Advances in Computational Sciences and Technology
ISSN 0973-6107 Volume 10, Number 6 (2017) pp. 1635-1649

http://www.ripublication.com

Study and Analysis of Object-Oriented Languages using Hybrid Clone Detection Technique

Gurvinder Singh[#1] and Jahid Ali[*2]

[#]Research scholar, I.K.Gujral Punjab Technical University, India.

** Director SSICMIT, Badhani, Punjab, India.*

Abstract

Code duplication is prevalent software development traditions that announce various complementary or identical segments of code. Clone detection is a ground of dynamic research in which great equipments previously exists to inspire code clone detection. Most of the studies have explicitly or implicitly expected that code cloning is negative. Clones are harmful for software maintenance because it increases the complexity of system and maintenance cost. If we detect software clones it can decrease software maintenance cost. Numerous code clone detection techniques were exiting. The goal of those researches is the exploration of various clone detection techniques and tools. In this study, we propose an efficient clone detection tool which is used to detect clones in various programming languages. This tool enhanced the performance metrics such as recall and precision. The result is demonstrate that the proposed tool outperform as compare to traditional tools, which are shown by simulations using Net Beans.

Keywords: Clone Detection, Hybrid Approach, Metrics Computation.

1. INTRODUCTION

Duplication is exposed by measuring features of source fragments. The challenging problem for the detection is that source code is rarely copied exactly [1]. The detection system must be able to ignore the superficial differences and to concentrate on essential similarities in order to find relevant duplication. While the high level records yielded by syntactic and semantic code analysis can be put to powerful use, the drawbacks of these deep evaluation techniques is that they reduced adaptability to different programming languages. Because duplication is a ubiquitous trouble, however, support for duplication detection and management is needed for every

programming language in use. Software maintenance is the principal driver of aggregate expenses in the lifecycle of long-living software systems. The maintenance phase consists of those changes that are made to a software system after it has been deployed to the client upon client acceptance. The studies show that the major fraction of the annual software expenditure is being spent for maintaining existing software systems. The replication of code fragments across the system often diminishes maintainability as it expands the code size and hinders manual code change, review and investigation. With the increasing levels of sophistication and complexity of software systems, the standards for software quality and productivity are also getting upward. Developers continually look for various procedures, tools and practices to accelerate software development without falling into additional software defects.

A code clone is a code block in source files which is identical or similar to another code block. It is a very common practice by developers to copy existing code and pasting it somewhere else with major or minor edits to increase productivity. This reuse mechanism results in duplicate or very similar code fragments in the code base which are commonly known as code clones. If the presence of clones in program artifacts causes the artifacts to be more frequently changed and the cloned code shows unstable behavior, then clones are considered harmful. Due to huge measure of data involved, it becomes highly difficult to detect code duplication in large code bases or across project boundaries. Clones are connections between different projects. Duplicated fragments often result in significantly increase in the work to be done during code optimization. Recent related studies also show that inconsistent changes to cloned code are frequent and lead to severe unexpected behavior. Subsequently, it gets more difficult to maintain the software systems with code clones and can lead to include subtle errors. Hence, while cloning is often intentional and can be useful in many ways still it can also be harmful in software maintenance and evolution.

The rest of the paper is organized as follows. Section 1 includes the introduction about clone detection area. Section 2 demonstrates background information about the clone detection system. Section 3 presents the problem formulation of the background study. Section 4 exhibited the materials and methods which includes the information about dataset, objectives and proposed work. In Section 5 we describe results. Finally, we discuss conclusion in section 6.

2. RELATED WORKS

Chanchal K. Roy et al. [1] presented first empirical study of function clones in open source software using a new hybrid clone detection tool, known as NICAD, combining the strengths of both text based and AST-based clone detection techniques and hence yielding highly accurate identification of cloned code in software systems. They have provided an in-depth empirical study of function clones in more than 15

open source C and Java systems including Apache httpd and the entire Linux Kernel. Mark Gabel [2] presents the first scalable algorithm for semantic clone detection based on dependence graphs. They have provided an extension to the definition of a code clone to include semantically related code and provided an approximate algorithm for locating these clone pairs. The difficult graph similarity problem has been reduced to a tree similarity problem by mapping interesting semantic fragments to their related syntax. Chanchal K. Roy [3] built a hybrid clone detection method followed by proposing a meta-model of clone types. They provided a scenario-based comparison and evaluation of all of the currently available clone detection techniques/tools. Yoshiki Higo et al. [4] proposed two techniques for enhancing the quality of code clone detection using program dependency graph. Undoubtedly, inter procedural PDGs is more time-consuming, it detects more interesting and more beneficial code clones because functionalities straddling two or more methods are identical, detected as a single clone set. Iman Keivanloo et al. [5] introduced a novel hybrid clone detection approach named SeClone clone search tool, which is based on multi-layer indexing. A clone ontology (CLON) is introduced to model the code clone detection vocabulary to support the use of reasoning services and to provide a formal result sharing and integration approach. Norihiro Yoshida et al. [6] reported a case study of code clone analysis on an industrial software system and detected code clones from two versions of source code of the target system: one was source code after unit testing; the other was source code after combined testing. Kanika Raheja1 et al. [8] devised an algorithm for detecting duplicity in the software by using hybrid software clone detection technique. The algorithm first computes the required software metrics that provide sufficient information regarding the software application and then depending on software metrics matches, the potential clone is detected. Rajkumar Tekchandani1 et al. [9] presented an algorithm that can detect semantically equivalent code fragments using formal grammars with the insight that the grammar recovery can be used to find semantically equivalent code fragments. They also proposed an algorithm to recover the grammar from parse trees. Robin Sharma [10] describes the hybrid approach combining textual and metric based approach, for detecting code clone. This approach is used to detect functional clones from source code. The output of the tools which developed based on this proposed approach are further compared with the existing tool in term of recall and precision parameter which shows that a hybrid approach is lightweight technique which gives accurate result being less complex. Al-Fahim Mubarak Ali et al. [11] produces source units using the pre-processing and transformation rules without effecting and compromising the information of the java source files. The proposed work was able to be implemented using Netbeans 7.3 as development platform. The runtime process result also shows the proposed work is faster as compared to tree based representation form process.

3. PROBLEM FORMULATIONS

In recent decades, the branch of clone detection has undergone a great advancement. This progress is due to the development of various methods, which involves the implementation of complex algorithms and tool chains to offer clone detection. Various clone detection methods that are already available include textual comparison, token comparison and comparison of Abstract Syntax Trees, Suffix Trees and Program Dependency Graphs. The most prevalent scalable and semantics-based approaches are restricted to the discovery of program fragments that are identical in their syntax or semantically equivalent control structures alone. Moreover, these Clone detection techniques are limited to a particular programming language environment only. The goal of the work is to illustrate a tool which is user friendly and is easy to maintain also and is not limited to small and big software. This method of clone detection can also be implemented to more complex applications such as web based applications i.e. a website code related to PHP or JSP or it can be an application which is linked with internet not a standalone application. In addition to this, the proposed approach is applicable to all the languages and platforms. Hence the proposed system is a platform independent system.

4. MATERIALS AND METHODS

This section can be divided into three subsections. In subsections, section one includes the information about the dataset. Sub Section two includes the evaluation criteria; using this evaluation metrics checks the performance of the proposed work. In lastly subsection three includes the proposed algorithm.

4.1. Objective of the Research

The objective of this study is to understand and analyze the concept of software Cloning and its detection. The following are various assumptions of the proposed work:

Table 1: Objectives of Research

Sr. No.	**Objective**	**Action/Steps**	**Expected Output**
1.	To develop a clone detection tool for object oriented languages.	Different languages source codes are given to the tool.	A tool that can detect clones of object oriented languages.
2.	To propose a novel clone detection technique for object oriented and platform independent language.	Finding various source codes in different languages and analyzing the clones in them using the hybrid approach.	Detection of clones.

3.	To validate the proposed technique with the existing tool.	Studying various existing tools of clone detection.	Validation of results.

4.2. Dataset Used

Some researchers have conducted comparative experiments of clone detectors. Bellon conducted the largest scale experiment [7]. Bellon's benchmark is one of the most famous benchmarks in the clone community. We compared four clone detectors from the perspective of accuracy and performance.

Table2: Showing the Bellon dataset Projects

Name	Language	LOC	No. Clone references
Netbeans	Java	14,360	55
ant	Java	34,744	30
weltab	C	11,460	275
cook	C	70,008	440

Table 3: Showing the Previous clone detection tools

Tools	Technique
Dup	Token based
CCFinder	Token based
CLAN	Metric
Duploc	Text based

4.3. Performance Evaluation Criteria

Performance is always a key factor in determining the success of any approach. We must first select some criteria called metrics for performance evaluation. There are different metrics used to measure the performance. Some of them are discussed in following:

- **Precision:** It is the probability that a randomly chosen candidate clone group is significant. It is the ratio of the number of relevant records retrieved to the total number of irrelevant and relevant records retrieved.

$$\textbf{Precision}=\frac{TP}{TP+FP} \quad \text{..... 1.1}$$

- **Recall:** It is the probability that a significant clone group chosen from the hypothetical set of all significant clone groups is enclosed in a detection result.

$$Recall = TP/(TP+FN) \quad \ldots\ldots\ldots\ldots\ldots\ldots 1.2$$

TP is an abbreviation for true positive i.e. these are the actual clones which are detected by the tool.

TN is an abbreviation for true negative i.e. these are the actual clones which are not detected by the tool.

FP is an abbreviation for false positive i.e. these are not the actual clones but are detected as clones by the tool.

FN is an abbreviation for false negative i.e. these are not the actual clones and also the tool didn't detect these

4.4. Present Work

The proposed tool is an implementation of a hybrid approach that combines metrics-based clone detection approach and token-based approach to detect clones. The technique is divided into two phases. In the first phase, metric based approach is used to detect the potential clones. Potential clones are selected on the basis of metrics match between the two source files. The metrics are calculated based on class level, function level and the threshold level is also defined for the matching of metrics. In the second phase, if the metrics match count reaches the threshold value then only actual clones are detected by using token based approach otherwise no need to calculate the actual clones as there is no potential clone between the source files.

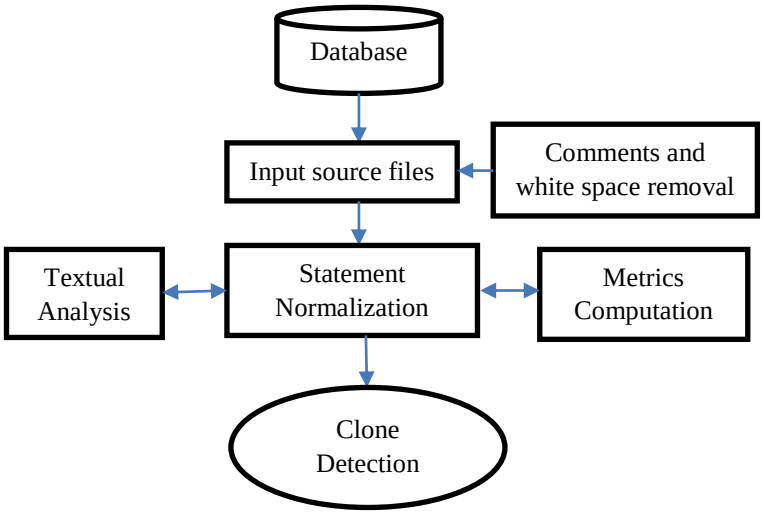


Figure: Clone Detection Architecture

The proposed clone detection process is divided into number of phase. As depicted in figure 1 above, Pre-processing, metrics computation, Tokenization approach, and detecting the clone types and results prediction. The proposed tool does not require any external parser so it contributes to less overhead than other methods.

1. **Pre-processing Phase**
2. **Metric Computation Phase**
3. **Tokenization Phase**
4. **Clone Detection Phase**

1. **Pre-Processing Phase:** This phase includes filtering out the uninteresting part from the source code which is not used in the comparison phases. It includes removal of white spaces, all types of comments and pre-processor statements like # defines and libraries import. The remaining source code portioned into a set of disjoint fragments known as source units. These units are involved in direct clone relations to each other. Depending on the comparison technique used source units may need to be further partitioned into smaller units.

2. **Metric Computation Phase:** Various class level and function level metrics are calculated to search for the potential clones. These metrics are mapped into excel sheets and then evaluation is performed. The comparison is performed on the basis of similarity between the metric values of two source files. If the number of metrics match reaches the threshold defined then on the basis of this result potential clones are identified. This technique can detect syntactic and semantic clone in the software or program but this technique is not so efficient i.e. not detect all possible clone present in the programs or software. The metrics that are calculated in proposed technique:

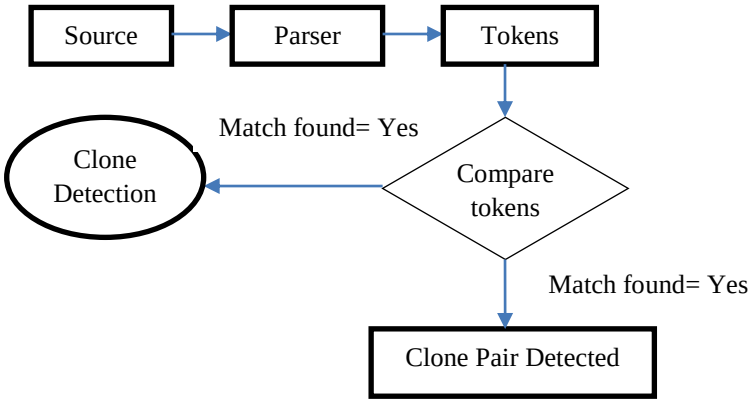
Class level metrics

1. LOC
2. No of private variable
3. No of public variable
4. No of protected variable
5. Total no of variable
6. No of loop control statements
7. Redirect statements
8. No of conditional statements
9. Friend variable
10. Private function
11. Public function
12. Protected function
13. Total no of functions

Function level metrics

1. Function name
2. No of local variables
3. No of Function calls
4. No of arguments/parameters passed in function
5. No of loop controls
6. No of return statements.

3. Tokenization Phase: If there exist potential clones between the two source files then only the token based approach is applied on the file otherwise no need to check the clones in the files as there is no existence of potential clones. This approach will remove the false positives from the potential clones as the metric based approach doesn't directly work on the code. A token based clone detection approach provides a suitable level of flexibility for the task. It involves limited language dependence, is resilient to the differences in code layout, and provides a good mechanism for detecting parameterized clones, where parametric differences are allowed between otherwise identical code fragments. This technique detects the code clone by comparing source code line by line in the form of token. In this technique, the source code has to convert into tokens with the help of parser or lexical analysis of the source code. This technique is better than the text based technique when the comments and spaces are present in the source code but it has low efficiency because while conversion source code to token sequence various false positive may introduction in the code. This technique is used to detect only type 1 and type 2 code clones.



Basic Token based technique

Proposed Token based approach includes following steps:

1. **Lexical analysis of the source**: Reading the source files and convert into tokens either using parsers or lexical. In our proposed technique, stream tokenizer is used to extract the tokens from the source files.
2. **Calculating tokens weight for each statement**: A weight is generated for each statement in a source file based upon the number of tokens in that statement e.g. statement contains if (flag). Here the number of tokens are 4 includes "if", "(","flag","")", therefore the weight of this statement is 4.
3. **Identifying similar token weight Sequences:** Similar token weights sequences are identified from weights sequences generated in step 2 by using the stream tokenizer. A sequence of 4 common weights are identified in order to depict the match i.e. threshold level for matching the token sequences is 4 and above.
4. **Mapping Identical Sub-sequences to source code:** The identical sub-sequences detected are mapped to the source code to find the actual clone pair in a file.

Example of Detection Process Using Proposed Technique

Source File 1

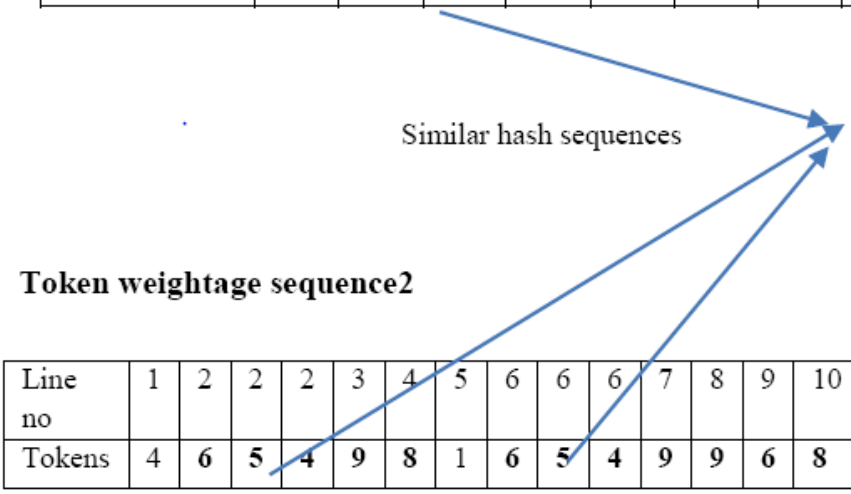
```
1. StringBuffersb = new StringBuffer ();
2. for (int j = 0; j <ss. length; j++) {
3. sb. append (ss[j]);}
4. sb. append (getComma ());
5. String res= sb. toString ();
6. System.out.println(res);
```

Source File 2

```
1. if (flag) {
2. for (int k=0; k<ss. length; k++) {
3. sb. append (ss[k]);
4. String rs = sb. toString(); }
5. } else {
6. for (int l=0; l<ss. length; k; l++) {
7. sb. append (ss[l]);}}
8. if (j % 2 == 0){
9. sb.append(",");}}
10. String rs = sb.toString(); }
11. return rs;
```

Token weightage sequence1

Line no	1	2	2	2	3	4	5	6
Tokens	7	**6**	**5**	**4**	**9**	**8**	**8**	8



Token weightage sequence2

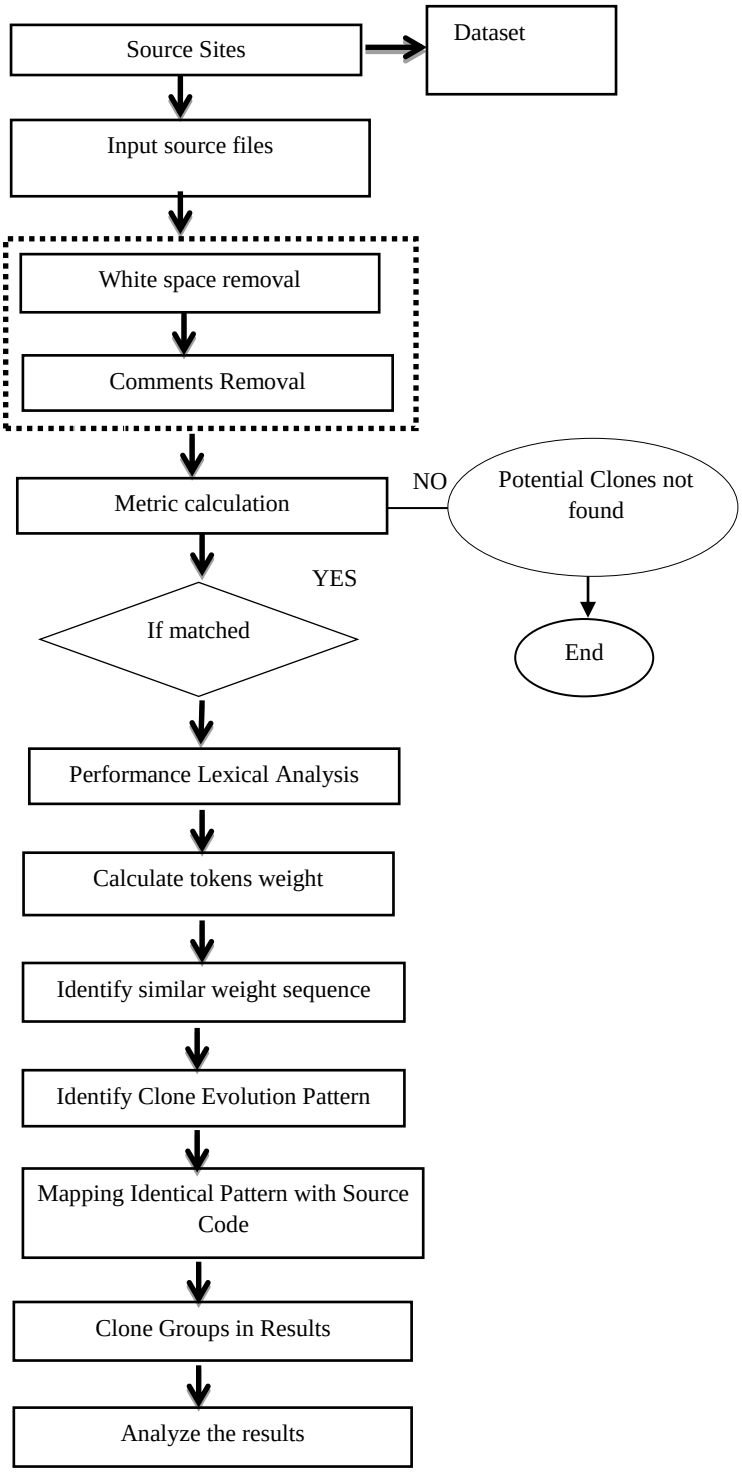
Line no	1	2	2	2	3	4	5	6	6	6	7	8	9	10	11
Tokens	4	**6**	**5**	**4**	**9**	**8**	1	**6**	**5**	**4**	**9**	**9**	**6**	**8**	2

Cloned File 1

```
1.  StringBuffersb = new StringBuffer();
2.  for (int j = 0; j <ss. length; j++) {
3.  sb. append (ss[j]);}
4.  sb. append (getComma ());
5.  String res= sb. toString ();
6.  System.out.println(res);
```

Cloned File 2

```
1.  if (flag) {
2.  for (int k=0; k<ss. length; k++) {
3.  sb. append (ss[k]);
4.  String rs = sb.toString(); }
5.  } else {
6.  for (int l=0; l<ss. length; k; l++) {
7.  sb. append (ss[l]);}}
8.  if (j % 2 == 0){
9.  sb.append(",");}}
10. String rs = sb. toString(); }
11. return rs;
```



Flowchart of propsed work

5. RESULTS AND DISCUSSIONS

We study that our proposed method detects the clones available within the supply documents in an efficient way. We have compared the proposed work with the already existing clone detection tool. From the simulation results proposed method perform well in terms of precision and recall. The graph obtained for ability clones is shown in following subsection.

5.1 The Bellon compared recall and precision between the target detectors. The results in Bellon's benchmark are summarized as follows.

1. Line-based and token-based detection techniques have high recall and low precision. This means that these techniques detect many clone references, but yield many false positives.
2. Metric-based detection techniques have low recall and high precision. This means that many of clones detected by these techniques are in clone references, even though these techniques miss many clone references.
3. Both metric and token based detection technique (Proposed Technique) have high recall and high precision i.e. technique detect many clone references and many of the clones detected by the technique are in clone references.

Table 4: Abbreviation of the Actual Clone

		Detected Clones	
		Yes	No
Actual clones	Yes	(TP)	(FN)
	NO	(FP)	(TN)

5.2 Comparison of Precision and Recall Results with Existing Techniques

In the experiment section we have compared proposed approach with existing approaches such as CC-Finder,CLAN, Duploc and Dup. We compared these four popular approaches with proposed clone detection approach in term of precision and recall. According to results it has observed that our technique has the best performance.

Table 5: Comparison of Precision of Proposed Technique with Existing Techniques.

Software Name	Dup	CC Finder	CLAN	Duploc	Proposed
NetBeans	0.28	0.025	0.49	0.35	0.91
Ant	0.49	0.20	0.79	0.49	0.80

We have compared our technique with four other techniques: CC-Finder, CLAN, Duploc and Dup. In comparison with these other techniques our technique has the best performance on Precision as shown in table 5.

Table 6: Comparison of Recall of Proposed Technique with Existing Techniques.

Software Name	Dup	CCFinder	CLAN	Duploc	Proposed
NetBeans	0.55	0.79	0.22	0.45	0.93
Ant	0.63	0.95	0.48	0.42	0.83

The comparison of proposed approach on recall metric is done with four other techniques: CC-Finder, CLAN, Duploc and Dup. According to the simulation results our technique has the best performance on Recall as shown in table 6.

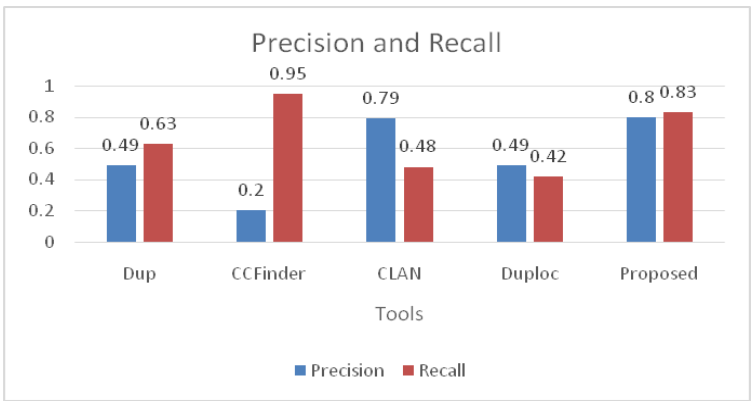


Figure 2: Comparison of Precision and Recall of Netbeans Bellon Project

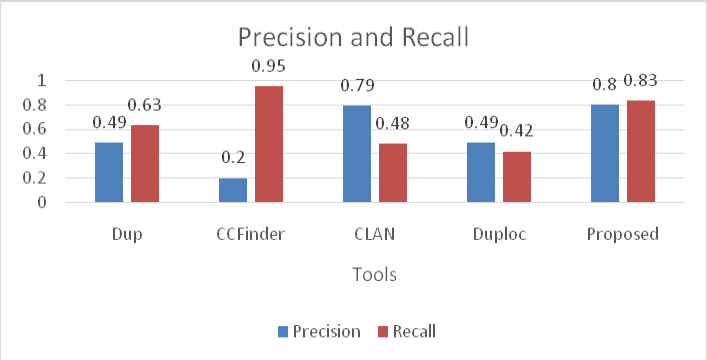


Figure 3: Comparison of Precision and Recall of eclipse Ant Bellon Project

Figure 2 and 3 illustrates the end to precision and recall of netbean bellon and eclipse ant bellon respectively. The blue column represents precision while the red column represents the recall. From the both graphs it has been clear the proposed technique providing better results.

6. CONCLUSIONS

From the study of various research papers, it can be concluded that cloning is a great demand today. Over the closing decade many strategies and equipment for software clone detection had been proposed. The current trend towards the use of multiple languages in the development of software systems introduces new challenges for software comprehension and software maintenance. In this paper we have presented a new code clone detection approach. Our method is able to identify clones within large source codes and is distinctive in its capability to detect code duplication independent of the source language. In this regard, an effort has been made on this paper to propose an efficient clone detection tool with high targets: Firstly, by using metric based method to find the capability clone. Secondly, to discover the clone via using token based method. We have described an approach for token based clone detection. This approach is compared with source code of netbean bellon and eclipse bellon respectively in terms of precision and recall. From the simulation results it has been observed that the proposed approach gives better results as compared to existing approach.

ACKNOWLEDGMENTS

The authors would like to thank all the participants in the experiment. This paper has been made with the kind help, direction and support of my supervisor who has helped me in this work. The authors also thank the anonymous reviewers for their valuable comments.

REFERENCES

[1]. Chanchal K. Roy and James R. Cordy, "An Empirical Study of Function Clones in Open Source Software", 1095-1350/08 $25.00 © 2008 IEEE.

[2]. Mark Gabel Lingxiao Jiang Zhendong Su, "Scalable Detection of Semantic Clones", ICSE'08, May 10–18, 2008, Leipzig, Germany. Copyright 2008 ACM.

[3]. Chanchal K. Roy, "Detection and Analysis of Near-Miss Software Clones", 978-1-4244-4828-9/09/$25.00 2009 IEEE.

[4]. Yoshiki Higo, and Shinji Kusumoto, "Enhancing Quality of Code Clone Detection with Program Dependency Graph", 2009 IEEE.

[5]. Iman Keivanloo, Juergen Rilling, Philippe Charland, "SeClone - A Hybrid Approach to Internet-scale Real-time Code Clone Search", 1063-6897/11 $26.00 © 2011 IEEE.

[6]. Norihiro Yoshida, Yoshiki Higo, Shinji Kusumoto, Katsuro Inoue, "An Experience Report on Analyzing Industrial Software Systems Using Code Clone Detection Techniques", 2012 IEEE.

[7]. S. Bellon, R. Koschke, G. Antniol, J. Krinke, and E. Merlo. Comparison and Evaluation of Clone Detection Tools. Transactions on Software Engineering 31(10):804–818, 2007.

[8]. Kanika Raheja1, Raj Kumar Tekchandani2, "An Efficient Code Clone Detection Model on Java Byte Code Using Hybrid Approach", 2013 IEEE.

[9]. Rajkumar Tekchandani1, Rajesh Kumar Bhatia2, Maninder Singh, "Semantic Code Clone Detection Using Parse Trees and Grammar Recovery", 2013 IEEE.

[10]. Robin Sharma, "Hybrid Approach for Efficient Software Clone Detection", IRACST – Engineering Science and Technology: An International Journal (ESTIJ), ISSN: 2250-3498 Vol.3, No.2, April 2013.

[11]. Al-Fahim Mubarak Ali, Shahida Sulaiman, "A Hybrid Technique in Pre-processing and Transformation Process for Code Clone Detection", 2014 IEEE.

[12]. Gurunadha Rao Goda, Avula Damodaram, "An Efficient Software Clone Detection System based on the Textual Comparison of Dynamic Methods and Metrics Computation", International Journal of Computer Applications (0975 – 8887) Volume 86 – No 6, January 2014.