XSCDF: Towards a Framework for Comprehensive Software Clone Detection and Visualization using Ontology

Syed Mohd Fazalul Haque[1], V Srikanth[2], E. Sreenivasa Reddy[3]

[1]*Maulana Azad National Urdu University, Hyderabad, Telangana, India*
[2]*K L University , Guntur, Andhra Pradesh, India*
[3]*Acharya Nagarjuna University, Guntur, Andhra Pradesh, India*

Abstract: **- Software development has become a complex phenomenon as there are increased and ever-changing expectations from clients. In fact the development teams often feel the pressure of releases. They indulge in less than ideal approaches as well to produce code. Sometimes they cut and paste code causing code duplicates or code clones. Clones can lead to propagation of bugs and cause maintenance issues. Detection of code clones has plethora of advantages including copyright protection, elimination of duplicates by refactoring, exploration of design patterns for industry best practices and so on. Analyzing big software projects and finding duplicates is tedious task. Many researchers contributed towards identifying different kinds of clones and detection techniques. However we felt a comprehensive and extendable framework that not only supports clone detection but also visualization techniques for easy comprehension are lacking. In this paper, we propose such framework named eXtensible Software Clone Detection Framework using ontology concept (XSCDF) which is generic and supports clone detection of different languages. It provides placeholders for future techniques. We built a prototype application using Java programming language to demonstrate the proof of concept. Ontology concept is used to visualize clone detection results. The empirical results reveal that the framework has multi-language support for duplicate code detection.**

Index Terms **– Clone, clone detection, SCDF, visualization**

I. INTRODUCTION

Clones are considered to be identical or near identical piece of codes in source code. Usually code clones are created just for avoiding coding. Stated differently code clones results in copy paste operations performed for using the same code in different parts of software. Sometimes code clones occur unintentionally due to similar API usage. In the process of developing huge systems, code cloning became a common phenomenon. Large software systems need continuous maintenance. With code clones there is possibility of bug propagation. It in turn leads to maintenance problems. For instance a JDBC connectivity code is repeated in 100 Java programs in a project. In this case the code is duplicated instead of reusing code. When there is need for switching to different backend or different environment, there are many programs to be modified and recompiled. It causes maintenance problem. It increases the cost of maintenance of software. There is the need for finding duplicates of clones in software and refactor them in order to have a system that can work with reduced maintenance.

Code clones are broadly classified into two types. They are clones with similar source code and clones with similar functionalities. Based on the similarity of source code or functionality four types of clones are identified. They are known as type 1, type 2, type 3 and type 4. Type 1 clones are similar except differences in comments and whitespaces. Type 2 clones are syntactically and structurally identical but differ in identifiers, comments, layout, types and literals. Type 3 clones are identical code fragments with further modifications in addition to having differences in comments, layout, types, literals, and identifiers. Type 4 clones perform identical computations but implemented with different syntactical variants. The type 4 clones are example for functional similarities while the first 3 types exhibit similarity of source code. Therefore it is very important to have code detection techniques for leveraging software industry to have best practices.

Our Contributions

Keeping the importance and impact of finding clones in software we proposed a framework known as eXtensible Software Clone Detection Framework using ontology concept (XSCDF) which provides generic architecture which can help to detect clones in multiple languages. Moreover it provides placeholders to accommodate future detection techniques. In addition to this we proposed a methodology for clone detection besides visualizing clones. We built a prototype application to demonstrate the proof of concept. The results are presented using visualization of text based GUI and ontology based knowledge representation as well.

The remainder of the paper is structured as follows. Section II talks about review of literature pertaining to software cloning and clone detection. Section III proposes a comprehensive framework that can cater to the needs of clone detection. Section IV provides implementation details. Section V presents results of experiments while section VI provides conclusions and directions for future work.

II. RELATED WORKS

This section review literature on code clone detection. Gybels and Kellens (2005) [8] explored clone detection concepts in Aspect Oriented Software Development (AOSD) or Aspect Oriented Programming (AOP). AOP is the paradigm shift in programming of object oriented (OO) languages. When code is transformed to AOP approach, there might be some duplicates that form clones. Young et al. (2005) [11] studied the concept of cloning from biological perspective and provided analogy with software cloning. Salvi and Tuberosa (2005) [17] used positional cloning concept in case of biological experiments. Their research was pertaining to DNA sequences that can be understood in terms of code cloning as well. Cline et al. (2005) [20] explored clone detection approaches using gene expression data. They proposed a framework to serve this purpose. Kuhn et al. (2005) [24] explored the concept of semantic clustering. They also used the process of high-level clone detection in order to improve quality of clustering process.

Nikolsky et al. (2005) [30] focused on drug discovery in biochemical experiments. They used similar expressions in order to find out duplicates. Vollenveider et al. (2006) [26] used clone detection concepts in biological experiments. They used concept known as clone tolerance in the environmental and experimental botany. Laufs et al. (2006) [10] explored biological concepts with respect to cloning. Ratiu et al. (2006) [9] opined that code redundancies is one of the reasons of software clones. They also said that synonymy and polysemy concepts can also be found as clones in some cases. Groups of elements or concepts which can have duplicates can be located using clone detection methods. Czarnecki et al. (2006) [5] explored feature models, clone detection in feature models and formal representation of features using ontology. The feature models are presented in the form of views on ontology. A good review of clone detection techniques is found in [32].

In [5] the authors also explored software cloning when the software involves DNA sequences with duplicates. Similar kind of work was done by Darias et al. (2007) [12]. Meditskos and Bassiliades (2007) [15] explored object oriented similarity measures in order to find out effective service discovery with respect to web services. In the process they focused on cone cloning in order to find similar services. Poshyvanyk et al. (2009) [22] explored object oriented software systems for finding coupling measures. They used the measures for finding concept clones and impact analysis. By finding coupling they focused on the quality of software. Pariset et al. (2009) [16] used clone detection methods for gene expressions. Roy et al. (2009) [1] compared and reviewed many clone detection techniques. They explored textual approaches, lexical approaches, tree-matching approaches; metrics based approaches, semantic approaches, and hybrids. The tools compared by them include usage facets, interaction facets, language facets, technical facets, adjustment facets, processing facets, and evaluation facets.

Martin and Cordy (2011) [7] explored the concept of contextual clones to find out web service similarities. Web Service Description Language (WSDL) is used in order to search for duplicate services or clones. Contextual clones make use of contextual codes which are duplicates in WSDL files. Keivanloo et al. (2011) [6] proposed an approach for searching real-time clones in the Internet. Their approach is hybrid in nature as it uses many techniques such as semantic web reasoning, information retrieval clustering, and code patter indexing. It is able to detect clones with less response time. Jia et al. (2011) [23] focused on mutation testing. In the process various clones are injected into source code. Such code is detected using clone detection techniques and tools. Takuya and Masuhara (2011) [25] explored associative search of source code in order to find duplicates while developer is typing source code.

It does mean that they proposed a method to identify duplicates as you type source code. Malhotra et al. (2012) [27] used similarity measures and lexical concepts in order to find similar users in social networks using footprint of users. Mishne et al. (2012) [29] explored type state-based semantic code search in programs in order to identify duplicates. Thus they found code duplicates in source code. Wursch et al. (2012) [19] focused on explored ontologies to present pyramid of software evolution. In the process they also explored code clones. They opined that code clones are part of software evolution. Shamshirband et al. (2013) [18] explored multi-agent based approach for clone detection and clone selection with respect to gene expressions. Stephan and Cordy (2013) [4] presented model comparison approaches that help in detecting duplicates in software models. Model clone detection or code clone detection is needed when software contains duplicates due to similar requirements are repeated in a project.

Keivanloo et al. (2011) [13] explored ontology models to represent source code. Complex code search was performed for various purposes including clone detections. They proposed a linked data framework for providing sharable services. Towards software mining and analysis they provided some baseline implementation. Kaur et al. (2012) [31] compared clone detection tools such as SolidSDD and CONQAT by using different clone metrics. Stephan and Cordy (2012) [3] focused software models in model-driven development. Especially they explored on clone detection with respect to high-level software models. Leopold et al. (2014) [21] studied process model clones with respect to the abstractions of process models. They used process model repositories and found clones in the models for making strategic decisions.

Kelkar and Deobagkar (2014) [28] explored DNA sequences for finding clones. As part of drug effectiveness testing, clones are used to find effective results. Tonella et al. (2007) [14] reviewed various reverse engineering studies for modernizing legacy systems. In the process they also explored to detect clones in the software in order to refactor it for better maintenance. They identified unique components and then transformed into different representation in order to upgrade legacy code to modern code. Krishnan and Ananthapadmanaban (2014) [2] focused on the clone detection process in sensor networks. Clone detection techniques used in such networks can help secure communications in the network.

In this paper we proposed a methodology for discovering code clones and visualizing them software. It makes use of source code analysis in order to find code duplicates. Our work in this paper is comparable with tools such as SolidSDD and CONQAT explored by Kaur et al.

III. PROPOSED FRAMEWORK

This section describes our generic framework which provides flexible and comprehensive means of achieving clone detection. It is extensible and provides placeholders for pluggable clone detection methods and visualization techniques for future enhancements. An important feature of the proposed framework is that it supports personalized user preferences. The user preferences can help in choosing target language for clone detection, selection of visualization technique preferred and even the selection of clone detection technique. Thus the personalized preferences are associated with the user session. These preferences can be changed when required.

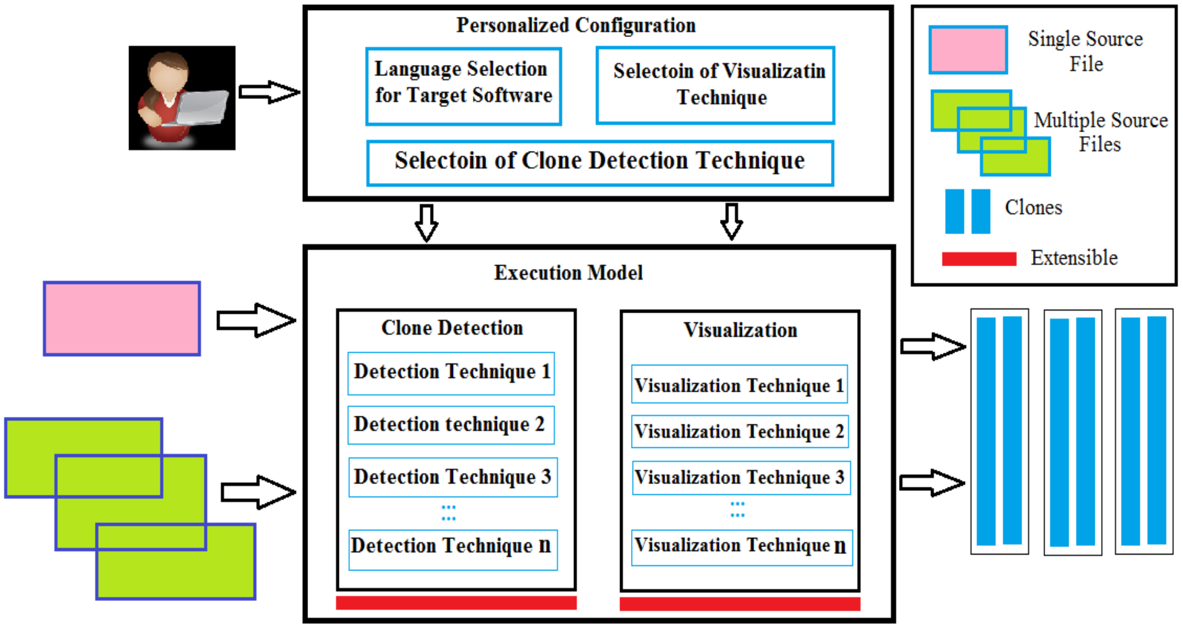

Personalized Configuration
Language Selection for Target Software
Selectoin of Visualizatin Technique
Selectoin of Clone Detection Technique
Single Source File
Multiple Source Files
Clones
Extensible
Execution Model
Clone Detection
Detection Technique 1
Detection technique 2
Detection Technique 3
Detection Technique n
Visualization
Visualization Technique 1
Visualization Technique 2
Visualization Technique 3
Visualization Technique n

Figure 1 – Overview of XSCDF

As shown in Figure 1, the framework has execution model apart from the personalized configuration. The execution model is runtime functionality of the proposed framework. The runtime functionality includes both clone detection and visualization. The clone detection procedure depends on the preference in terms of clone detection technique chosen by end user. The visualization technique is also same. Based on the user preferences, the chosen visualization technique is used. The framework supports single source file as input of a set of source files as folder.

One Size Does Not Fit All

It is true that one size does not fit all. In the process of proposing and implementing a generic framework named XSCDF we intend to support multiple clone detection techniques and visualization methods using ontology concepts in order to cater to the needs of different users. This makes the proposed framework extensible and flexible besides helping users to have intuitive interface based on the personalized preferences.

Pseudo Code for the Flow of Execution Model

```
Initialize target language vector TL
Initialize visualization vector V
Initialize clone detection vector CD
Obtain target language tl from TL
Obtain visualization technique vt from V
Obtain clone detection technique dt from CD
IF dt=A THEN
     Set A as default dt FOR user u
ELSE IF dt=B THEN
    Set B as default dt for user u
ELE IF dt=C then
    Set C as default dt for user u
END IF
IF vt=V1 THEN
     Set V1 as default vt for user u
ELSE IF vt=V2 THEN
    Set V2 as default dt for user u
ELE IF vt=V3 then
    Set V3 as default dt for user u
END IF
Input user file f
IF f is source file THEN
    Detect clones
    Visualize clones
ELSE
     Instruct user to choose source file
END IF
```

Listing 1 – Pseudo code for execution model

The pseudo code provides the details of the proposed execution model which takes care of runtime user preferences before applying clone detection and visualization techniques. It makes use of chosen preferences for performing clone detection and visualization of clones.

Zero Maintenance Approach

Industry best practices are required when new techniques are to be adapted without much maintenance. In the implementation of clone detection, the XSCDF supports user preferences. The preference range may increase in future. To avoid reinventing the wheel again, design patterns are introduced. The interfaces are kept same while the framework supports future implementations also without any maintenance cost.

DetectionTechnique.java

It is an interface which contains common interface required by clone detection. The methods are abstract in nature and the interface can have many implementations.

VisualizationTechnique.java

It is an interface which contains common interface required by visualization. The methods are abstract in nature and the interface can have many implementations.

Class Hierarchy for Clone Detection Factory

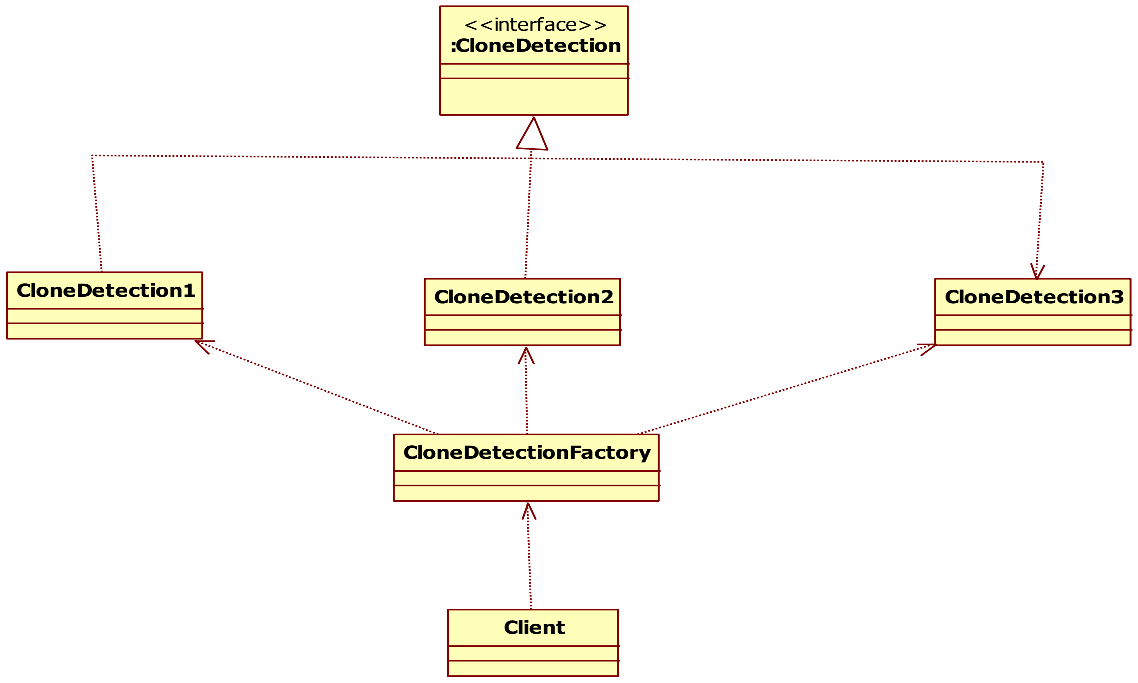


Figure 2 - Class Hierarchy for Clone Detection Factory

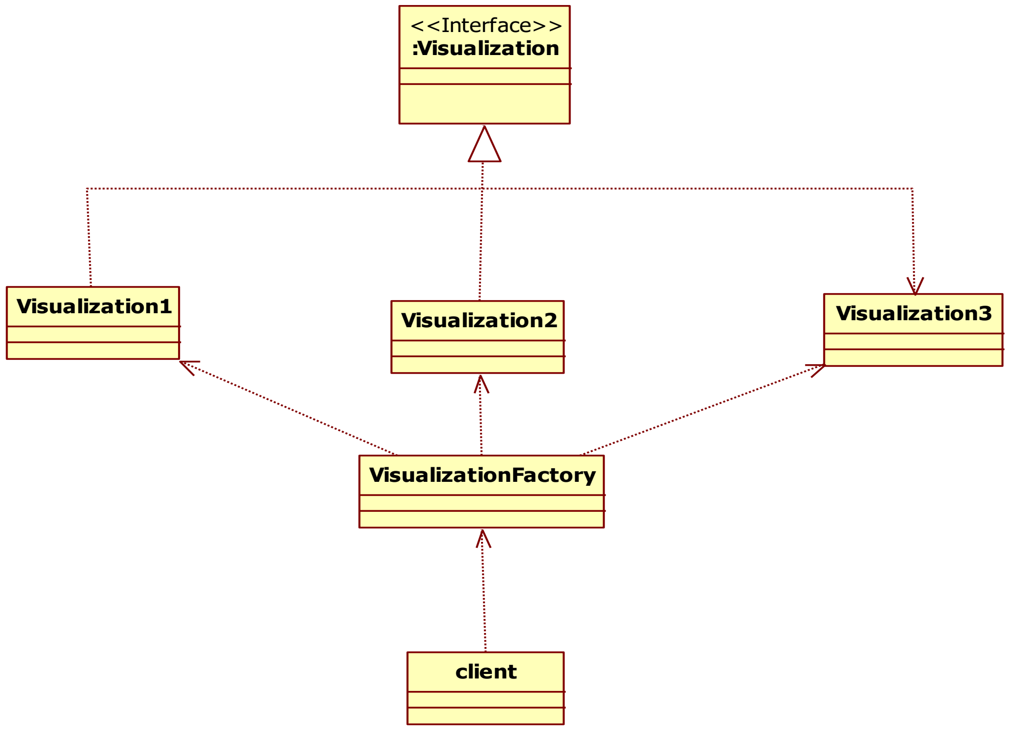


Figure 3: Class hierarchy for visualization of clone detection results

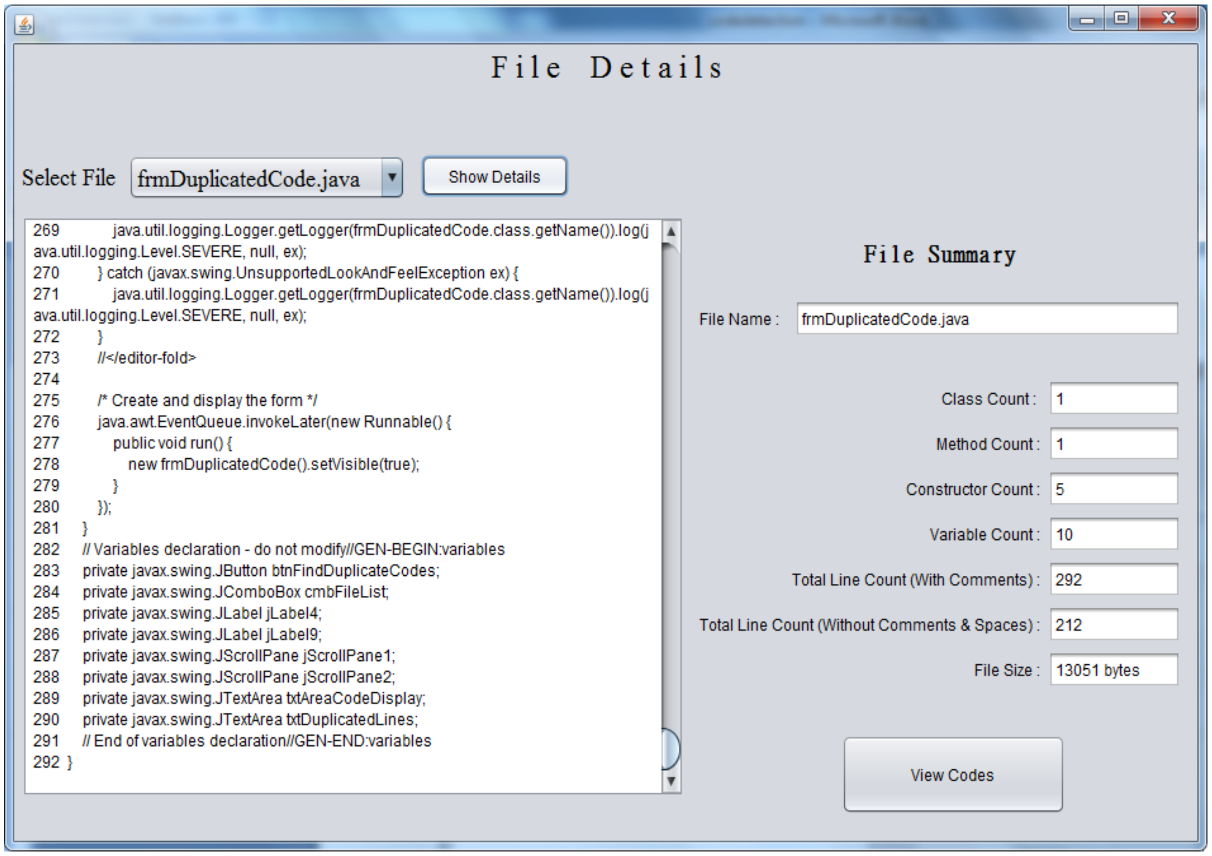
```
public class CloneDetectionFactory {
    public static CloneDetection getCloneDetectionTechnique(String technique) {
        if(technique.equals("A"))
            return new CloneDetection1();
        else if(technique.equals("B"))
            return new CloneDetection2();
        else
            return new CloneDetection3();
    }
}
```

Listing 2 – Factory pattern

As found in listing three the factory pattern is able to take client need and return an instance of chosen technique. Thus it helps in accommodating future techniques as well. In the same fashion, we can assume VisualizationFactory class too.

IV. IMPLEMENTATION

The proposed methodology explained earlier in this paper is used to implement the clone detection method. We built a prototype application to demonstrate the proof of concept. The application makes use of Java's Swing API for intuitive user interface. The functionality is implemented using collection API present in java.util package, regular expression API of the java.util.regex package, java.swing.text package for visualization of clones, java.io package to deal with IO operations on files and java.util.logging API for recording events into a log file. JFileChooser class of swing API is used for having interactive selection of a file.



```
            java.util.logging.Logger.getLogger(frmDuplicatedCode.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(frmDuplicatedCode.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new frmDuplicatedCode().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton btnFindDuplicateCodes;
    private javax.swing.JComboBox cmbFileList;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTextArea txtAreaCodeDisplay;
    private javax.swing.JTextArea txtDuplicatedLines;
    // End of variables declaration//GEN-END:variables
}
```

File Name :	frmDuplicatedCode.java
Class Count :	1
Method Count :	1
Constructor Count :	5
Variable Count :	10
Total Line Count (With Comments) :	292
Total Line Count (Without Comments & Spaces) :	212
File Size :	13051 bytes

Figure 4 – UI showing details of selected file.

Figure 4 shows the selected file and its metrics like class count, method count, constructor count, variable count, total tile count with and without comments and spaces, and total file size in bytes. Figure 1 appears after choosing a file directory. The prototype application supports choosing either a file for duplicate detection or a folder with multiple source files. A folder in turn may have sub folders. The sub folders in turn may have sub directories. To hand the complexity two recursive functions by name getTotalFileFount() and getFileList() are implemented.

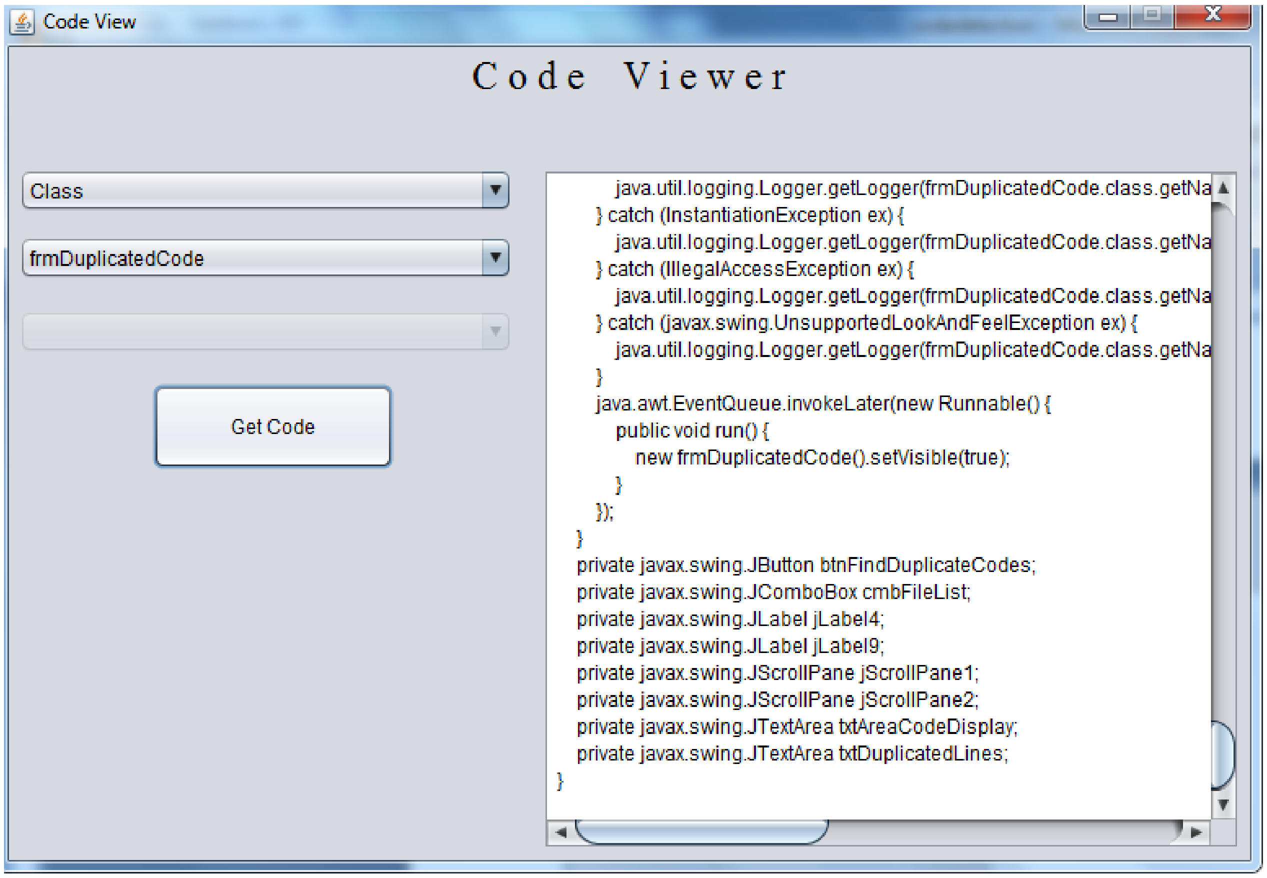
```
public void getTotalFileCount(File dir) {
                try {
                        File[] files = dir.listFiles();
                        for (File file : files) {
                                if (file.isDirectory()) {
                                        System.out.println("directory:"+ file.getCanonicalPath());

                                        getTotalFileCount(file); //recursive call
                                } else {
                      if (file.getName().endsWith(".java"))
                        TotFileCount++;
                                        System.out.println("    file:" + file.getCanonicalPath());
                                }
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
```

Listing 3 – Shows recursive calls to track the count of files in directories and sub directories

```
public void getFileList(File dir) {
                try {
                        File[] files = dir.listFiles();
                        for (File file : files) {
                                if (file.isDirectory()) {
                                        System.out.println("directory:" + file.getCanonicalPath());
                                        getFileList(file);
                                } else {
                      if (file.getName().endsWith(".java"))
                      {
                        path[TotFileCountTemp] = file.getAbsolutePath();
                        TotFileCountTemp++;
                      }
                                        System.out.println("    file:" + file.getCanonicalPath());
                                }
                        }
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
```

Listing 4 – Shows recursive calls to obtain the list of files of different directories

The purpose of this code is to obtain all source files from the given project folder. Thus it facilitates duplicate code detection of various files. These code listings show the part of finding all source files only. The rest of the functionality is in the form of code analysis which finds duplicate code and detects it. Collection API is used to maximum extent in order to deal with huge amount of code and comparisons and intermediate results. Figure 2 shows the source code based on the selection of a file.

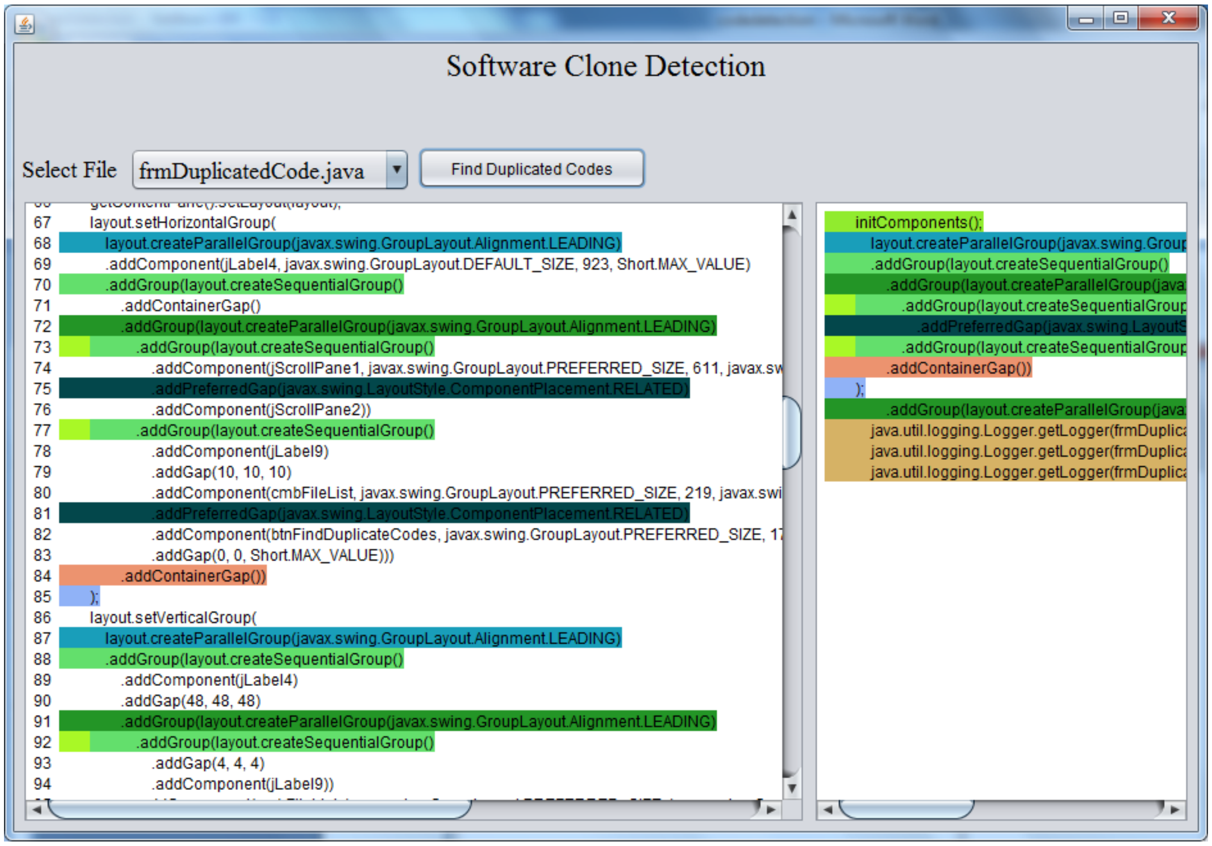


```
        java.util.logging.Logger.getLogger(frmDuplicatedCode.class.getNa
    } catch (InstantiationException ex) {
        java.util.logging.Logger.getLogger(frmDuplicatedCode.class.getNa
    } catch (IllegalAccessException ex) {
        java.util.logging.Logger.getLogger(frmDuplicatedCode.class.getNa
    } catch (javax.swing.UnsupportedLookAndFeelException ex) {
        java.util.logging.Logger.getLogger(frmDuplicatedCode.class.getNa
    }
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new frmDuplicatedCode().setVisible(true);
        }
    });
  }
  private javax.swing.JButton btnFindDuplicateCodes;
  private javax.swing.JComboBox cmbFileList;
  private javax.swing.JLabel jLabel4;
  private javax.swing.JLabel jLabel9;
  private javax.swing.JScrollPane jScrollPane1;
  private javax.swing.JScrollPane jScrollPane2;
  private javax.swing.JTextArea txtAreaCodeDisplay;
  private javax.swing.JTextArea txtDuplicatedLines;
}
```

Figure 5 – Shows UI viewing source code of selected file

As can be seen in Figure 5, it is evident that the source code of selected file is presented in a text area control. However, it is not easy to find duplicates manually provided the size of files in the source code. We automated the clone detection process. The implementation of duplicate code detection mechanism is described here. Vector class from java.util package is used to have multiple instances in order to store all classes, all methods of a class, and all variables of a class. A Plain Old Java Object (POJO) class or Java Bean class named VariableDefinition is used to hold details of one variable. A set of such bean instances can hold the state and meta data of all variables. The data about variable include variable name, data type, position, access type, start line number, whether variable has been initialized.

In the same fashion MethodDefinition class is another Java Bean class which can hold the state of Method instance. This class can hold method name, return type, parameter definitions, local variables, content, method line number, variable count, maximum depth of method nesting, method type, access type, and start line number. In the same fashion a ClassDefinition class instance can capture details of a class in terms of interface names, method names, class variable list, parent class name, class name, content of class, class line number, and maximum class nesting depth, access type, variable count, and start line number. There is another class for having content of a file in terms of different vectors such as file content line by line with and without comments, class code blocks, method code blocks, variables, and so on. This class is defined with methods to obtain class details, method details, literals, code block types, removal of comments, obtaining entire code block, obtaining primitive data types, counting variables, and obtaining details of variables.



```
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addComponent(jLabel4, javax.swing.GroupLayout.DEFAULT_SIZE, 923, Short.MAX_VALUE)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 611, javax.sw
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(jScrollPane2))
                .addGroup(layout.createSequentialGroup()
                    .addComponent(jLabel9)
                    .addGap(10, 10, 10)
                    .addComponent(cmbFileList, javax.swing.GroupLayout.PREFERRED_SIZE, 219, javax.swi
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(btnFindDuplicateCodes, javax.swing.GroupLayout.PREFERRED_SIZE, 17
                    .addGap(0, 0, Short.MAX_VALUE)))
            .addContainerGap())
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addComponent(jLabel4)
            .addGap(48, 48, 48)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(layout.createSequentialGroup()
                    .addGap(4, 4, 4)
                    .addComponent(jLabel9))
```

```
    initComponents();
        layout.createParallelGroup(javax.swing.Group
        .addGroup(layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(java
                .addGroup(layout.createSequentialGroup
                    .addPreferredGap(javax.swing.LayoutS
                .addGroup(layout.createSequentialGroup
            .addContainerGap())
    );
            .addGroup(layout.createParallelGroup(java
        java.util.logging.Logger.getLogger(frmDuplica
        java.util.logging.Logger.getLogger(frmDuplica
        java.util.logging.Logger.getLogger(frmDuplica
```

Figure 6 – UI visualizing code clones

The duplicate code detection is actually done by an iterative process after pre-processing which eliminates unnecessary processing of certain things such as white spaces and comments. The content of file line by line is processed in order to add duplicates to different vectors. DefaultHighlighter class is used to apply a colour to one piece of code duplicate stored in a vector. The Random class of java.util package is used to have a colour combination with Red, Green and Blue (RGB) with arbitrary values between 0 and 255. Each code clone is painted using separate randomly picked colour. The application also supports pluggable look and feel for presenting results with native look and feel of OS. Two vectors are used to add duplicate lines and corresponding colour codes respectively. These two vectors are used later for visualization of results.

V. RESULTS AND EVALUATION

The proposed methodology and the framework named XSCDF is evaluated using different metrics. The results are compared with other tools namely SolidSDD and CONQAT. The metrics used for evaluation are described here.

Population of Clone Class (POP)

It refers to the number of elements in a clone class. A clone class is the class which contains at least on clone pair. Clone pair is a pair of code segments that happen to be identical. If the POP is more it does mean that code clones are more frequent in the in the system.

Ratio of Non-Repeated Token Sequences (RNRS)

In a given clone set RNRS is the ratio of non-repeated token sequences of code clones. If the RNRS is higher it indicates that each code clone contains more non-repeated token sequences.

$$RNR(S) = \frac{\sum_{i=1}^{n} LOS_{non-repeated}(C_i)}{\sum_{i=1}^{n} LOS_{whole}(C_i)}$$

LEN

In a given clone set, it refers to the average length of token sequences of code clones. Higher in length indicates that more token sequences exist in the code clones.

Execution Time

It is the amount of time taken by the tool in question to identify clones in a given source file.

Clones

It is the number of clones identified by the tool in question in a given source file.

Gaps

It is the statistical measure to know how many insertions or deletions are in a source file.

Metrics → Tools	**Clones**	**Gaps**	**RNR**	**POP**	**LEN**	**Execution Time**
Solid SDD	9	2	5.2	386	80	1s
CONQAT	2	0	7	398	82	20s
XSCDF	9	2	8	398	82	o.3s

Table 1 - Tool comparison based on metrics

Tools or Features	**Solid SDD**	**CONQAT**	**XSCDF**
Languages Supported	C,C++,JAVA,C#	ABAP,ADA,C++,C,JAVA,COBOL	JAVA,C++,C#
Domain	Clone detection	Clone detection	Clone detection
Requirements	No requirements	Java1.6,graphviz2,Microsoft.net.2.0	No requirements
Source data	Programming language, files	files	Source file or folder
Result output	Source code, graphical view	Reports, clone, compare view	Clones graphical view
Metrics produced	Clone metrics	Clone metrics, line metrics	Clone metrics

Table 2 – Tool comparison based on features

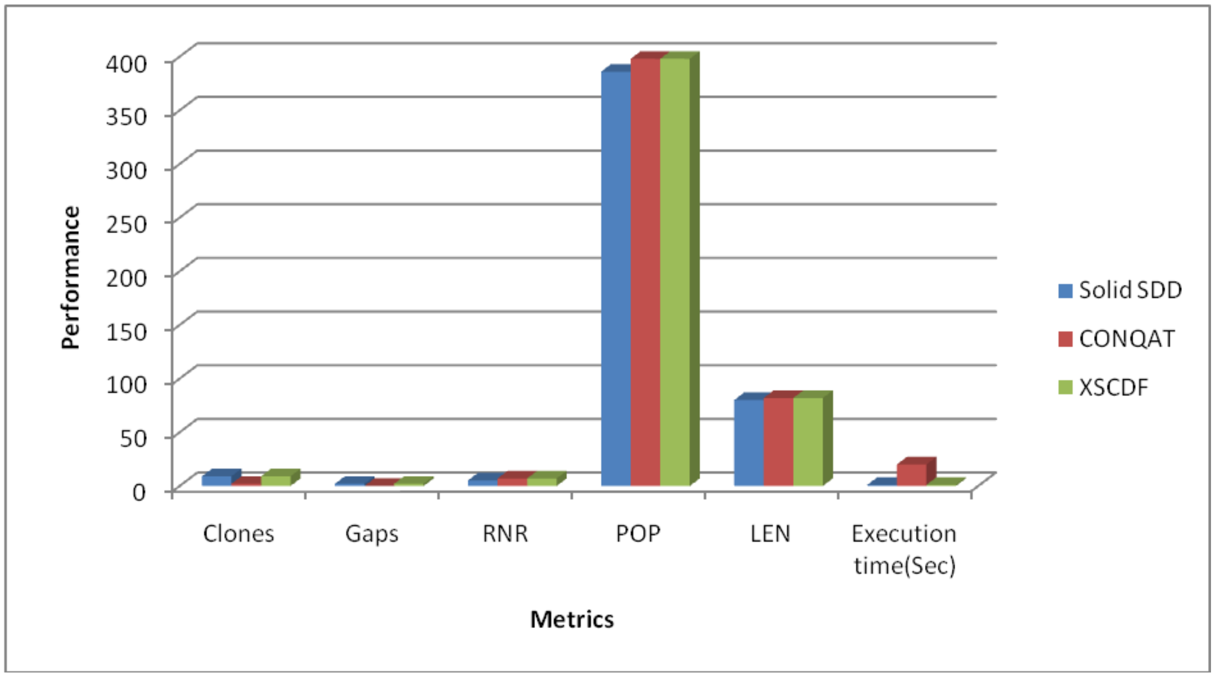


Figure 7 – Performance comparison of tools based on metrics

As shown in Figure 7, the performance comparison of tools is made in terms of metrics such as clones, gaps, RNR, POP, LEN and execution time. The proposed system XSCDF has comparable performance with other tools such as SolidSDD and CONQAT. The ontology concept is used to visualize the duplicates found in the empirical study.

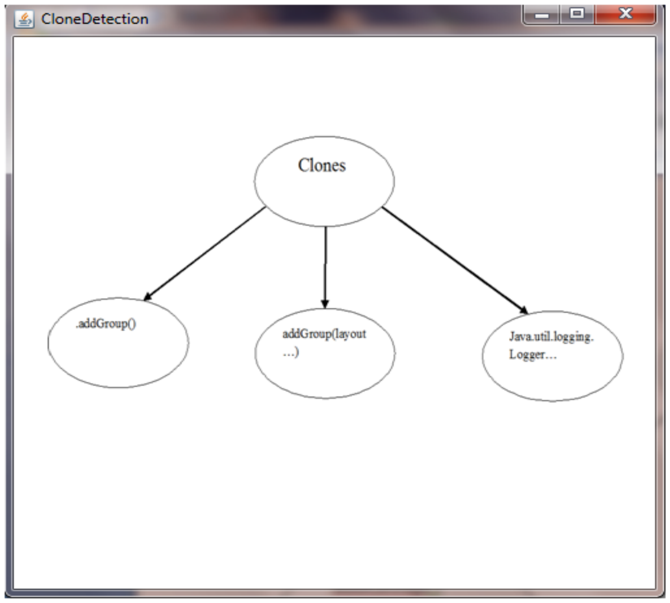


Figure 8 - Shows clones as concepts in ontology knowledge representation

As shown in Figure 8, it is evident that the ontology concept is used to visualize the duplicates found in the given source code. The concepts are used to have knowledge representation.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we studied software clones and clone detection mechanisms. Software clones cause potential risk to software industry as they can propagate bugs and cause maintenance problems. The existing detection techniques are based on different means such as static code analysis, reflection and other techniques. There is need for a comprehensive and extendable framework that can support future detection methods and visualization techniques. In this paper we proposed a generic framework for clone detection and visualization. It is named as eXtensible Software Clone Detection Framework using ontology concept (XSCDF). This framework provides placeholders for different cloning techniques that can be plugged in future. We proposed methodology for clone detection. We built a prototype application to realize the framework and he methodology for proof of concept. Moreover it supports clone detection in source codes built in multiple languages such as C, C++, and Java and C #. Our empirical results revealed that the proposed framework is effective in clone detection. Two kinds of clone visualization are made in this paper. They are textual visualization with graphical view and ontology visualization. This research can be extended further by exploring different clone detection techniques and their applications besides visualization techniques to have intuitive comprehension of clones for making well informed decisions. Another research direction is to leverage clone detection using ontology.

REFERENCES

[1]. Chanchal K. Roy, James R. Cordy and Rainer Koschke. (2009). Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. *Science of Computer Programming* , p1-43.
[2]. K.B.Gopi Krishnan and K.R.Ananthapadmanaban. (2014). On the Node Clone Detection in Sensor Networks for Electronic Copy Right Management System. *International Journal of Computer Applications*. 2 (8), p345-355.
[3]. Matthew Stephan and James R. Cordy. (2011). A Survey of Methods and Applications of Model Comparison. *ACM* , p1-42.
[4]. Matthew Stephan and James R. Cordy. (2013). A Survey of Model Comparison Approaches and Applications. *IEEE*, p1-12.
[5]. Krzysztof Czarnecki, Chang Hwan Peter Kim and Karl Trygve Kalleberg. (2006). Feature Models are Views on Ontologies. *IEEE*, p1-10.
[6]. Iman Keivanloo, Juergen Rilling and Philippe Charland. (2011). SeClone - A Hybrid Approach to Internet-scale Real-time Code Clone Search.*IEEE* , p1-12.
[7]. Douglas Martin and James R. Cordy. (2011). Analyzing Web Service Similarity Using Contextual Clones. *ACM*, p1-6.
[8]. Kris Gybels and Andy Kellens. (2005). Experiences with Identifying Aspects in Smalltalk Using 'Unique Methods. *ISSN* , p1-6.
[9]. Daniel Ratiu and Florian Deissenboeck. (2006). How Programs Represent Reality (and how they don't). *IEEE*, p1-10.
[10]. Stephanie Laufs, Guillermo Guenechea, Africa Gonzalez-Murillo, K. Zsuzsanna Nagy, M. Luz Lozano, Coral del Val, Sunitha Jonnakuty, Agnes Hotz-Wagenblatt, W. Jens Zeller, Juan A. Bueren and Stefan Fru. (2006). Lentiviral vector integration sites in human NOD/SCID repopulating cells. *Inter Sciences*. . (.), p1-11.
[11]. Jason A. Young , Quinton L. Fivelman , Peter L. Blair , Patricia de la Vega , Karine G. Le Rochd, Yingyao Zhoud, Daniel J. Carucci , David A. Baker and Elizabeth A. Winzeler. (2005). The Plasmodium falciparum sexual development transcriptome: A microarray analysis using ontology-based pattern identification. *ISSN*. 143 , p67–79.
[12]. M. J. Darias , J. L. Zambonino-Infante , K. Hugot & C. L. Cahu and D. Mazurais. (2008). Gene Expression Patterns During the Larval Development of European Sea Bass (Dicentrarchus Labrax) by Microarray Analysis. *International Journal of Computer Applications*. 10 , p416–428.
[13]. Iman Keivanloo, Christopher Forbes, Juergen Rilling and Philippe Charland. (2011). Towards Sharing Source Code Facts Using Linked Data. *ACM*, p1-4.
[14]. Paolo Tonella · Marco Torchiano · Bart Du Bois · Tarja Systä. (2007). Empirical studies in reverse engineering: state of the art and future trends. *Springer Science*. 12 , p551–571.
[15]. Georgios Meditskos and Nick Bassiliades. (2007). Object-Oriented Similarity Measures for Semantic Web Service Matchmaking. *IEEE* , p1-10.
[16]. Lorraine Pariset, Giovanni Chillemi, Silvia Bongiorni, Vincenzo Romano Spica and Alessio Valentini. (2009). Microarrays and high-throughput transcriptomic analysis in species with incomplete availability of genomic sequences. *Ind. Eng. Chem. Res*. 25, p1-8.
[17]. Silvio Salvi and Roberto Tuberosa. (2005). To clone or not to clone plant QTLs: present and future challenges. *ISSN*. 10 (6), p1-10.
[18]. Shahaboddin Shamshirband , NorBadrulAnuar, MissLaihaMatKiah and AhmedPatel. (2013).An appraisal and design of amulti-agent system based cooperative wireless intrusion detection computational intelligence technique. *IEEE*, p1-23.
[19]. Michael Würsch , Giacomo Ghezzi , Matthias Hert , Gerald Reif and Harald C. Gall. (2012). SEON: a pyramid of ontologies for software evolution and its applications. *Springer Science*. 94 , p857–885.

[20]. Melissa S. Cline, John Blume, Simon Cawley, Tyson A. Clark, Jing-Shan Hu, Gang Lu, Nathan Salomonis, Hui Wang and Alan Williams. (2005). ANOSVA: a statistical method for detecting splice variation from expression data. *ISSN* , p1-10.

[21]. Henrik Leopold, JanMendling , HajoA.Reijers and MarcelloLaRosa. (2014). Simplifyingprocessmodelabstraction: Techniquesforgeneratingmodelnames. *ISSN*. 39, p134–151.

[22]. Denys Poshyvanyk , Andrian Marcus , Rudolf Ferenc & Tibor Gyimóthy. (2009). Using information retrieval based coupling measures for impact analysis. *Springer* . 14 , p5-32.

[23]. Yue Jia and Mark Harman. (2011). An Analysis and Survey of the Development of Mutation Testing. *IEEE*. 37 (5), p1-30.

[24]. Adrian Kuhn, Stephane Ducasse and Tudor Gırba. (2005). Enriching Reverse Engineering with Semantic Clustering. *IEEE*. . (.), p1-10.

[25]. Watanabe Takuya and Hidehiko Masuhara. (2011). A Spontaneous Code Recommendation Tool Based on Associative Search. *ACM* , p1-4.

[26]. Pierre Vollenweider, Claudia Cosio, Madeleine S. Gunthardt-Goerg and Catherine Keller. (2006). Localization and effects of cadmium in leaves of a cadmium-tolerant willow (Salix viminalis L.).*ACM*. 58 , p25–40.

[27]. Anshu Malhotra, Luam Totti, Wagner Meira Jr,Ponnurangam Kumaraguru and Virgılio Almeida. (2012). Studying User Footprints in Different Online Social Networks. *IEEE* , p1-6.

[28]. Ashwin Kelkar and Deepti Deobagkar. (2016). A novel method to assess the full genome methylation profile using monoclonal antibody combined with the high throughput based microarray approach. *ISSN* , p1-6.

[29]. Alon Mishne,Sharon Shoham and Eran Yahav. (2012). Typestate-Based Semantic Code Search over Partial Programs. *ACM*, p1-20.

[30]. Yuri Nikolsky,Tatiana Nikolskaya and Andrej Bugrim. (2005). Biological networks and analysis of experimental data in drug discovery. *ACM*. 10 (9), p1-10.

[31]. Kaur, P., Kaur, H. and Kaur, R. (2012). Comparison of Clone Detection Tools: CONQAT and SolidSDD. International Journal of Advanced Research in Computer Science and Software Engineering, Volume 2, Issue 5, p1-5.

[32]. Syed Mohd Fazalul Haque, V Srikanth and E. Sreenivasa Reddy (2016). Present State-of-the-Art of Software Cloning and Detection Methods. Communicated to International Association for the Engineers.