION

The large complex difficulty within software industries is refactoring, but there can be small bit of support for removing software product clones. A big demanding difficulty is a association plus inclusion of the replicated source code, particularly behind first introduce to the software clone they are going by means of the number of alterations in that. In this novel algorithm it mechanically reviews whether a clone couple could be securely refectory whether as well as which is also with no altering a actions of the software source code.

IV. PROPOSED SYSTEM

There are two dissimilar types of input are given by means of this algorithm:

1) Two source code fragments are confirmed as clones by the clone discovery tool inside the structure of the identical technique, or dissimilar techniques.

2) Two technique statements measured to be replicated, or it can include replicated source code fragments anywhere within their structures.

Three main stages for reviewing the refactorability in this algorithm:

1) Nesting Structure Matching [1]:

In this first stage, nesting arrangement of the input clone fragments is investigated that is valuable in the discovering maximum isomorphic sub trees. It can unspecify that two source code fragments can be unified simply if they have an equal nesting arrangement. Every matched sub tree couple would be additional examined as the disconnect clone refactoring chance in the coming stages. When we used nesting arrangement matching function it will offer the replicated code as output by referring two files.

2) **Statement Mapping [1]**:

In this statements removed from the prior stage inside the sub tree couples are mapped in divide-and-conquer manner. They obtain benefit of equal nesting arrangement among the isomorphic sub trees; worldwide mapping difficulty is partition within slight sub difficulty. Equivalent Program Dependence sub graphs are mapped through regarding a Maximum Common Sub graph (MCS) algorithm for every sub-problem. These sub explanations are mutual to provide the worldwide mapping resolution within last part. When we hit it off on Statement mapping as well as Statistics then it will demonstrate Predictive Code Clone Detector Frame. It will also show every output by submitting two files.

3) Precondition Examination [1]:

Different set of conditions concerning the conservation of a program action are studied based on dissimilarity among the mapped comments in worldwide explanation, and also the declarations that can have not been mapped. If there is no conditions are desecrated, the clone fragments equivalent with the mapped declarations could be securely refactored, and therefore those are measured to be refactored.

Following Fig 1. Shows the stages declared above:

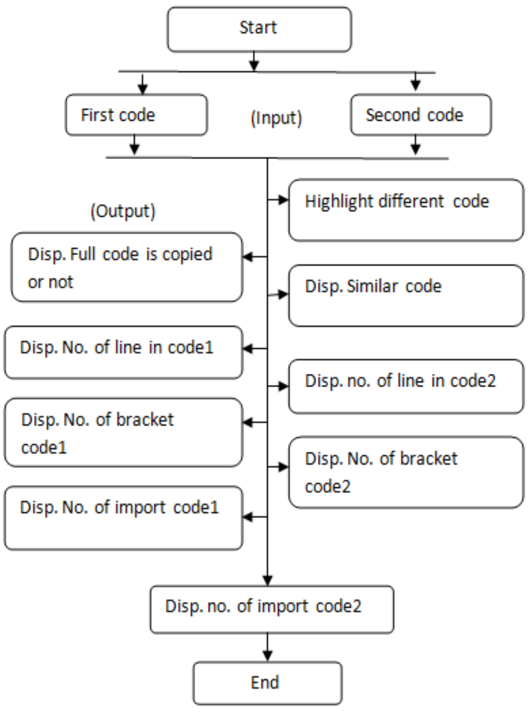

Start
First code
(Input)
Second code
(Output)
Highlight different code
Disp. Full code is copied or not
Disp. Similar code
Disp. No. of line in code1
Disp. no. of line in code2
Disp. No. of bracket code1
Disp. No. of bracket code2
Disp. No. of import code1
Disp. no. of import code2
End

Fig 1. Proposed Refactorability Analysis Approach.

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

Here we have calculated different number of lines, number of brackets as well as number of imports count by using two code fragments. It Also shows given code fragments are identical nesting structure or not.

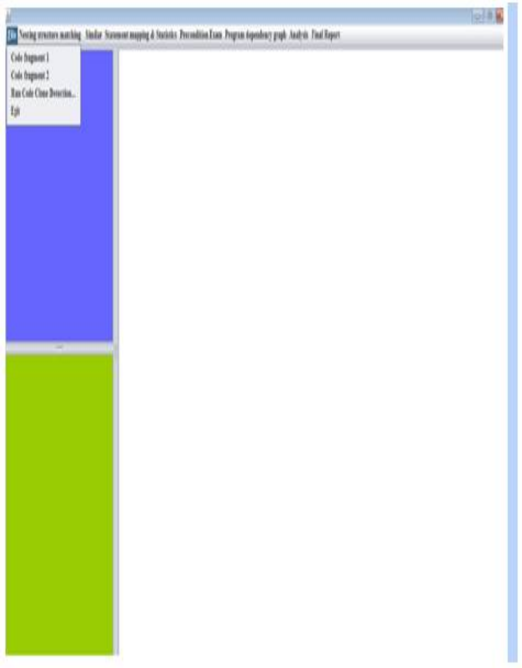
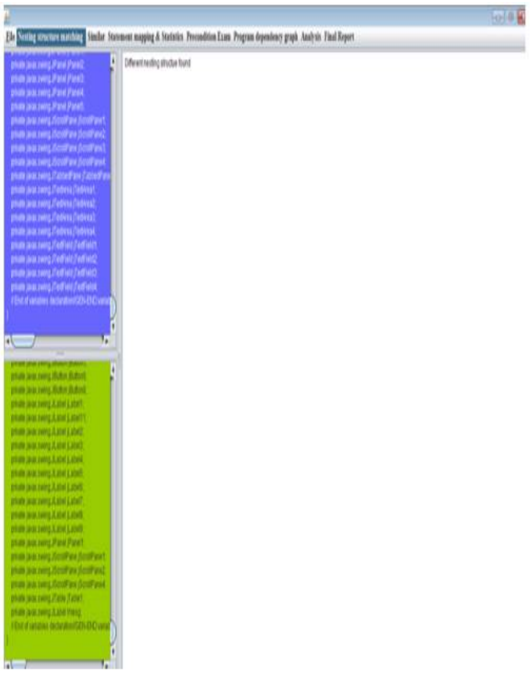


Fig.2. “File” - To select code fragment 1 & 2



Different nesting structure found

Fig.3. Nesting structure mapping”-to check selected codes copied or not.

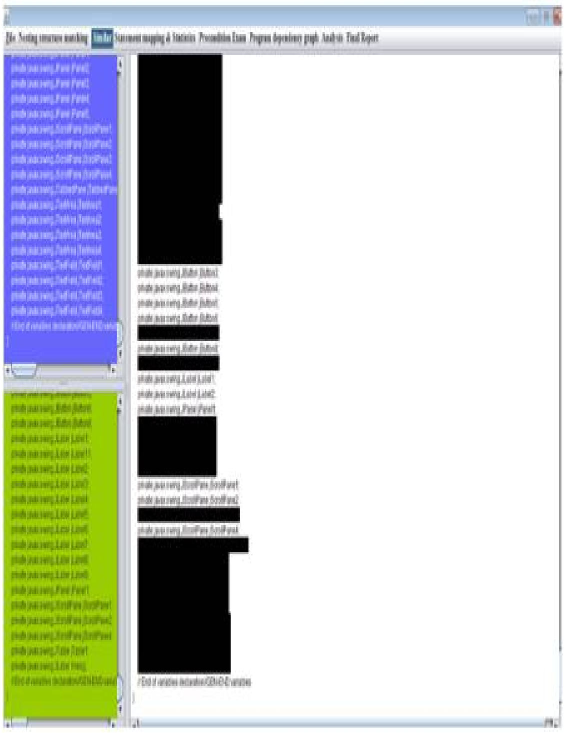
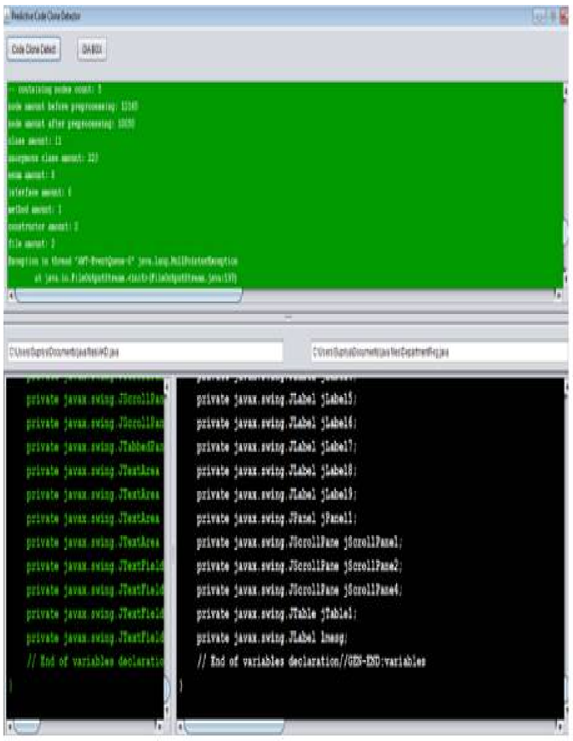


Fig.4. Similar – to check similarity



```
-- containing nodes count: 1
node amount before preprocessing: 1330
node amount after preprocessing: 1030
class amount: 11
anonymous class amount: 22
enum amount: 1
interface amount: 1
method amount: 1
constructor amount: 3
file amount: 2
Exception in thread "AWT-EventQueue-0" java.lang.NullPointerException
        at java.io.FileOutputStream.<init>(FileOutputStream.java:130)
```

```
    private javax.swing.JScrollPan
    private javax.swing.JScrollPan
    private javax.swing.JTabbedPan
    private javax.swing.JTextArea
    private javax.swing.JTextArea
    private javax.swing.JTextArea
    private javax.swing.JTextArea
    private javax.swing.JTextField
    private javax.swing.JTextField
    private javax.swing.JTextField
    private javax.swing.JTextField
    // End of variables declaratio
}
```

```
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JScrollPane jScrollPane4;
    private javax.swing.JTable jTable1;
    private javax.swing.JLabel lmsg;
    // End of variables declaration//GEN-END:variables
}
```

Fig.5. “Statement mapping & statistics”-to check all outputs

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016

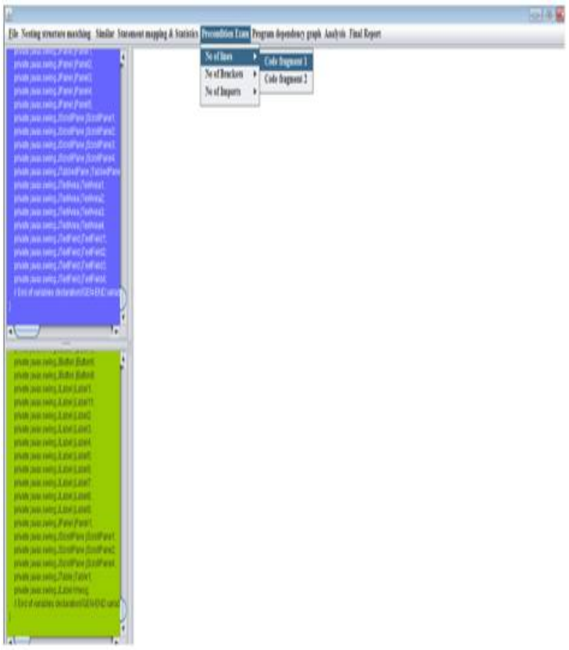
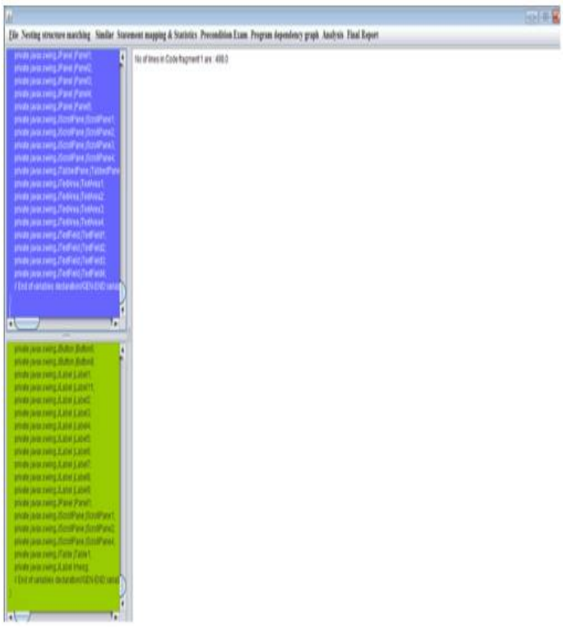
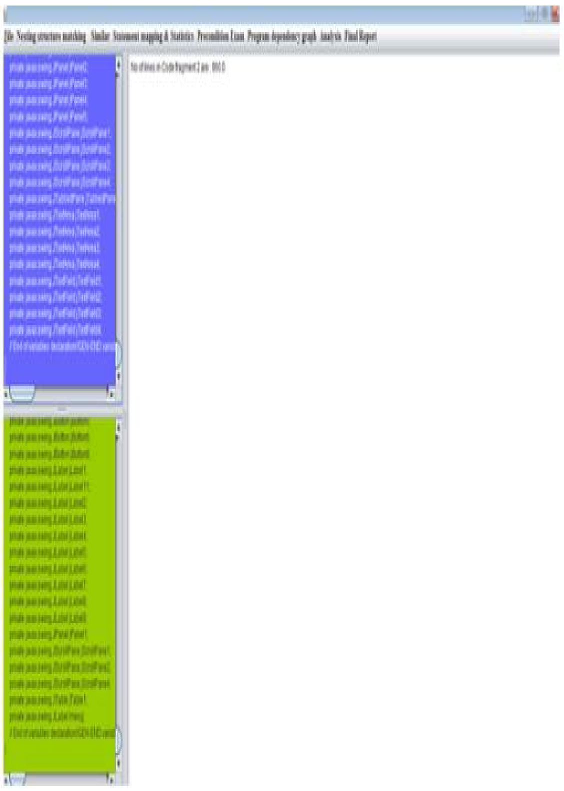

File Nesting structure matching Similar Statement mapping & Statistics Precondition Exam Program dependency graph Analysis Final Report
No of lines
No of Brackets
No of Imports
Code fragment 1
Code fragment 2

Fig.6. To check precondition Term


File Nesting structure matching Similar Statement mapping & Statistics Precondition Exam Program dependency graph Analysis Final Report

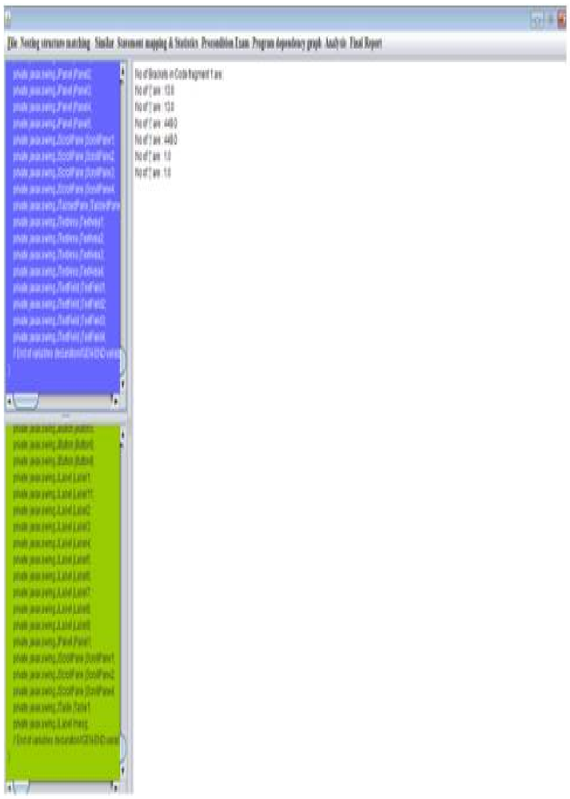
```
No of lines in Code fragment 1 are : 498.0
```

Fig 7. "No of line"- no of line count for code 1


File Nesting structure matching Similar Statement mapping & Statistics Precondition Exam Program dependency graph Analysis Final Report

```
No of lines in Code fragment 2 are : 866.0
```

Fig.8. "No of line"- no of line count for code 2


File Nesting structure matching Similar Statement mapping & Statistics Precondition Exam Program dependency graph Analysis Final Report

```
No of Brackets in Code fragment 1 are:
No of ( are : 13.0
No of ) are : 13.0
No of { are : 448.0
No of } are : 448.0
No of [ are : 1.0
No of ] are : 1.0
```

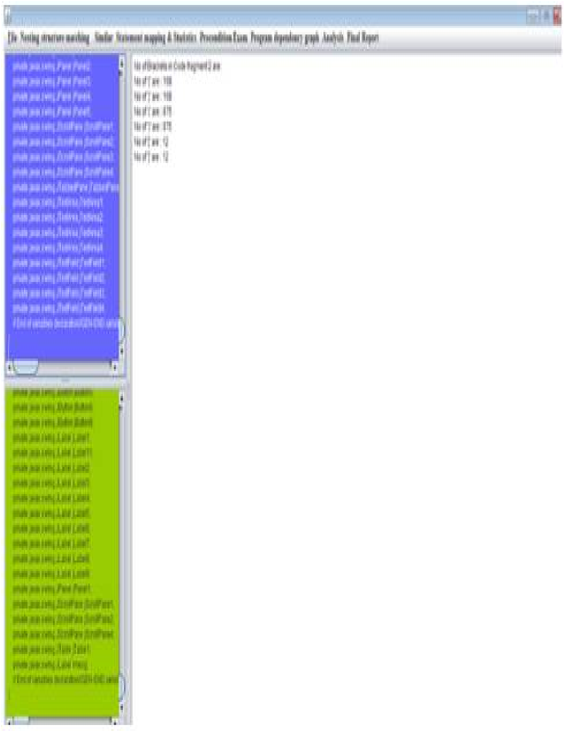
Fig.9. "No of bracket"- no of bracket count for code 1

ISSN(Online): 2320-9801
ISSN (Print) : 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

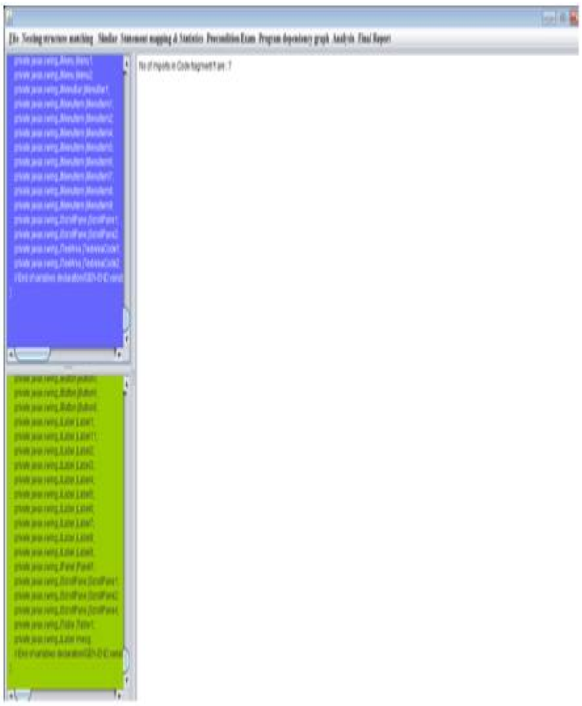
(An ISO 3297: 2007 Certified Organization)

Vol. 4, Issue 3, March 2016



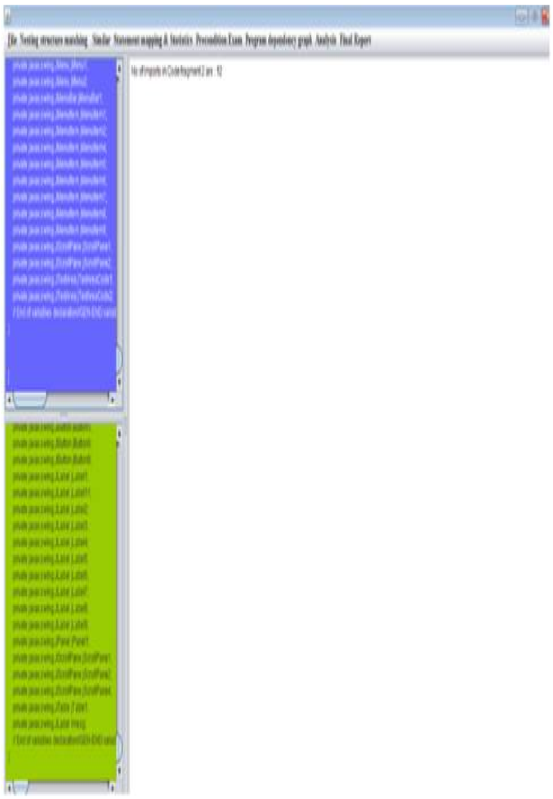
```
No of Brackets in Code fragment 2 are
No of { are : 118
No of } are : 118
No of ( are : 879
No of ) are : 879
No of [ are : 12
No of ] are : 12
```

Fig.10. “No of bracket”- no of bracket count for code 2



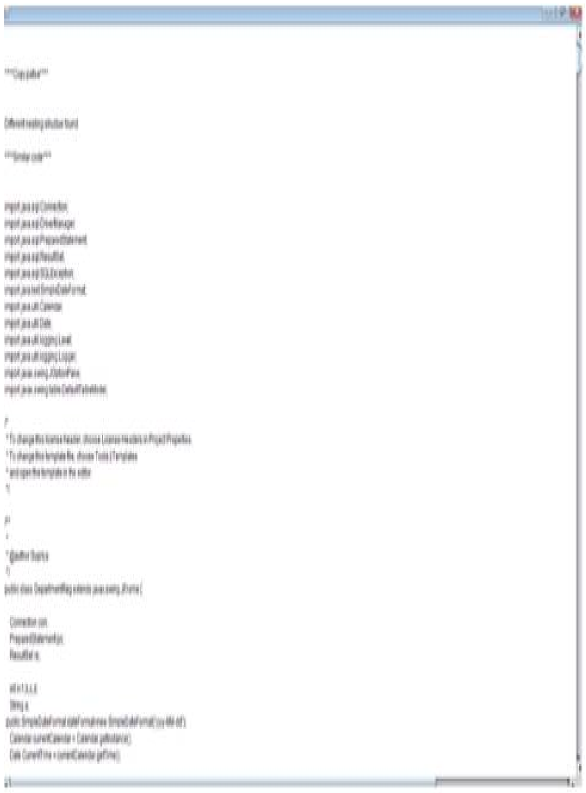
```
No of imports in Code fragment1 are : 7
```

Fig.11. “No of import”- no of import counts for code 1



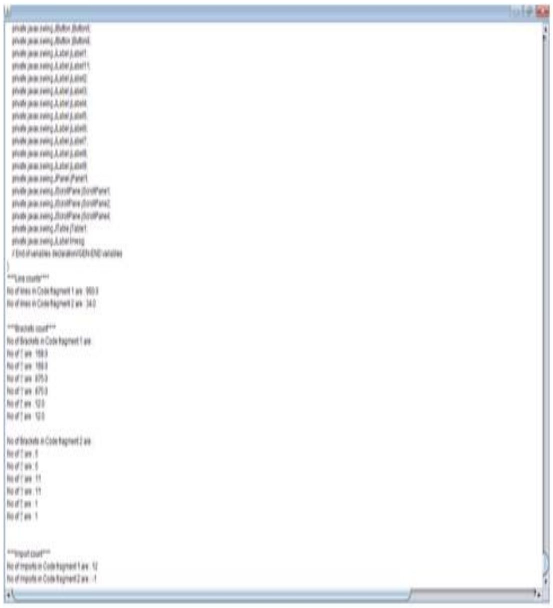
```
No of imports in Code fragment2 are : 12
```

Fig.12. “No of import”- no of import count for code 2.



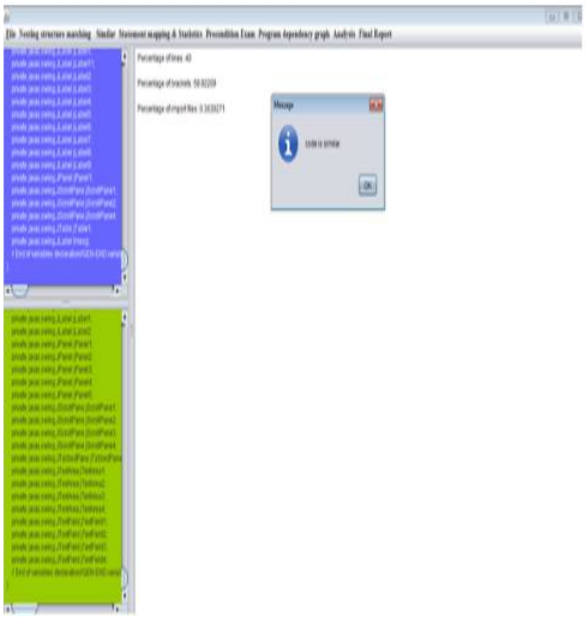
```
***Copy paste***

Different nesting structure found

***Similar code***

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import javax.swing.table.DefaultTableModel;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author [illegible]
 */
public class [illegible] extends javax.swing.JFrame {

  Connection con;
  PreparedStatement pst;
  ResultSet rs;

  int [illegible]
  String s;
public SimpleDateFormat dateFormat=new SimpleDateFormat("yyyy-MM-dd");
  Calendar currentCalendar = Calendar.getInstance();
  Date currentTime = currentCalendar.getTime();
```

Fig.13. Analysis



```
private javax.swing.JButton jButton[illegible]
private javax.swing.JButton jButton[illegible]
private javax.swing.JLabel jLabel1
private javax.swing.JLabel jLabel11
private javax.swing.JLabel jLabel2
private javax.swing.JLabel jLabel3
private javax.swing.JLabel jLabel4
private javax.swing.JLabel jLabel5
private javax.swing.JLabel jLabel6
private javax.swing.JLabel jLabel7
private javax.swing.JLabel jLabel8
private javax.swing.JLabel jLabel9
private javax.swing.JPanel jPanel1
private javax.swing.JScrollPane jScrollPane1
private javax.swing.JScrollPane jScrollPane2
private javax.swing.JScrollPane jScrollPane4
private javax.swing.JTable jTable1
private javax.swing.JLabel lmsg
// End of variables declaration//GEN-END:variables
}
***Line count***
No of lines in Code fragment 1 are : [illegible]
No of lines in Code fragment 2 are : [illegible]

***Brackets count***
No of Brackets in Code fragment 1 are:
No of { are : [illegible]
No of } are : [illegible]
No of ( are : [illegible]
No of ) are : [illegible]
No of [ are : [illegible]
No of ] are : [illegible]

No of Brackets in Code fragment 2 are:
No of { are : [illegible]
No of } are : [illegible]
No of ( are : 11
No of ) are : 11
No of [ are : 1
No of ] are : 1

***Import count***
No of imports in Code fragment 1 are : 12
No of imports in Code fragment 2 are : [illegible]
```

Fig.13.1. Analysis



```
Percentage of lines: 40
Percentage of brackets: [illegible]
Percentage of import files: [illegible]
```

code is similar

Fig.14. Final report

V. CONCLUSION

This novel approach establish a vital quality in clone detection organization i.e. refactorability study that was unconfirmed formerly. To achieve this different objective, a algorithm which initially equivalents to the declarations of the clones in such a manner which reduces the amount of dissimilarity within them. Following this, dissimilarity is studied in opposition to a set of fundamentals to decide or not they could be parameterized with no altering the source code performance. If there is no condition destructions are established, then it offers tool which support for the mechanical refactoring of the clones.

REFERENCES

1. Nikolaos Tsantalis, Member, IEEE, Davood Mazinanian, and Giri Panamoottil Krishnan "Assessing the Refactorability of Software Clones" , IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. XX, NO. XX, MONTH 2015
2. G. P. Krishnan and N. Tsantalis, "Unification and refactoring of clones," in Proceedings of the IEEE Conference on Software Maintenance,Reengineering and Reverse Engineering (CSMR-WCRE),2014 Software Evolution Week, 2014, pp. 104–113.
3. C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitativeapproach," Science of Computer Programming, vol. 74, no. 7, pp. 470 – 495, 2009.
4. C. Roy, M. Zibran, and R. Koschke, "The vision of software clone management: Past, present, and future (keynote paper),"in Proceedings of the IEEE Conference on Software Maintenance,Reengineering and Reverse Engineering, 2014 Software Evolution Week, 2014, pp. 18–33.
5. R. Johnson, D. Pearson, and K. Pingali, "The program structure tree: Computing control regions in linear time," in Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, 1994, pp. 171–185.
6. J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," ACM Transactions on Programming Languages and Systems, vol. 9, no. 3, pp. 319–349, Jul. 1987.