FAST AND EFFICIENT SOURCE CODE PLAGIARISM IDENTIFICATION BASED ON TOKENIZATION

Dissertation submitted in fulfilment of the requirements for the Degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

SANDEEP KAUR

41500022

Supervisor

SANDEEP KAUR

(Assistant Professor)

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

November, 2017

ABSTRACT

When a code fragment is copied by another software system for reusing purpose, this is known as cloning. The code that is copied called clone. Clone Detection is a process that is used for detecting clones in a software system. Code cloning or software cloning makes the work of the programmers easier by reusing existing code. But it increases the maintenance cost. If any error is occurred in a code fragment that is cloned, then that error have to be removed from all cloned modules. So the code clone detection is very indispensable part of software engineering. There are so many techniques have been proposed for code clone detection. These are like Text based, Metric based, Token based, AST-based, and PDG-based. All these methods have some limitations. Some techniques take more time for clone detection like PDG and AST and some need to improve the efficiency.

So we proposed a tokenization based method in which Needleman Wunsch algorithm will be used for comparison of two source files with token values. Needleman Wunsch is a global sequence alignment method that is used for two end to end sequence alignments. It is based upon dynamic programming and also an optimal matching algorithm. It takes very less time for comparing two sequences and also identifying sequences with gaps efficiently as compared to previous techniques.

DECLARATION STATEMENT

I hereby declare that the research work reported in the dissertation entitled "FAST AND EFFICIENT SOURCE CODE PLAGIARISM IDENTIFICATION BASED ON TOKENIZATION" in partial fulfilment of the requirement for the award of Degree for Master of Technology in Computer Science and Engineering at Lovely Professional University, Phagwara, Punjab is an authentic work carried out under supervision of my research supervisor Mrs. Sandeep kaur. I have not submitted this work elsewhere for any degree or diploma.

I understand that the work presented herewith is in direct compliance with Lovely Professional University's Policy on plagiarism, intellectual property rights, and highest standards of moral and ethical conduct. Therefore, to the best of my knowledge, the content of this dissertation represents authentic and honest research effort conducted, in its entirety, by me. I am fully responsible for the contents of my dissertation work.

Signature of candidate

Name of the Candidate

R. No

SUPERVISOR'S CERTIFICATE

This is to certify that the work reported in the M.Tech Dissertation entitled "FAST AND EFFICIENT SOURCE CODE PLAGIARISM IDENTIFICATION BASED ON TOKENIZATION", submitted by **Sandeep kaur** at **Lovely Professional University, Phagwara, India** is a bonafide record of her original work carried out under my supervision. This work has not been submitted elsewhere for any other degree.

Signature of Supervisor

(Sandeep kaur)

Date:

Counter Signed by:

1) **Concerned HOD:**
   HoD's Signature: _____

   HoD Name: _____

   Date: _____

2) **Neutral Examiners:**

   External Examiner

   Signature: _____

   Name: _____

   Affiliation: _____

   Date: _____

   Internal Examiner

   Signature: _____

   Name: _____

   Date: _____

ACKNOWLEDGEMENT

Inspiration, co-ordination, patience and proper guidance are few of the components that lead to the successful and timely completion of any job. When doing a research, these play a vital role. I am highly thankful to my guide Mrs Sandeep kaur School of computer science and technology, Lovely Professional University, Phagwara under whose supervision I have had the privilege to conduct this thesis.

I would like to thanks to my teachers of department of computer science, Lovely Professional University, Phagwara for their continuous guidance and co-ordination during this research.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

When a programmer or developer uses the existing software code (by minor or extensive editing) for his software development, this procedure is known cloning and the code piece that is copied or reuse is called clone of real code[1]. The code fragments which are similar in their variables, literals or functions, are called clone pairs[2]. More than one pairs make a clone class. In post development phase, it is difficult to identify which codes are clones. Only the similarity or equality of their contents proved them as clones. Several studies of software clone detection represent the software with code clones are burdensome to handle as compared to without code clone. The tendency of code cloning not only produce maintenance issues, But also produce subtle errors. Because, if any error is introduced in one code segment then the entire identical fragments have to be changed. That is very time consuming and costly process in big software projects. It is also increases protection cost. The open source software and its variation also enhanced code reuse. There is no clear definition exist exactly what is clone till now. There are so many definitions according to different authors. Clone is also known as a duplicate. But not every identical code is a clone. Sometimes, two code segments can be same by chance. They are not copied by each other. According to Baxter et al. “Clones are portions of code which are identical according to some definition of sameness” [3]. According to this definition similarity is to be based on text, lexical, syntax or semantic representation.

Software clone detection is appears as a vital research area. These studies are suggested that 20-30% of large software systems consist of cloned code. Kamiya et al.[5] has reported 29% cloned code in JDK .Code clone detection can be useful in clone maintenance , copyright infringement detection, plagiarism detection, code simplification and detection of bug report[1].

1.1 CLONE TERMINOLOGIES

Clone detection process output clones in the sort of clone pair, clone classes or both. Exactly what is the meaning of these terminologies in clone detection process? Now we discuss here:

Clone fragment (CF): A code fragment is a method or function or sequence of statements that are needed to run a program.

Clone pair (CP): when two fragments are similar in syntactically or semantically or both are clones of each other then called them clone pair[1].

Clone class (CC): Many code fragments that contain two clone pairs are called clone class.

1.2 TYPES OF CODE CLONES

Types of code clone available in software systems are based upon their similarity between textual representation and functionality. So we can categorize them into following four types. First three types like Type1, Type2 and Type3 are textually or syntactically similar and Type4 is similar in its functionality [2][4][3][1].

Type1 (Exact clone): Program fragments which are similar but slight different in white spaces and comments as shown in Table 1.1

Table 1.1 Type 1 clone

FRAGMENT 1	FRAGMENT 2

//printing
for(h=1;h<=4;h++) {
print h;}
	for(h=1;h<=4;h++)
	{
	Print h;
	}

Type 2(Renamed clone): When two code fragments are syntactically identical but different in comments, layout, literals and identifiers. Reserved words and sentence formation are identical like real source code as shown in Table 1.2. We can see two code fragments are changed in their appearance, variables name and values.

Table 1.2 Type 2 clone

FRAGMENT 1	FRAGMENT 2

if(m>= n){
q= n+m;
else
q= n-m;
	if(n>=o)
	z=n+o;
	else
	z=n-o;

Type 3(Near Miss clone or Gapped clone): In this type of clone, there is a insertion and deletion gap between similar statements within two code fragments as shown in below Table 1.3

Table 1.3 Type 3 clone

FRAGMENT1	FRAGMENT 2

If(k>l)
{
l++;
k=1;
}
	if(o>p)
	{
	o=o/2; //statement inserted
	o++;
	}

Type 4 (Semantic clone): Semantic code clones are two segments that are similar in their functionality or semantically not syntactically or textually. As shown in below Table 1.4.

Table 1.4 Type 4 clone

FRAGMENT1	FRAGMENT 2

If(s>l)
{
u=s*l;}
else
u=s/l;
	switch(true)
	{
	case s>l:
	u=s*l;
	case s!=l;
	u=s/l;
	}

1.3 REASONS FOR CLONING

There are so many reasons for copying the code by programmers. Some of the reasons have been listed below [4] :

1. A programmer reuses the existing code, logic and design of a system by copy and paste operations. It is called code cloning or duplication.

2. When a programmer merges two similar systems to make a new system.

3. Clones are frequently occurred in the financial products. Because companies do not wants to take risks of new product's high rate of errors. So they reuse the already existed well tested codes for adapting to the new product [6].

4. Cloned fragments in the systems may improve maintenance. If all fragments will be independent, then we have to maintain them separately. This process will take more time.

5. Complexity of code: Programmer sometime find hard to understand large and complex code so they just copy the code.

6. Time limit: Time limit that is assigned to programmers to complete a product is very less. So they have to copy the existing clone.

7. Accidently: Sometime unintentionally code may be copied by the programmers for the solution of similar types of problems.

1.4 WEAKNESSES OF CLONING

Software cloning is widely used by all programmers. But it has so many adverse effects on software engineering. Some harmful effects of cloning are listed below [3] :

1. More maintenance cost: Software cloning increases the efforts of maintenance by duplicating multiple fragments.

2. Bug propagation: When an error is presented in a one code fragment that is copied at multiple places in a system. Thus the code cloning elevate the bug propagation.

3. Difficult to understand by maintainer: Because the maintainers have no information regarding duplicate fragments.

1.5 ADVANTAGES OF CLONE DETECTION

As there are so many flaws in code clone, yet so many benefits are existed. Some of the advantages of software cloning are discussed below [4] :

1. Helps in maintenance of software system by detecting code clones.

2. Detects Library functions by detecting codes that are reused again and again.

3. Detects plagiarism and copyright infringement.

4. Helps in code compactness by reducing the size of source code

1.6 STEPS OF CODE CLONE DETECTION

The task of clone detector is to search code fragments which have high sameness in a software real code. The major problem is searching of code segments that are duplicated. So the detector has to compare every code segment with every other possible segment. Such activity to estimate the similarities and dissimilarities is very costly from computation view. So before performing the actual comparisons, there are so many calculations are used to lessen the domain of comparison. In this section, this report provides a net summary of fundamental paces in a clone detection procedure. The following figure 1.1 [2] shows the all paces that a classic clone detector may follow. These steps are [2][4]

Preprocessing:

In this step, the source code is filtered by removing uninteresting parts and then subdivided and the field of the comparison is fixed.

Remove uninteresting source code fragment: In this step, the unwanted code which have no importance in comparison process are to be removed.

Fix comparison unit: In this pace, the Source code is to be separated into small scale units based upon the detection method. Source unit may be begin-end blocks, classes, files, functions or statements. This step determines the source units of code.

Transformation:

Transformation in clone detection process is a step that used to transformed source code into a suitable middle delineation for comparison. This is also called extraction. This extraction transformed target code to the form appropriate for the resource to the actual clone detection algorithm. Depending upon the technique, it can have following steps:

Token or lexical form: Here, each line of target code is chopped up into token values according to the lexical guidelines of the programming language of heed. The whitespaces and comments are detached from the sequences of tokens. CC-Finder and Dup are the prime techniques that utilize this kind of tokenization.

AST (Abstract syntax tree) form: In this, the entire source code is converted or parsed into Abstract syntax tree. The source units to be compared are then shown as sub-trees of the AST and comparison algorithm look for similar sub-trees to mark as clones.

Program dependence graph (PDG) analysis: Semantic based methods induce program dependence graph from the source file. The vertices of a PDG act as a substitute for the lines

and conditions of codes, edges indicate data and control dependencies. Source units are represented as sub-graphs of these PDG.

Normalization: This step is a voluntary pace knowing to remove specious difference such as removal of whitespaces, comments, formatting or structural transformation.

Clone Detection

After transformation, code is then input into a comparison function. Then it is differentiate to other code fragments employing a comparison tool to identify equal code portions.

Formatting

In this step of clone detection process, the clone code obtained as a result of previous step is converted into its original source code.

Filtering

This step is not performed by all clone detectors. Here, code clone are took out and a human specialist refine out the false positives clones. This manual analysis process is to be defined based on length, range or frequency.

Aggregation

While some tool results clone pairs not classes. Then to decrease the bulk of data, the clone pairs can be collected into mass, bunches or clone classes.

1.7 TECHNIQUES/TOOLS OF CLONE DETECTION

Today, so many clone identification methods are available. Some are based upon string based methods or some others are token based, PDG or AST based. Now we will discuss all these types of techniques and tools briefly.

TEXTUAL BASED METHODS

In this type of clone detection techniques, the source file code is first converted into sequence of lines or words. Then comparison is to be performed on them. If the contents of more than two code segments are similar in their maximum possible extent, are announced as pairs of clone or clone class by that detection method. These techniques returned a small number of false positives, are effortless in appliance and are language independent. The textual methods are mostly detects only type 1 clone. Here, we will discuss several well-known textual approaches as follows:[1][3][2]

Dup: Dup is a text based tool by Baker. This tool, first, transformed or normalized source code by removing comments, whitespaces and deals with identifier naming. It hashes each line of target files and then compared them with suffix-tree algorithm. This tool extracts only exact and near-miss clones. Accuracy of Dup is less.

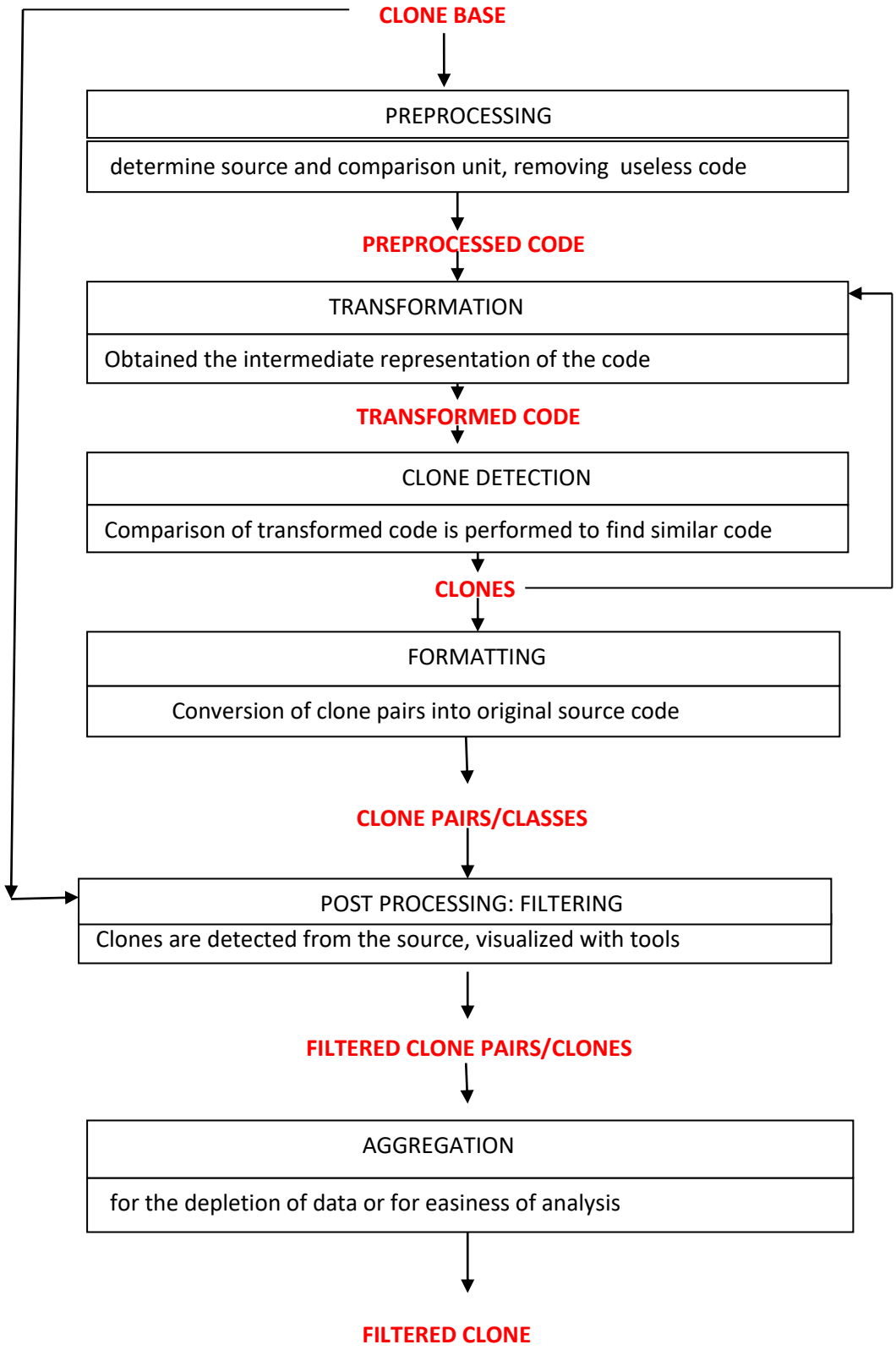


Figure 1.1 Clone Detection operations

Duploc: Duploc by Ducesse et al. is used two steps for clone detection. 1) First performs normalization by removing extra whitespaces, comments and change all characters to lowercase. This process decreases false positive and false negative by removing common constructs and insignificant difference between code clones respectively. 2) In the second step, string matching algorithm is to be used for comparing code line by line. The results of comparison are represented as a dotplot. That is too expensive in computational complexity.

NICAD: NICAD by Roy and James is a hybrid type clone detection technique used for detecting type 3 (near miss clone). It is mixer of two techniques of clone detection textual and AST. NICAD detects type 1, 2 and 3 code clones. There are three phases in it1) First, a parser extricates procedures and divides the separate fragments into lines.
2) In the second phase, normalization is to be performed using modification rules.
3) LCS algorithm is used to matching two code fragments. Each possible code fragment has to be compared with every other fragment. So it is very costly tool.

SDD: SDD by Seunghak and Jeong is a textual clone detection technique known as similar Data detection (SDD). This method is beneficial in matching in big size system for recognizing clones. Here, the index and inverted index for code segments and their locations are produced and then a n neighbor distance algorithm is used for detecting clone fragments [3].

LEXICAL APPROACH

Lexical approach is a token based approach in which the whole source code is parsed to a series of token values. Then the examination of sequence is to be performed for detecting duplicated subsequence of tokens and ultimately converts the resulted code clone into original code portions. As compared to text based approach, it is more durable averse to code variations such as formation and arrangement. The leading token-based approaches are:

CC-Finder: CC-Finder by Kamiya et al., a tool in which first the source codes are transformed into tokens by a lexer or parser and then all files of token values are merged into a single sequence of tokens. After that, the transformation laws of the language considered are applied. In the next step, the identifier related to variables, constants and types is substitute with a unique token. Then the suffix tree matching is to be used for identifying equivalent clone pairs or classes. At last phase clone pair is converted in the original source files. CC-Finder can handle page break relocation, name changes which cannot handle by

line-by-line approach. But it takes more CPU time and memory and it cannot detect type 3 clones.

CP-Miner: CP-Miner by Li et al. is token based technique that is used for detection of clones and bugs in large software system. CP-Miner uses frequent subsequence mining for copy-pasting code blocks. It performs two functions:1)**Detecting copy-pasted code fragment** 2) **Finding software bugs.** It detects gapped clone with the help of enhanced version of Clospan algorithm.

FRISC: FRISC by Murakami et al. is also a token-based technique which follows five steps:

1) It first normalizes the source files into token values and replaced all identifiers with an exceptional token.

2) Then the hash values are generated against every statement between";","{" and "}" with every token included in a statement. Then the numbers of repeated subsequence added to their first repeated subsequence and all repeated subsequence are removed.

4) Then detects similar subsequence from the wrapped hash sequence. If the minimum token length is larger than the number of tokens identified, they are not accepted as clone.

5) Then, transforms detected subsequence to the original target code. **FRISC** support java and c only.

Limitations of token based approach:

1) It based on the structure of program lines. If organization is changed in repeated fragment then code will not be recognized.

2) This approach cannot detect type 3 or gapped clone.

3) These techniques are very expensive in time and space complexity.

SYANTACTIC APPROACH

Syntactical techniques are categorized into two ways:

1 Tree Based Techniques: A program or source code is parsed to AST (abstract syntax tree) with a parser of the language which considered in tree based approach. Then by using same matching techniques, identical sub-trees are detected as clone classes or pairs. There are several tree based techniques are recently available. Here we will discuss only two types of tools:

i) CloneDR: CloneDR by Baxter et al is one of the pioneers AST based clone technique. In which a compiler generator is first used to convert source code into parse tree and then compared its sub-trees by depicting metrics based on a hash method through tree equalization. Then similar sub-trees source code is resulted as clone.

ii) Wahler et al: This technique detects clones by converting AST into XML with frequent itemset mining. Frequent itemset mining is used for searching sequence of actions or events that performed regularly.

2 Clone Detection Based On Metrics: Here, numbers of metrics are considered for comparison purpose instead of AST. The following seven metrics are mostly used by different authors:

1) Parameters
2) Function calls
3) Return statements
4) Executable statements
5) Conditional statements
6) Loop statements
7) Declaration statements

There are many metrics based techniques purposed. But here we explain only two in brief as follow:

Mayrand et al: It computes metrics from names, layouts, control flow of functions and expressions. When metrics of two functions are same, that functions are to be considered as clones.

Kodhai et al: It is combination of metric approach with a textual method to identify procedural clones in C language code. This method has five steps:

i) In first step, the source code is to be normalized by removing comments, whitespaces and preprocessed statements.

ii) Arrangement transformation is used in the textual comparison of potential clone. It changes the name of variables, data types and functions.

iii) Clone detection detects each function and recognize it.

iv) Acme calculations.

v) exact clone and renamed clone detection.

Limitation of syntactic approach:

1) AST tools are unable to deal with identifiers and literal values.

2) They cannot handle rearranged statement clone.

3) Metrics based methods require a parse or PDG producer to get metrics value.

SEMANTIC APPROACH

In this approach, semantically different fragments which execute same function are detected. This approach further of two types:

1) PDG (Program Dependency graph) Based

PDG based approaches uses semantic information of the source for getting a source code representation of big abstraction than other approaches. It contains control flow and data flow information of a source program and also keeps semantic information. The isomorphic sub-graph methods are used for comparing a set of PDG. PDG based clone detection techniques are as follows:

Komondoor and Horwitz'sPDG-DUP: Which proposed isomorphic PDG sub-graph using program slicing. There are three steps its follows:

1) First the pairs of clones are searched by dividing all PDG vertices into identical classes. The matching nodes are vertices existed in the same class.

2) Discard subsumed clones.

3) The clone pairs are clustered into larger groups using transitive closure.

GPLAG: It is proposed by Li et al. and used to detect plagiarism by mining PDGs.

2) Hybrid Based: Compounding of more than two techniques are called hybrid techniques. A hybrid technique is used for removing weaknesses arised by individual tools or techniques. For example:

Funare et al: It is a hybrid tool that is combination of AST and Text based methods. It follows three steps:

a) Formation of AST.

b) Serializing the AST and cryptograph into a sequence of strings with an inverse mapping procedure.

c) Then detecting clones.

d) Reconstructing clones.

Limitations of Semantic Approach

1) PDG tools cannot extendable to large system.

2) Need a PDG generator.

CHAPTER 2
REVIEW OF LITERATURE

The clone detection in coding has become a very major concept of research these days. So many techniques have been proposed for code clone detection by different researchers. In this chapter, the literature review about papers or topics related to software cloning will be given.

Chanchal kumar roy and James R.Cordy(2007)[4]**:** In this survey of code clone detection, the terms of clone commonly used in the literature like clone pairs, clone class and clone fragment are described. It is also discussed clone types which are commonly used. Second, this paper provides a review of detection techniques, different clone taxonomies and experimental evaluations of different tools of detections.

Roy, James and Rainer Koschke(2009)[2]**:**This paper gives us the very basic information regarding code cloning like clone types, clone detection steps and all-inclusive of recent techniques and tools. There are two different dimensions used for comparing, classifying and evaluating all tools and techniques. First, it divide and compare methods based on a number of aspects based on usages, interaction, language, clone ,technical, adjustment etc. and second a predictive scenario-based approach is followed that estimate maximal potential of each clone detection technique.

Dhavleesh Rattan et al.(2013)[3]**:** This survey is a systematic literature review of software clone detection that is based on 213 articles, 37 premier conferences and workshops. In this review, 9 different types of clone, model based and semantic clone detection description, 13 intermediate representations are reported.

Abdullah sheneamer, Jugal kalita(2016)[1]**:**This study gives us the benefits and flaws of all available clone detection techniques and tools by employing measurement based on precision, F-measure and recall metrics, scalability and portability. The goal of this study is to compare the recent status of the tools and techniques and highlight the future scope of them.

Toshihiro kamiya et al. (2002)[5]**:**In this paper, Kamiya et al. has proposed a tool named cc-finder which detects code clone in so many languages like java, c++, COBAL and other source files. It first transform the input source text files into token sequences and applies rule-based transformation to the sequence and then perform token by token comparison with

suffix tree matching algorithm for extracting clones.

Rainer Koschke et. al(2006)[7]**:** In this paper, the author has proposed a method for finding syntactic clones in linear time and space by using combined approach of tokenization and AST(Abstract syntax tree). Here, first source files are parsed and formed AST. Then serialize AST and employ Suffix tree method for detection of syntactic clones.

Hamid Abdul Basit(2007)[8]**:** In this research, author has proposed a clone detection tool called RTF(Repeated Tokens Finder). The working of this tool is as:

1 First the source program is transformed into a string of token values. 2. A suffix array based algorithm is used for computing clones in the string of tokens 3 Then, to get rid of the probable false positive, the pruning that is based upon a heuristic is used [8].

Warren Toomey(2012)[9]**:** The author has developed a tool called CT-Compare. It uses a noval tokenization approach. Each source code file is first parsed into tokens with lexical analyzer and then divided into tuples of N successive tokens. The tuples are then hashed and the hashed sequences are used to detect type 1 and type 2 clone pairs[9].

Yang Yuan and Yao Guo [10]**:**In this report, Boreas, a scalable and accurate token based approach for detecting code clone[10]. Here, variables are used instead of matching sequences or structures. Using Counting-Based characteristics matrix, the similarity of two code fragments is identified by using the characteristics of proportion variables.

Rajnish kumar [11]**:** This research paper has proposed a clone detection technique using program slicing with a token based matching algorithm. This method detects type1, 2 and type 3 code clone and also best for detecting non contiguous code clone.

Benjamin Hummel et al. [12]**:** The author has proposed a index based method for detection of exact and renamed clones. This method is both scalable and incremental to very big code base and deal with real-time detection in large system.

Kodhai et al.(2014) [13]**:**In this paper, author has proposed a clone manager named tool based upon metrics and textual analysis for detection of procedure level semantic and syntactic clones in c and java projects.

Vera Wahler et al.[14]**:**In this paper, first the source files from java, c++ or prolog are parsed into XML and then frequent data mining technique is applied for extraction of clones.

Zhenmin Li et al(2006)[15]**:**The author gives us a tool called CP-miner. It uses "clospan" frequent sub sequence mining algorithm for code clone detection. The procedure of this tool as:

1) First parse the source code into tokens. 2) Then do mining for basic duplicate segments. 3) Then prune false positive. 4) Then compose larger copy-pasted segments.

Hamid Abdul Basit(2009)[16]**:**In this method, the author has represented a technique that is used for structural or higher level clone detection with the application of data mining techniques.

Hiroaki Murakami(2016)[17]**:**In this dissertation, the author has proposed two clone detection techniques that upgrade the existing weaknesses. The first technique is a token based technique that folds every repeated instruction for reducing uninteresting clones and then apply token based detection techniques. This tool is called FRISC. Second technique detects gapped clone applying a local sequence alignment method named Smith Waterman (SW). This algorithm is faster and more accurate than previous algorithms like LCS, suffix-tree, suffix-array, PDG-based etc.

Hiroaki Murakami(2013)[18]**:**In this paper, author has proposed a tool named CDSW. This tokenized based tool uses the smith waterman algorithm for clone detection. This is very efficient algorithm and takes less time as compared to previous algorithms.

Abhilash CB [19]**:** This is a comparative study on global and local alignment. This study puts light on all concepts related to pair wise sequence alignment that are of further two types: Global sequence alignment algorithm like Needleman Wunsch algorithm and Local sequence alignment algorithm like Smith Waterman algorithm.

Shakuntala Baichoo(2017)[20]**:** This paper is a review of sequence analysis method's time and space complexity and put light on their properties in a inclusive way[20]. Here, the memory capacity and time complexity of Needleman wunsch and Smith Waterman algorithm is compared in which Needleman wunsch algorithm proved better in time and both are equal in space complexity.

CHAPTER 3
PRESENT WORK

3.1 PROBLEM FORMULATION

Code clone detection is major area of research for developers recently. So many techniques and tools have been proposed for clone detection. Some tools detects type1 (exact clone), type 2(renamed clone) and some detects all types including gap clone. All these techniques are based on Text-based, Metric-based, Token-based, PDG (Program dependency graph) based or AST (Abstract Syntax Tree) based techniques. Each of these tools has its own strengths and weaknesses according to their way of represented code, granularity and matching algorithms. There are so many matching algorithms have been proposed before like LCS, Suffix-tree, Suffix-array, Smith waterman, PDG-based and AST-based etc.

The Smith Waterman algorithm is a tokenized based local Sequence alignment algorithm that detects the region of highest similarity including gaps between two sequences[18]. But this does not align the edges of similar sequences. The SW makes more comparisons when matching the score to 0 and when searching the largest value score during the trace back [20].

To alleviate the above problems in SW algorithm, the proposed technique in this research work is used tokenized based technique in which clones are detected by using Needleman Wunsch algorithm. NW is a global sequence alignment method which performs end to end comparisons between two potentially equivalent sequences and identifies the equivalent alignments even if they contain some gaps.

Comparisons of time and space complexity between Smith Waterman and Needleman Wunsch(NW)

Time complexity of NW algorithm: Following step by step approach is to be followed for analyzing time complexity:

1. Determine the first row and column: **O(r + s)**

2. Scoring in the table with all the values T (i,j) where i ranges from 1…r and j ranges from 1…s computing the scoring function : **O(r x s)**

3 The trace back : **O(r + s)**

4 Finding a final path, suppose s>r. so max(r, s): **O(s)**

This overall time complexity is

O(r + s) + O(rs) + O(r + s) + O(s)

When r and s are very large, the lower order terms are discarded and thus the total time complexity is in the **O(rs)** [20].

Time complexity of Smith Waterman: Same procedure is followed for analyzing complexity of SW:

1. Determine the first row and column: **O(r + s)**

2. Filling in the table : **O(r x s)**

3 The trace back takes: **O(r x s)**

4 Finding a final path, if suppose s > r this step can take max(r, s): **O(s)**

This overall time complexity is

O(r + s) + O(rs) + O(rs) + O(s)

When r and s are very large, the lower order terms are ignored , So the total time complexity is in the **O(2rs)** [20] .

Thus whole running time of SW is lesser than that of Needleman [20].

Space complexity of both: Space consumed by both algorithms is equal O(rs). Because both use same matrix and same quantity of space is required for trace back [20].

Accuracy: Accuracy of NW algorithm is better than SW when the sequence being aligned might be quite diverse[20].

3.2 OBJECTIVE OF THE STUDY

1) To propose a fast and efficient source code plagiarism identification technique based on tokenization

1.1 To perform preprocessing of input files.

1.2 To perform lexical analysis and normalization on preprocessed files for generating token

sequence.

1.3 To calculate hash value for every statement.

1.4 To identify same hash sequence using Needleman Wunsch algorithm.

1.5 To identifying gapped tokens.

1.6 To map identical subsequence to the source code.

2) To perform empirical comparison of proposed approach with the existing base paper approach in terms of recall, F-measure and precision.

3.3 RESEARCH METHODOLOGY

Our proposed method is a fast and efficient source code plagiarism identification technique. First of all source code files are transformed into tokenized form and then compared by Needleman Wunsch algorithm.

Introduction to Needleman Wunsch(NW)

The NW algorithm[21] is a global sequence alignment method that is an example of dynamic programming. NW provides a mechanism of acquiring the best global alignment of two sequences including gaps by dividing the problem into several sub parts. There are following steps that to be followed by NW:

1 Initialized a Matrix: First the initial matrix is created with M+2columns and N+2rows (where M & N are length of two sequences). Extra row and column is given, so as to align with gap at the starting of matrix as shown in below figure 3.1.

Suppose mismatch, match and gap are chosed as -2, 2, and -1 and the two sequences are

DLIMDIILD

DLMLDILD

2 Scoring the Matrix: In second step, matrix filling is performed from the upper left hand

	-	D	L	I	M	D	I	I	L	D
-	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
D	-1									
L	-2									
M	-3									
L	-4									
D	-5									
I	-6									
L	-7									
D	-8									

Figure 3.1 Table initialization

Corner. Scores of each cell in matrix are calculated by using following formula:

$$M(i, j) = (Max(M_{i-1,j-1} + S(A_i , B_j)$$

$$M_{i-1, j} + gap, M_{i, j-1} + gap)$$

$$S(A_i, B_j)=\{ match (a_i = b_j) , mismatch (a_i != b_j)$$

In above example of sequences match= 2, mismatch= -2, gap= -1

	-	D	L	I	M	D	I	I	L	D
-	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
D	-1	2	1	0	-1	-2	-3	-4	-5	-6
L	-2	1	4	3	2	1	0	-1	-2	-3
M	-3	0	3	2	5	4	3	2	1	0
L	-4	-1	2	1	4	3	2	1	4	3
D	-5	-2	1	0	3	6	5	4	3	6
I	-6	-3	0	3	2	5	8	7	6	5
L	-7	-4	-1	2	1	4	7	6	9	8
D	-8	-3	-2	1	0	3	6	5	8	10

Figure 3.2 scoring matrix

3. **Trace back the matrix:** The diagonal movements during trace backing indicate a match and vertical and horizontal movements show gaps. The figure 3.3 is example of trace back step of above example:

-	-	D	L	I	M	D	I	I	L	D
-	0	-1	-2	-3	-4	-5	-6	-7	-8	-9
D	-1	2	1	0	-1	-2	-3	-4	-5	-6
L	-2	1	4	3	2	1	0	-1	-2	-3
M	-3	0	3	2	5	4	3	2	1	0
L	-4	-1	2	1	4	3	2	1	4	3
D	-5	-2	1	0	3	6	5	4	3	6
I	-6	-3	0	3	2	5	8	7	6	5
L	-7	-4	-1	2	1	4	7	6	9	8
D	-8	-3	-2	1	0	3	6	5	8	10

Figure 3.3 Trace backing

This analysis results the following sequence:

D L I M – D I I L D
| | | | | | | | |
D L – M L D I – L D

Procedure of research methodology

The proposed method will be able to detect code clone of Type 1(exact clone), Type 2(Syntactic clone) and type 3(gapped clone). This method consists of following procedure:

1. First, it takes following inputs

a) Source files

b) Total Number of tokens

c) Maximum gap rate

d) Score parameters (match, mismatch, gap)

2. Performing Parsing and Transformation

3. Statement hash

4. Recognizing identical hash sequences

5. Discovering tokens with gaps

6. Mapping to the original code

The following Figure 3.4 Shows the procedure of proposed method.

2) Performing Parsing and Transformations

The input target files are parsed into token values. The special tokens are used instead of user defined identifiers. So that code clones can be find which have exactly not same variables.

3) Statement hashing

In statement hashing, every statement that is between ";", opening brace ("{") and closing brace ("}") has been assigned a hash value. A number of tokens in every statement are attached to its hash value.

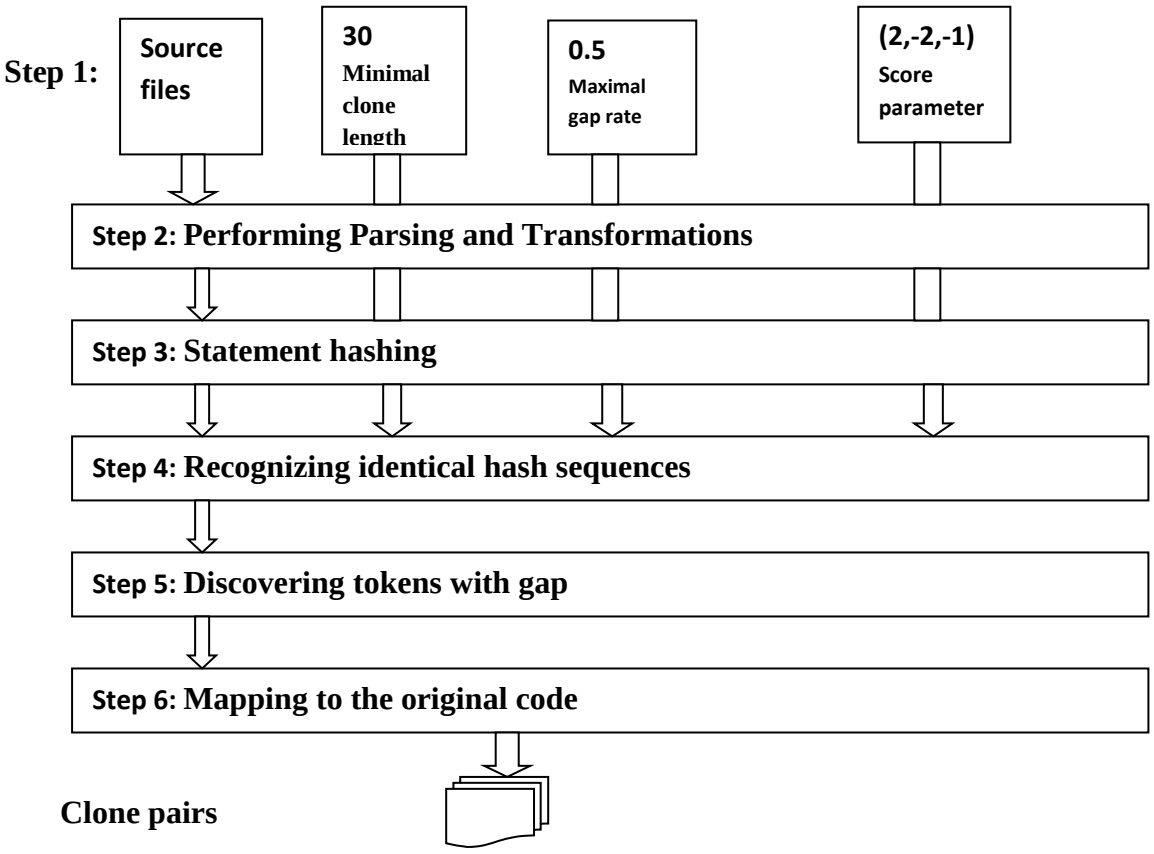


Figure 3.4 Paces of proposed method

4) Recognizing identical hash sequences

The Needleman Wunsch algorithm is used for recognizing similar hash sequences. Trace back is done from lower right corner towards upward by following maximal values and end at 0 value.

Step 5: Discovering tokens with gaps

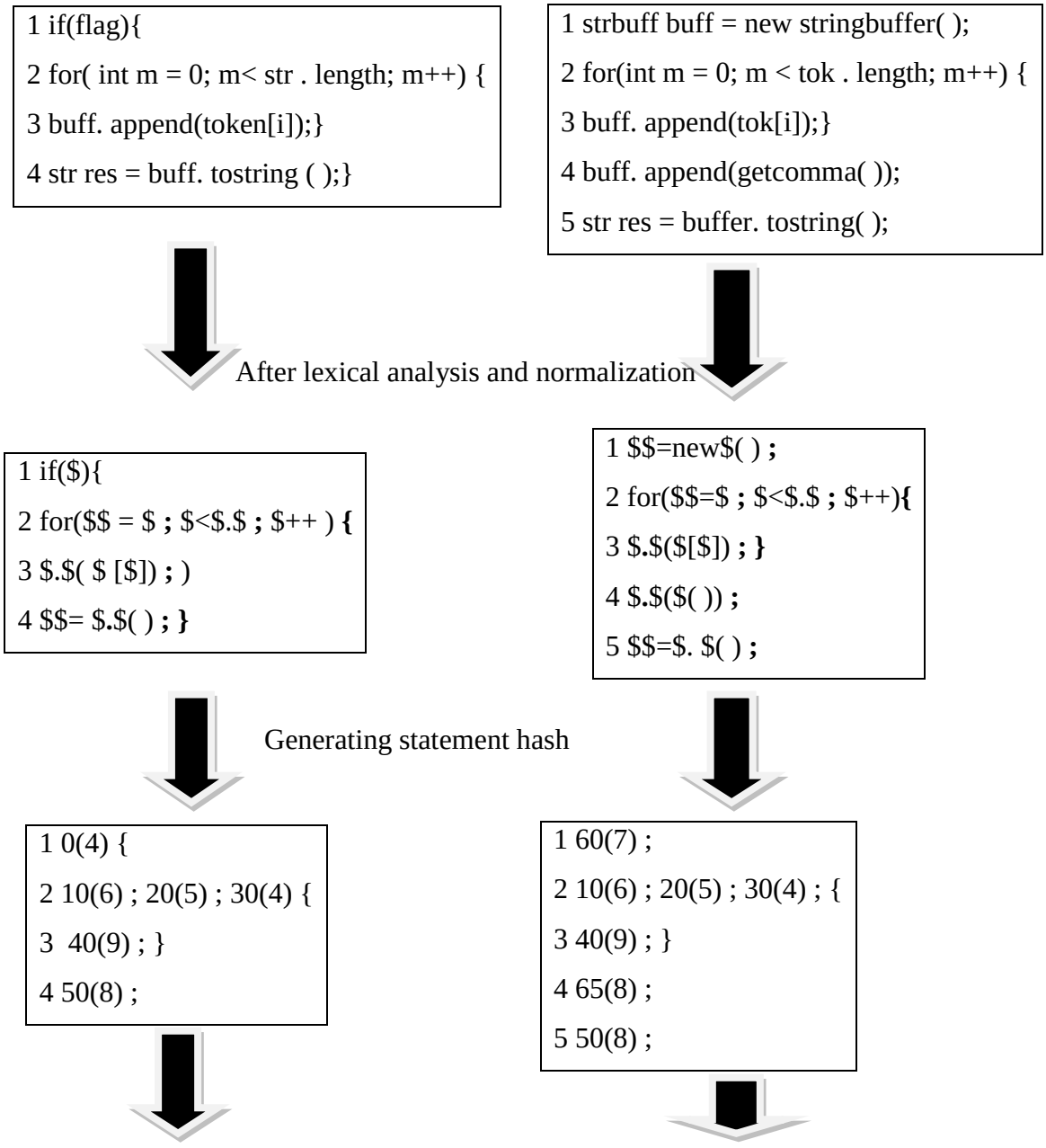
The LCS algorithm is used to identify token-level gaps.

Step 6: Mapping to the source code

This step is used for mapping of code clone detected in step 4 and 5 into original source code.

A EXAMPLE SHOWS PROPOSED METHOD

First take following two source files code as input:



Generating hash sequences

0 10 20 30 40 50

60 10 20 30 40 65 50

Creating and initializing table and scoring

_	_	60	10	20	30	40	65	50
_	0	-1	-2	-3	-4	-5	-6	-7
0	-1							
10	-2							
20	-3							
30	-4							
40	-5							
50	-6							

_	_	60	10	20	30	40	65	50
_	0	-1	-2	-3	-4	-5	-6	-7
0	-1	-2	-3	-4	-5	-6	-7	-8
10	-2	-3	0	-1	-2	-3	-4	-5
20	-3	-4	-1	2	1	0	-1	-2
30	-4	-5	-2	1	4	3	2	1
40	-5	-6	-3	0	3	6	5	4
50	-6	-7	-4	-1	2	5	4	7

Trace backing

_	_	60	10	20	30	40	65	50
_	0	-1	-2	-3	-4	-5	-6	-7
0	-1	-2	-3	-4	-5	-6	-7	-8
10	-2	-3	**0**	-1	-2	-3	-4	-5
20	-3	-4	-1	**2**	1	0	-1	-2
30	-4	-5	-2	1	**4**	3	2	1
40	-5	-6	-3	0	3	**6**	**5**	4
50	-6	-7	-4	-1	2	5	4	**7**

Similar hash sequences

gap

0 10 20 30 40 - 50	**60 10 20 30 40 65 50**

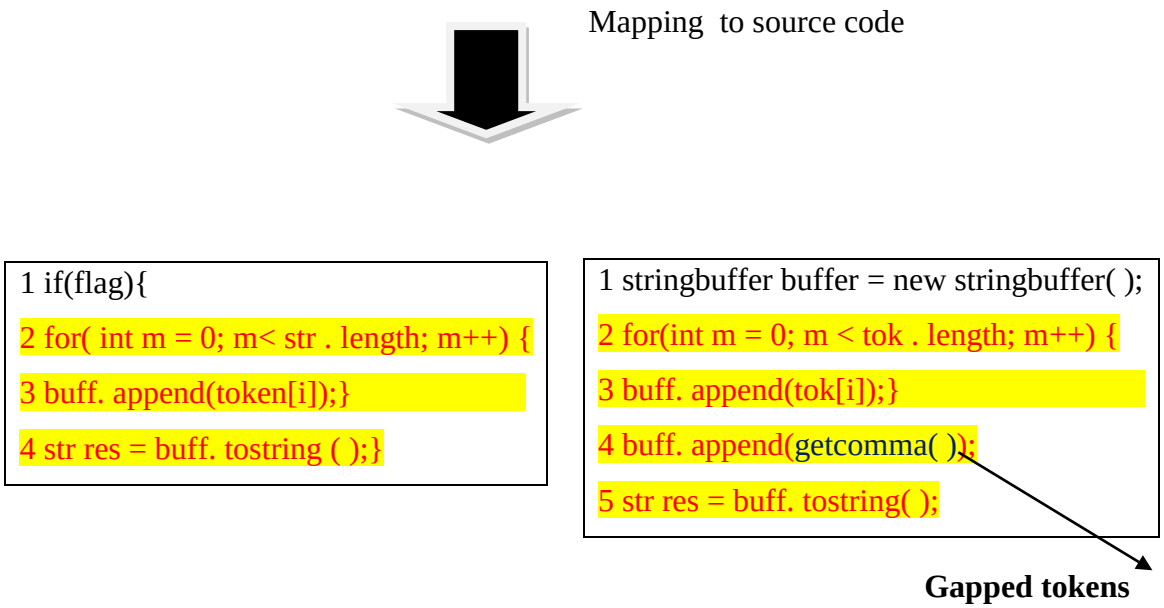


Figure3.5 Example of proposed method

EXPERIMENTAL DESIGN

we will used freely available data set called Bellon's benchmark for comparing the F-measure, recall and precision of proposed method. This data set contains eight software systems information. We compute precision , recall and F-measure for evaluating proposed method. These three parameters are shown below:

$$\text{Recall} = |S_R| / |S_{refs}|$$

$$\text{Precision} = |S_R| / |R|$$

$$\text{F-measure} = 2 \times \text{Recall} \times \text{Precision} / \text{Recall} + \text{Precision}.$$

CONCLUSION

The proposed method is a technique which detects code clone in fast and efficient way. Here, we are tried to find a better comparison algorithm. Needleman Wunsch algorithm is used instead of Smith Waterman. Needleman Wunsch algorithm is better in speed and efficiency than Smith Waterman. It is a global sequence alignment method which performs end to end comparison and more accurate also.

REFERENCES

[1] A. Sheneamer and J. Kalita, "A Survey of Software Clone Detection Techniques," *Int. J. Comput. Appl.*, vol. 137, no. 10, pp. 975–8887, 2016.

[2] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci. Comput. Program.*, vol. 74, pp. 470–495, 2009.

[3] D. Rattan, R. Bhatia, and M. Singh, *Software clone detection: A systematic review*, vol. 55, no. 7. Elsevier B.V., 2013.

[4] C. K. Roy and J. R. Cordy, "A Survey on Software Clone Detection Research," *Queen's Sch. Comput. TR*, vol. 115, p. 115, 2007.

[5] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. IEEE Trans Softw Eng," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 654–670, 2002.

[6] J. R. Cordy, "Comprehending Reality: Practical Challenges to Software Maintenance Automation," *Int. Work. Progr. Compr.*, pp. 196–206, 2003.

[7] R. Koschke, R. Falke, and P. Frenzel, "Clone detection using abstract syntax suffix trees," *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 253–262, 2006.

[8] H. A. Basit, S. J. Puglisi, W. F. Smyth, A. Turpin, and S. Jarzabek, "Efficient token based clone detection with flexible tokenization," *6th Jt. Meet. Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng. companion Pap. - ESEC-FSE companion '07*, p. 513, 2007.

[9] W. Toomey, "Ctcompare: Code clone detection using hashed token sequences," *2012 6th Int. Work. Softw. Clones, IWSC 2012 - Proc.*, pp. 92–93, 2012.

[10] Y. Yuan and Y. Guo, "Boreas: an accurate and scalable token-based approach to code clone detection," *Proc. 27th IEEE/ACM Int. Conf. Autom. Softw. Eng. - ASE 2012*, p. 286, 2012.

[11] R. Kumar, "Token based clone detection using program slicing," vol. 5, no. August, pp. 1537–1541, 2014.

[12] B. Hummel, E. Juergens, L. Heinemann, and M. Conradt, “Index-based code clone detection: incremental, distributed, scalable,” *Softw. Maint. (ICSM), 2010 IEEE Int. Conf.*, pp. 1–9, 2010.

[13] E. Kodhai and S. Kanmani, “Method-level code clone detection through LWH (Light Weight Hybrid) approach,” *J. Softw. Eng. Res. Dev.*, vol. 2, no. 1, p. 12, 2014.

[14] V. Wahler, D. Seipel, J. Wolff, and G. Fischer, “Clone Detection in Source Code by Frequent Itemset Techniques.”

[15] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, “CP-Miner : Finding Copy-Paste and Related Bugs in Large-Scale Software Code,” vol. 32, no. 3, pp. 176–192, 2006.

[16] H. A. Basit, S. Jarzabek, and I. C. Society, “A Data Mining Approach for Detecting Higher-Level Clones in Software,” vol. 35, no. 4, pp. 497–514, 2009.

[17] I. Science, “Fast and Precise Token-Based Code Clone Detection January 2016 Hiroaki Murakami.”

[18] H. Murakami, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto, “Gapped code clone detection with lightweight source code analysis,” *IEEE Int. Conf. Progr. Compr.*, pp. 93–102, 2013.

[19] A. C. B, “A Comparative Study on Global and Local Alignment Algorithm Methods,” *Int. J. Emerg. Technol. Adv. Eng. Website www.ijetae.com ISO Certif. J.*, vol. 9001, no. 1, pp. 34–43, 2250.

[20] S. Baichoo and C. A. Ouzounis, “Computational complexity of algorithms for sequence comparison, short-read assembly and genome alignment,” *BioSystems*, vol. 156–157, pp. 72–85, 2017.

[21] S. B. Needleman and C. D. Wunsch, “A general method applicable to the search for similarities in the amino acid sequence of two proteins,” *J. Mol. Biol.*, vol. 48, no. 3, pp. 443–453, 1970.