Webology (ISSN: 1735-188X)
Volume 19, Number 2, 2022

Detection Of Code Clones In Open Source Software Using Proposed Generalized Template

Ms. Harpreet Kaur[1*]

[1]Department of Computer Engineering, Punjabi University, Patiala, INDIA.

Abstract

The objective of current research work is to identify Function-level clones with lesser time and space complexity along with increased accuracy. A generalized template has been proposed on the basis of reserved keywords, object creation, and built-in-functions that Java Language exhibits. The presented methodology has been verified on an open source project Jhot Draw, which is purely a Java based freely available Drawing Framework. The advantage of the proposed template is that it gives a short intermediate form of the source code, which leads to signature generation for extracted functions in reduced time and space. It also leads to minimum false positives. The results have been compared with the available literature. The schema provided will help in building repositories of functions because functions are the most reusable part of any software.

Keywords: Granularity; Template; Function Clones; Signature; Complexity;

1. INTRODUCTION

Copying code fragments and reusability of code (known as "Code Cloning") possesses many positive and negative impact on software development. In relation to software reuse, the prominent advantage of cloning is that it is the simplest way to faster development of software. It provides quick start to the development team, and a faster solution to the problem. But from a viewpoint of program analysis, if any bug propagates in reusable-code, it will produce error-prone code clones and will cost more maintenance effort. Code fragments that are copied for development purposes are customized according to the requirement by the developers. Therefore, code fragments are generally differently evolved from the original fragments. And it generates gaps between the original code fragments and the copied code fragments. In order to detect code clones appropriately, it is of utmost importance to detect these gaps. Detection of gapped clones is necessary to better understand clones and the software systems. This issue can only be managed by providing totally new re-build framework. [1].Granularity plays an important role in detection of clones. Granularity varies from coarse-grained to fine-grained level. Clones carry **important** domain knowledge, and that knowledge can only be accessed by identifying clones at a particular level of granularity. Existing Tools and Techniques to detect clones, each operates at a different level of granularity such as character-level, statement level, function (method) level, file level etc. Larger the granularity, fewer will be the clones and will lead faster searching process, ands of course, it will provide some meaningful information because long text possesses some meaning. Keeping the conceptin mind, the research carried out in this paper focuses on function/method level clone detection from source code, because function/method possesses maximum functionality of the software. A general example of function clones is shown in table 1 below.

Table 1: General Example of Function/Method level clones

Original Function A	Function A1	Function A2	Function A3

Intprint Token( )
Inta,i;
Long int b;
While (is number(b))
{ if (a==token_gen)
System. out.
println("token
generated")
i++;
return i;
}
	Intprint Token( )
	Inta,i;
	Long int b;
	While (is number(b))
	{ if (a==token_gen)
	System.o ut.
	println("token
	generated")
	i++;
	return i;
	}
		Intprint Token( )
		Intx,j;
		Long int y;
		While (is number(y))
		{ if (x==token_gen)
		System.
		Out .println("token
		generated");
		j++;
		return j;
		}
			Intprint Token( )
			Intx,j;
			Char d;
			Long int y;
			While (is number(y))
			{ if (x==token_gen)
			System.out.println
			("token generated");
			If(d==' ')
			System.out.println
			("null");
			j++;
			return j;
			}

The emphasis here is to detect function-clones, but the big challenge is to propose a generalized template, which will serve as a basis for clone detection. A generalized intermediate transformation has been implemented in the proposed template using Java Language. Proposed template possesses some advantages as compared to literature such as it adopts a lightweight approach, which is free from parsing issues and relies only on representation of code fragments in the chosen intermediate format.

Structure of the paper: Related Work is discussed in Section II. Section III introduces the overall methodology. Results and discussion have been discussed in Section IV. Section V discusses Threats to Validity. Section VI concludes the research work. At last, some of the results have been shown in Appendix A.

1.2 Contributions/ Highlights

The primary contributions/highlights of this paper are: 1)An intermediate template has been proposed by adopting proposed encoding scheme for all extracted functions of the system under investigation 2) The proposed encoding scheme consumes less memory because it uses character-based nomenclature instead of numeric in contrast to literature 3) shortest encoding nomenclature limited to two characters has been adopted in contrast to literature. 4) Provide a schema for generating fingerprints based on function granularity. 5) Function level granularity has been chosen which leads to large portion and functionality coverage of software, rather than concentrating on short sentences 6) Function level similarity supports code refactoring, because it is easy to apply refactoring methods on functions.

2. RELATED WORK

An impressive measure of research has been directed for clone detection in software development [8-10]. Code clones emerge from exercises, such as copy-paste and adaptation. Identification and evaluation of software clones is vital due to its fundamental part in different development tasks: software maintenance, code quality analysis, plagiarism detection etc. and so forth [11]. It requires additional effort to upgrade a framework containing copied code parts. Research has demonstrated that 7– 23% of programming frameworks contain cloned code [9].

Numerous clone detection techniques have been proposed in the literature. Among various techniques in literature, Text-based techniques are lightweight and detect clones with a higher recall, but text based techniques are unable to detect suitable syntactic units. Token-based techniques are fast with high recall but possess low precision values. Moreover, parsing is required to compare ASTs of two code fragments, which will be more time consuming and will involve more effort. Parser-based techniques are capable enough to detect syntactic clones. However, these techniques give low recall values. Metric-based techniques detect syntactic as well as semantic clones with high precision values and also fast enough in processing. However, metric based techniques sometimes fail to detect some of the actual clones [11]. The other clone detection techniques i.e. Program Dependency Graph(where PDG is a directed graph which represents the dependencies among program elements in a program) based are more suitable to detect semantic clones. But sub-graph comparison in PDG techniques is very costly [12]. Moreover, PDG-based techniques takes longer time to detect code clones, because in these techniques source files are transformed into PDGs and the fraction of time is also devoted to comparing those PGDs [2]. The tool NICAD 3.4 identifies clones at function/ block level, but it possesses language-sensitive parsing which makes the detection process slow **[13]**. The tools CCF inder and NICAD are successful to detect the lexical clones (e.g., Type-1 and Type-2) and but are ineffective for detecting Type-3 clone in most of the cases [5]. AST (Abstract Syntax Tree) has been introduced in Deckard [36] tool and it measures the structure level similarity between two code fragments. Deckarduses structure information of software but it ignores lexical information of the source code, which might be helpful to detect Type-3 clones. There are researches which detect clones by comparing files. This detection of clones at coarse grain level makes the searching process faster, but it may miss fine grained level clones**[14]**. Software systems included in FreeBSD Ports Collection had been run on file level clone detection tool, FC Finder, which is developed by Sasaki et al. **[15].** But file level cloning is not suitable to detect function/method level clones if their dependencies with other files are not wholly duplicated. Kamiya et al.[16] has proposed a clone detection code clone finder (CC finder) tool, which is based on token based approach. In this tool, there is limit to set minimum size of tokens which is 25. This limit is variable. It can create biasing issues to detect clones with variation in number of tokens. Baxter et al. [17] implemented a Clone DR tool based on AST (annotated parse tree)-techniques. Compiler generator generates AST of source code and then sub-trees are compared on the metrics computed by hash function. Sub-trees having similar source code are returned as clones. The AST generation slows down the clone detection process. Jean Mayr and et al.[35] implemented a method to detect clones using metric-based approach. This has been accomplished using a DATRIX tool framework by converting source code into some intermediate form. Patenaude et al. [18] used metrics based on method-level approach to extend the functionality of DATRIX tool to identify Java-clones. Clones are identified using some unusual characteristics of Java code fragments to find specific sections of a system that will require special attention. Roy et al. [37] recently developed Clone Works tool to detect large scale near miss clones. Clone Works detects clones with modified Jaccard

Similarity. Clone Works gives user a full control over the processing to detect clones. This tool manages the issue of scalability and millions of lines of code at once. A pair-of-code-fragments is reported as clone-pair if similarity threshold is above 70%. Because Clone Works computes similarity on threshold basis, the reported results by this tool can possess biasness by varying threshold limit. Ragkhit wetsagul et al. [38] followed the approach of compilation/de-compilation to detect code clones in Java. It manages syntactic changes in the code and can be used as normalization. NICAD has been used to detect clones before de-compilation and after de-compilation in open source softwares, which helps to identify type-3 clones. Strüber et al. [39]proposed a method to detect clones across graph-based model transformation languages. Potential use cases have been introduced in the context of constructive and analytical quality assurance. Customization of Con Qat has been done to satisfy all key requirements. Kodhai et al. [3] proposed an intermediate template to detect method-level clones combining textual and metric based approach. But single encoding scheme adopted for all data types leads to false positives. Further in literature, in-depth empirical study of cloning in social programming platform such as GitHub, has been performed by Gharehyazie et al. [40]. The author used Deckard tool, to identify replicas of code fragments in GitHub. Another tool CLONE-HUNTRESS has been developed, it tracks changes to clones over time. Mondal et al. [41] used two tools [NiCad and CC Finder X to investigate different case studies with the motive to investigate the stability of type-1, type-2 and type-3 clones. The author used 8 stability-assessment-metrics and concluded that cloned code is more unstable, because it is more prone to changes and errors. Type-3 clones are highly unstable, because type-3 clones covers main functionality of the software and easily adapted for reuse. The drawbacks of existing technique/tools and methods, provide a path to investigate a novel, hybrid or modified approach/template to identify clones. Junaid Akram [44] et al. Implemented s index-based features extraction technique (IBFET) to compare subject system against a large data set of source code to detect code clones at file granularity level. The approach followed is good enough to handle with big datasets of source code, but in IBFET approach, there is overhead and extra time is consumed to create index. Chunrong Fang [45] et al. Proposed a novel joint code representation for function level clone detection that applies fusion embedding techniques to learn hidden syntactic and semantic features of source codes. Caller-callee relationships as a functionality has been used to identify relationship between different functions and then a supervised deep learning model to detect functional code clones has been implemented. Kluban [46] et al. dealt with package-level vulnerability tracking and measurements. A vulnerability detection framework has been developed that uses vulnerable pattern recognition and textual similarity methods to detect vulnerable functions in real-world projects. The main work done is concentrated on JavaScript security issues.

3. RESEARCH METHODOLOGY

In this research, a generalized template for Java programs has been proposed. This converts original source code into a common generalized intermediate pattern (it is suitable to all function bodies) and will serve as a uniform base for the programming constructs. Java follows function-based and classes-based programming development. Moreover, functions lead to code reusability and reduce the project code size. Therefore, to focus on ‘function’ granularity, none other than Java is suitable for this purpose.

3.1 Input/ Data Concepts

An Open Source Software Jhot Draw [19] has been considered for experimentation for the following reasons:

1. Jhot Draw is an opensource software and it is widely used by many researchers [3, 4, 5, 6, and 7].
2. The key idea is to extract clones from Java-based source code, and Jhot Draw is purely a Java-based platform.
3. Jhot Draw is totally a function/method based development; therefore, this software is suitable for chosen ‘function-level’ granularity.

One of the example of function clones present in Jhot Draw is represented in table 2 below.

Table 2: Examples of Function Clones in JhotDraw5.2

Function-Fragment1 (FF_1):Bring To Front Command_3.java
public Boolean is Executable ( )
{
return f View . selection Count ( ) > 0 ;
}
 	
Function-Fragment2 (FF_2):UngroupCommand_3.java
public Boolean is Executable ( )
{
return f View . selection Count ( ) > 0 ;
}
	Function-Fragment3 (FF_3):Trangle Figure_8.java
	public Boolean contains Po A (A x , A y) {
	return polygon ( ) . contains (x , y) ; }
	Function-Fragment4 (FF_4):AbstractConnector_8.java
	public Boolean contains Po A (A x , A y) {
	return owner ( ) . contains Po A (x , y) ;
	}
		Function-Fragment5 (FF_5): AbstractConnector_2.java
		protected Drawing create Drawing() {
		return new Standard Drawing ();
		}
		
		Function-Fragment6 (FF_6): ArrowTip_6.java
		protected Standard Drawing View create Drawing View() {
		return new Standard Drawing View (this 410 370);}

Clone Pair (FF_1, FF_2): Type-1 Clone Clone Pair (FF_3, FF_4): Type-2 Clone Clone-Pair (FF_5, FF_6): Type-3 Clone

3.2 Overall Steps of Methodology

All steps followed by proposed methodology have been described in Figure 1. The steps of methodology are:

1. **Input Source Code Files:** Java source code files are selected as input.

2. **Pre-processing:** Pre-processing of input source code is performed by removing comments, imported packages and extra white spaces.

3. **Function Identification:** All function bodies are extracted from the pre-processed source code. As an example all function bodies extracted from AbstractTool.java are shown in figure 1.

Table 3: Nomenclature for Keywords and Reserve Words

Reserve	Decoding	Reserve	Decoding	Reserve Word	Decoding	Reserve	Decoding

Word		Word				Word	
abstract	AT	**boolean**	BO	**assert**	AS	**break**	BR
case	CS	**char**	CR	**catch**	CH	**class**	CL
continue	CN	**do**	DO	**default**	DT	**double**	do
enum	EM	**final**	FI	**extends**	EX	**finally**	FY
for	FR	**if**	IF	**Goto**	GT	**implements**	IL
instance of	IO	**interface**	IR	**int**	IN	**long**	LG
new	NE	**private**	PT	**package**	PG	**protected**	Pr
return	RE	**static**	ST	**short**	SH	**strictfp**	SF
switch	ST	**this**	TH	**synchronized**	SC	**throw**	TH
transient	TT	**void**	VD	**try**	TY	**volatile**	VL
byte	BT	**float**	FT	**public**	PU	**while**	WL
Const	CT	**import**	IM	**super**	SP	**native**	NV
else	EL	**throws**	Th	String	St	**object**	Ob
IO Exception	IE						

4. **Removal of Blank Functions:** Blank functions are removed to filter the whole database. Deleting Blank-function bodies decreases the overburden of executing unnecessary Lines of Code (LOC).
   Encoding Scheme: All keywords and reserved words are encoded using encoding scheme presented in table 3.

5. **Uniform Intermediate Representation of Source Code (Proposed Template):** Pre- processed source code is converted into a new template. In addition to pre-processing and formatting of source code, a generalized template conversion is used here. This converts original source code into a common generalized pattern and will serve as a uniform base for the programming constructs between the clone pairs of the same type. Figure 1 shows AbsractTool.java after pre-processing, by deleting comments, header files etc. The file contains some blank function bodies: mouse Drag (), mouse Up (), mouse Move (), key Down (). These blank function bodies are filtered for further processing of the file. It further reduces processing database size as well as processing time. Rest of the functions: public void activate(),public void deactivate(), public void mouse Down(),public void mouse Drag(),public void mouse Up(),public void mouse Move(),public Drawing drawing(),public Drawing Editor editor() are extracted for clone detection. The encoding scheme is applied further to all keywords and reserve words. In the end, the generated template is reduced to a single sequence by removing spaces for less memory consumption and fast comparison. Another reason for choosing single sequence representation is that sometime number of spaces may vary in the same intermediate templates and it can lead to dissimilar signatures for same function bodies.
6. The approach followed stores two parameters of the extracted function as a record for future reference: function name and file name to which that function belongs. After executing all processing steps, Proposed Template is generated as depicted in Figure 2.

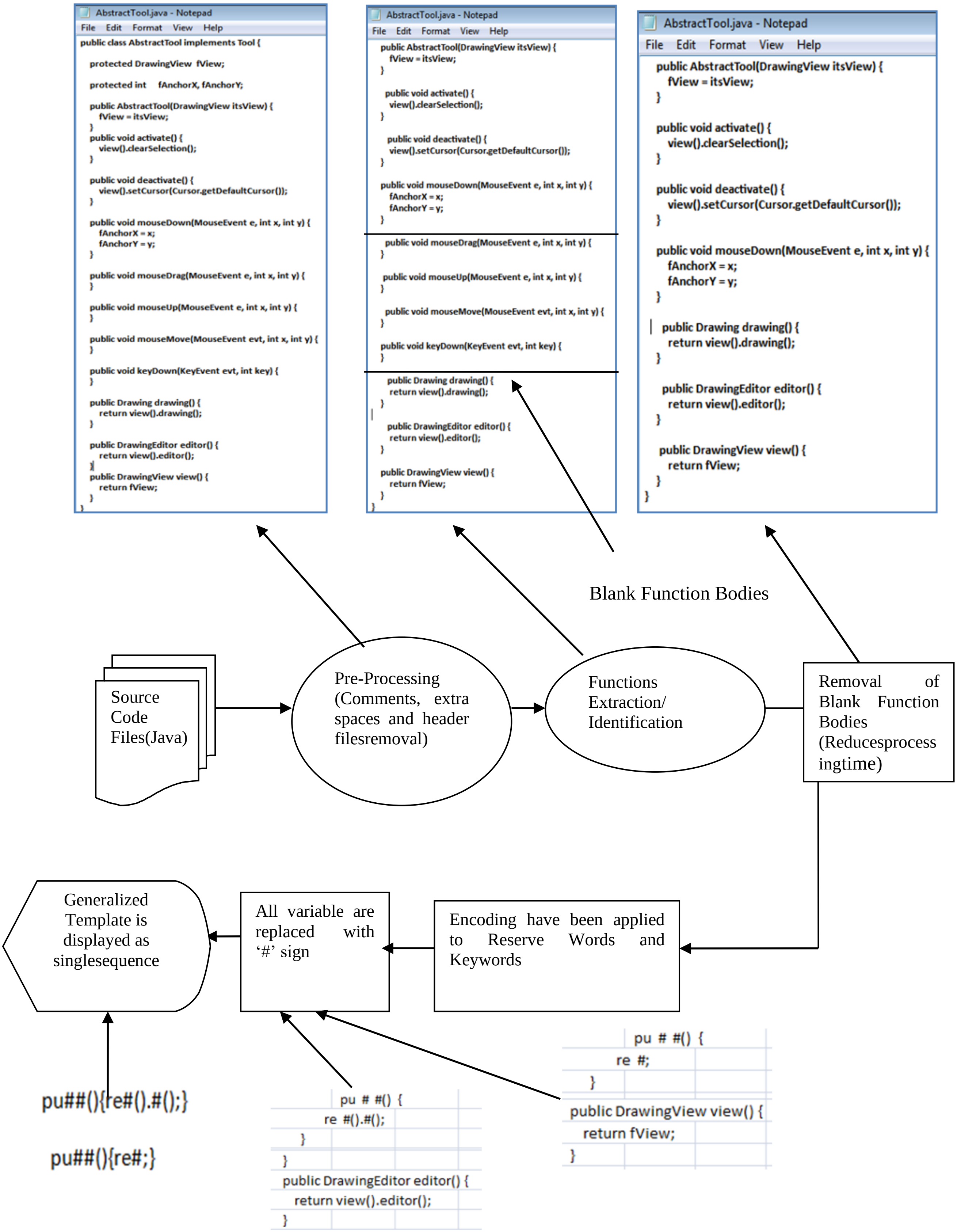

Figure 1: Schematic Diagram of Proposed Methodology

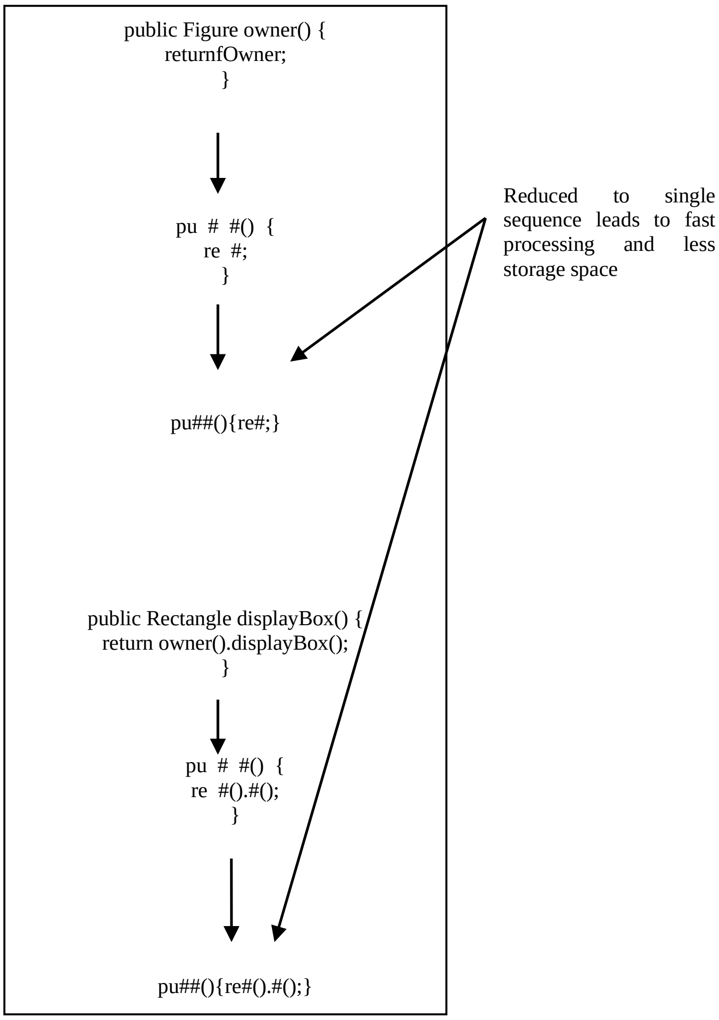

Figure 2: Proposed Template

3.2.1 Type-1 and Type-2 Identification using Proposed Template

For Type1, as per definition of code clones, two fragments should be 100% similar. Therefore, if generated template of two function-bodies is 100% similar, it will form a Clone Pair of Type1.For Type2 clones, two functions will form a Clone Pair if changes are evolved over function identifiers, variable names, types etc. Therefore, a uniform intermediate pattern is proposed in which editing differences are minimized. Table 4 illustrates Type1 and Type2 clones. Clone Pair (CF_x, CF_y) represent Exact (Type1) clones. Clone-pair (CF_m, CF_n) represents Type2 clones, in which display Box() has been renamed to f poly.

3.2.2 Type-3 Identification using Proposed Template

In Type3 clones, both code fragments can share same variable names, can exhibit parameterized clones and also two fragments can vary in LOC(lines of code) with insertions, deletions or modification of code. The proposed method aims to discover a maximum pairing of matched functions, whether in the same class or in different classes. Table 5 illustrates Type-3 clones. Clone pair (CF_1,CF_2) depicts Type-3 clone: Variable Drawing has been renamed as Drawing View and also public void invoke Start() has been modified by a mending invoke Start($, $, $, drawing()). In clone pair (CF_3, CF_4) Type-3 clones: Standard Drawing View()has been modified by adding (this 410 370) to it.

Table 4: Template for Type1 and Type2 Clones

Fragment No.	Code Fragment	Template	Signature
CF_x	public boolean is Executable() { return f View .selection Count()>0; }	pubo #() { re #.#()>#; }	Same Signature Generated
CF_y	public boolean is Executable () { return fView. Selection Count()>0;}	pubo#() { re.#.#()>#;}	
CF_m	public Boolean contains Point(int x, int y) { return display Box().contains(x, y); }	pubo #(#) { re #().#(#, #); }	A small difference in Signature Generated (with only one extra ( ) in CF_m)
CF_n	public boolean contains Point(int x, int y) { return f Poly. contains(x, y); }	pubo #(#) { re #.#(#, #); }	

Table 5: Template for Type-3 Clones

Fragment No	Code Fragment	Template	Signature
CF_1	public void invoke Start (int # , int # , Drawing #) { }	Puvo #(#,#,#) { }//line1	A large difference in Signature (Line1 and line2 are same. Line3 i.e. # (#,#,#.( )); has been inserted in CF_2) (th #,#has been inserted in CF_4.
CF_2	public void invoke Start (int # , int # , Drawing View #) { **invoke Start (# , # , # . drawing ( )) ;** }	Puvo #(#,#,#) //line2 { # (#,#,#.( ));}//line3	
CF_3	protected Drawing create Drawing() { return new Standard Drawing();}	Pr # ( ) { re ne # ( ); }	
CF_4	protected Standard Drawing View create Drawing View () { return new Standard Drawing View(this 410, 370); }	Pr # ( ) { re ne # (th #,#); }	

4. RESULTS AND DISCUSSION

The proposed approach has been demonstrated on different versions of Jhot Draw and has also been compared with different proposed templates in literature.

4.1 Fundamental comparison of various tools/techniques with the proposed method

Code Clone detection is a continuous process in software engineering. The motive of proposed work is to identify function-level-clones in source code with high speed and less memory consumption. No technique can be considered better until its quality or performance can be measured. Fundamental comparison of various tools/techniques with the proposed technique is shown in table 6, which aims to detect code clones in the source code. Clone detection tool PMD (http://www.PMD.sourceforge.net/) scans Java source code and detects duplicate code. PMD operates on the basis of threshold metric and allows setting the number of tokens to identify duplicated code. But the threshold here can be biased, because detected clones can vary according to threshold variation i.e. number of tokens [23]. Bauhaus (http://www.bauhaus-stuttgart.de/) provides support to identify reusable components of software, and estimation of change impact. The Bauhaus also discover same code blocks for: portions of identical code (Type-I), their variation with different variable names and identifiers (Type-II), and portions of similar code with added or removed statements (Type-III). In this research, current proposed template also detects these three types of clones but with the variation in granularity i.e. functions. Google Code Pro Analytix (https://developers.google.com/java-dev-tools/codepro/doc/) is used by Eclipse developers as a Java testing tool for improving software quality. It solves the purpose of code analysis, various metrics computations and similar code analysis. The tool offers various facilities: (1) to search the code that can possibly bere factored, (2) to identify code containing remaining errors and (3) To identify similar portion of code. Current method is also capable to detect function level clones useful for refactoring purposes which is comparable to Code Pro [23]. Current proposed template does not depend upon threshold and compares whole function-body at a time.The work is in comparison with other clone detection tools and possesses an advantage of adopting threshold-free-approach which makes it free from any biasing and secondly function-granularity serves a basis to refactoring [23]. Shortest encoding scheme ever adopted in literature have been implemented here and it makes the approach capable of consuming less memory.

Table 6: Fundamental Comparison with Recent and State-of-the-Art Methods

Tool/Approach	Intermediate Representation	Supported Languages	Method	Granularity	Types of Clones Detection
NICAD [31]	generic normalization template	Java C	text-based and abstract syntax tree-based	Functions and Blocks [31]	I, II, and III [33][34]
Kodhai et al. /Clone Manager[3]	White spaces, comments removed Data types and variable names are encoded	Java C	combination of textual comparison and metrics computation	Function Body	I, II, III and IV

PMD[23]	Tokens	C, C++, Java, Jsp, Fortran, Php, Ruby	string matches using Rabin-Karp [32]	String (25 tokens composed 4-6 LOC)	I and II
Bauhaus[23]	-----------	C, C++, C#, Java, Ada	Baxter et al.'s changes on AST [17]	-------	I, II, and III
Google Code Pro Analytix	------------	Java	Undocumented relies on Java AST	-----	I, II, and III
CC Finder[16]	Tokens	COBOL, Java, C and C++	Suffix-tree based on token matching	Token	I and II
Clone Dr	Syntax Tree	COBOL, Java, C and C++	Abstract Syntax Tree (AST) Method		I, II, and III [33]
Hamid Abdul et al. [20]	Tokens	Java	Numeric Encoding Scheme (numeric values consumes more space)	Token	-------
CK-Roy [21]	TXL Taxonomy	Java C	parsing based template (consumes extra time for parsing)	Code Fragment	Used for automating Precision and Recall
A R, Haga H. Et al. [22]	Tokens		variables/identifiers with '$'(keywords and reserve words are not encoded)		I, II and III
Hotta et al. [29]	extract blocks using JDT by Parsing source code		Parse source code to extract blocks using JDT		I and II
Marcus & Maletic [26]	comment removal and token regularization	Mosaic 2.7/ C	vector representation using LSI	code segments, files	high-level concept clones

Basit & Jarzabek [27]	Tokens	Eclipse Graphical Editing Framework/Eclipse Visual Editor/ Java, Open J Graph/ Java J2ME Wireless Toolkit 2.2/Java Pet Store 1.3.2/ Java	frequent item-set mining	files, methods	high-level concept clones
Grant &Cordy [28]	vector space representation	Linux/ C	independent component analysis	methods, blocks	Similar code fragments
Rattan, Bhatia, and Singh [25]	vector space representation	Dns java/ Java	principal component analysis and cosine similarity	class diagram, class file, and methods	Model Clones
[44]	indexing	Hadoop and Map Reduce	Divide and conquer technique to find similar features between different components	File level	Structural Clones
[45]	Fusion embedding technique	Deep Neural Network	Caller-Callee functionality to know relationship among functions	Function/Method	
Proposed Work	Header files removal, comment removal, token regularization and function/metho	Rabin-Karp Algorithm/Java	Intermediate Template using Character Encoding Scheme	Function/Method	I, II and III

	d detection				

4.2 Comparison with some of the proposed templates in Literature

In literature, various templates have been proposed for code clone detection [20, 3, 21, 16], which are closely comparable to our proposed template. Figure 3 shows various templates from literature proposed by different researchers. Here, proposed work has been compared with Hamid Abdul et al. [20], CK-Roy [21], Kodhai et al. [3] and A R, Haga H. Et al. [22].Hamid Abdul et al. [20] adopted numeric encoding scheme for all keywords and reserve words, hence consumes more memory space. In spite of this, character encoding should be adopted to reduce memory storage. CK ROY [21] proposed a parsing based template, it needs parsing phase to represent input as a parse tree for internal representation, which is a tedious task and takes more time for execution, so it further makes the process heavier. Kodhai Et al. [3] have proposed an intermediate template for clone detection, in which, whether the data type is int, char or float, all are decoded with 'DAT', therefore it can rather create more false positives (in terms of Type1 clones) when explored manually as depicted in Figure 4. For example, In Figure 4, code fragments CF_a and CF_b when transformed in internal format, both lead to the same intermediate template ($Template_a$ and $Template_b$ are equivalent). It is regarded as Type1 clones. But when explored manually, it shows Type2 clones. Thus, it leads to false positive. Ami R, Haga H. [22] replaced all variables/identifiers with '$' sign. In this method encoding scheme for data types 'int', reserve words: return, if, else, public etc has not been adopted. Therefore, the proposed template consumes more space. Rather, the operation can be optimized using some short encodings for the same to reduce execution time and space. In our proposed work, generated template has been converted into single sequence which makes the process lighter and reduces processing time. Hence, the proposed method shows improved time and space complexity as compared to already existing methods.

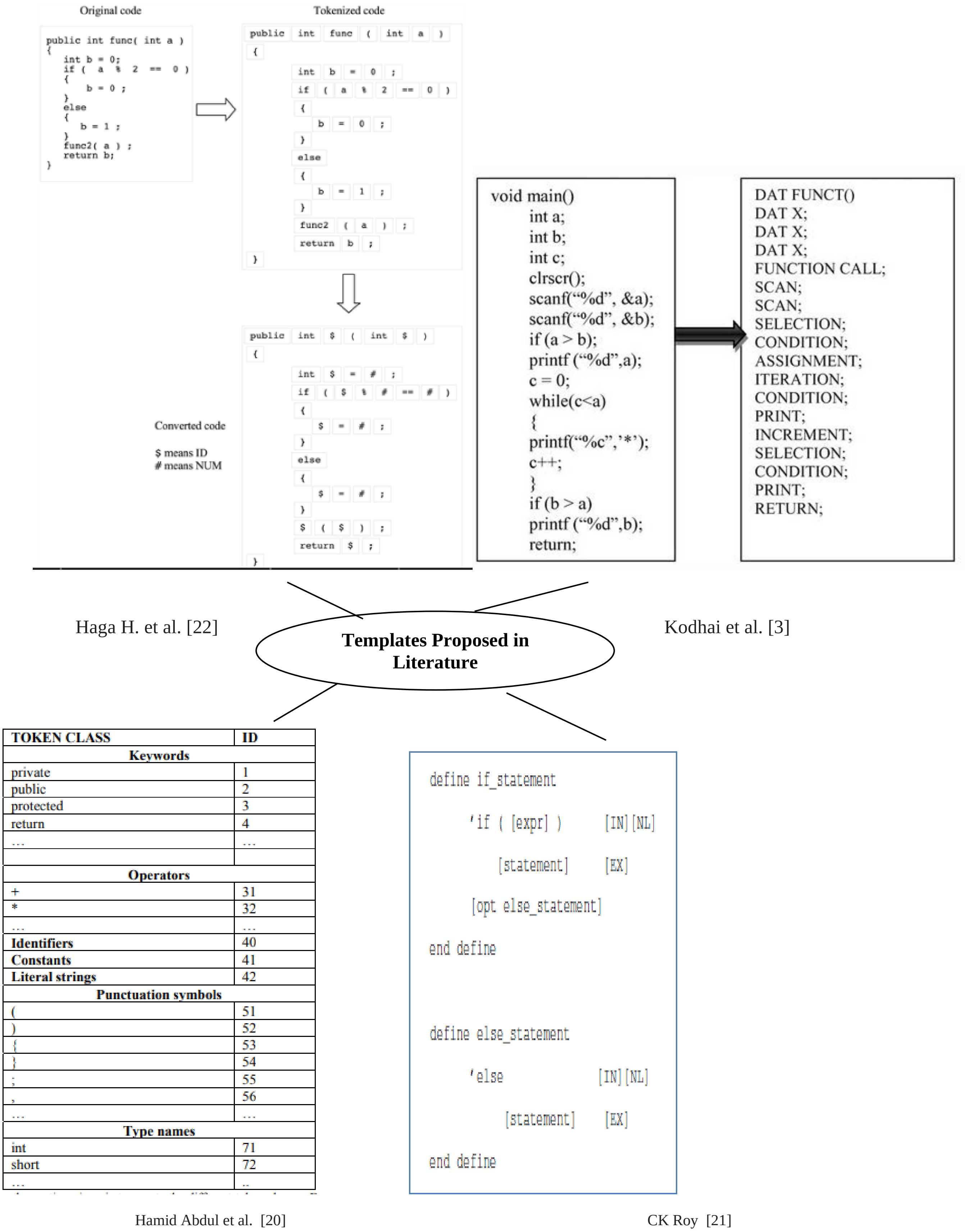

Figure 3: Already Proposed Templates in Literature

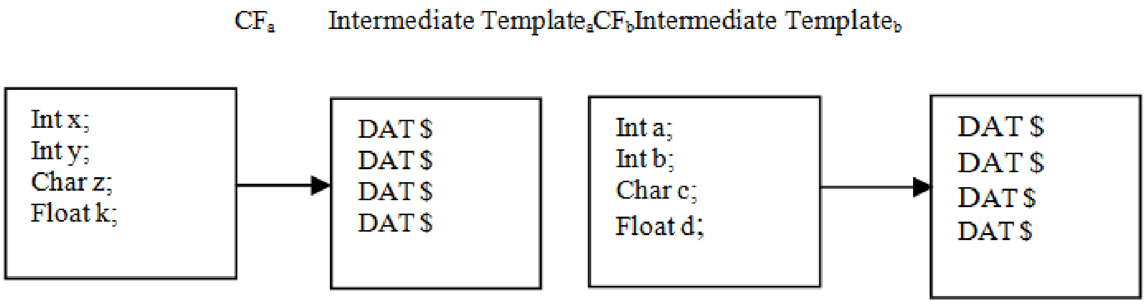

Figure 4: a) Code Fragment1 (CF_a) b) Code Fragment2 (CF_b)

4.3 Comparison on the basis of Performance Parameters

Some other parameters have also been computed to analyze the performance of proposed template such as: Precision, Recall, and False-Positives, with varying percentage of detected functions. Results have been shown at various performance levels: 2%, 4%, 6% and 10 % as illustrated in Figure 5a and 5b.

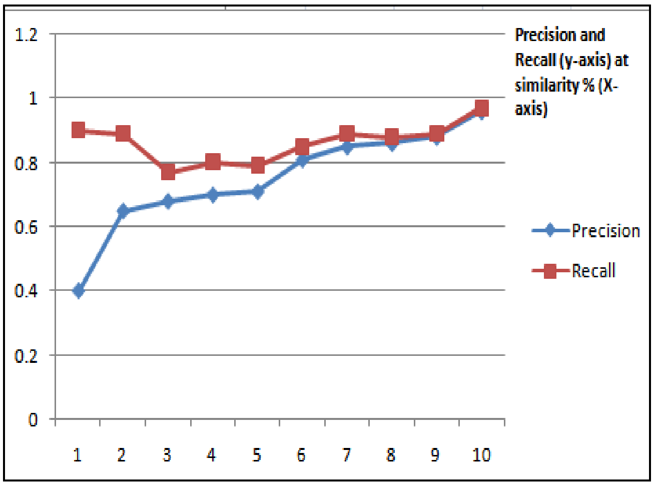

Figure 5a: The plot shows precision and recall values at various similarity percentage of detected functions.

Precision and Recall have been calculated from the candidate code fragments whose fingerprints are involved in a collision template and in-depth analysis has been performed manually. The precision and recall rate reported by experimentation are quite promising.

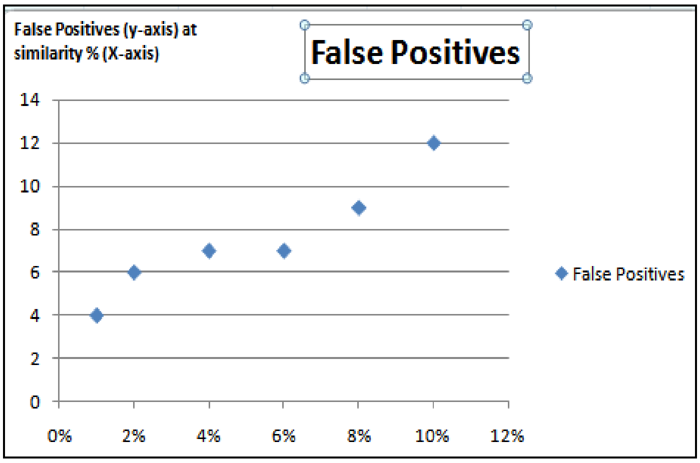

Figure 5b: The plot illustrates the number of false-positives detected at varying percentage similarity of detected functions

Figure 6 shows number of clone pairs at varying percentage levels of detected functions: 2%, 4%, 6% and 10 %.Maximum clone pairs are detected to be of type2 clones as depicted in Figure 8. Execution time of proposed method has also been measured. Fuyao et al. [24] demonstrated modified Rabin-Karp using bitwise operations. The author has reported results in terms of the varying size of alphabets; likewise our results have also been reported in terms of varying percentage of detected functions in software as depicted by table 7.

Table 7: Results with varying percentage of functions

	Type-1(2, 4,6,10 %)				Type-2(2, 4,6,10 %)				Type-3(2, 4,6,10 %)			
Clone size(LOC)	2-3				5-6				6-18			
Clone pairs	2%	4%	6%	10%	2%	4%	6%	10%	2%	4%	6%	10%
	-	3	7	9	1	7	8	11	2	3	5	8

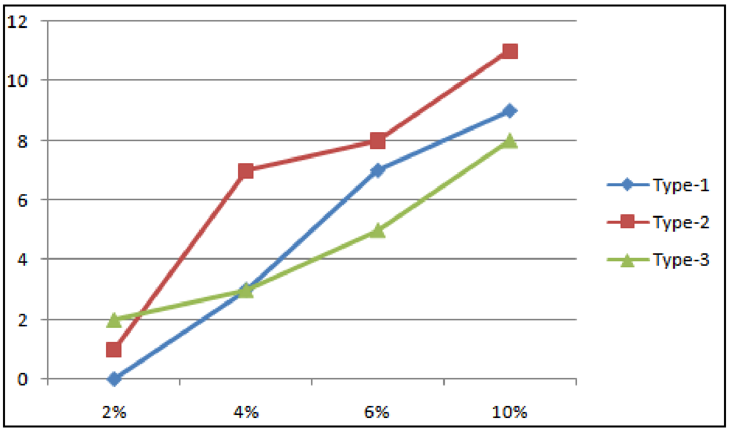

Figure 6: clone pairs on various levels 2%, 4%, 6% and 10 %

Regression analysis has also been shown between Function-Percentage from total software w.r.t Precision. Regression curve in Figure 7 describes the scenario of the rise in precision as the rise in the percentage of functions, the results here show accuracy at 95 % confidence level, it means deviations in accuracy occurs in only 5% cases, hence the results are very much promising.

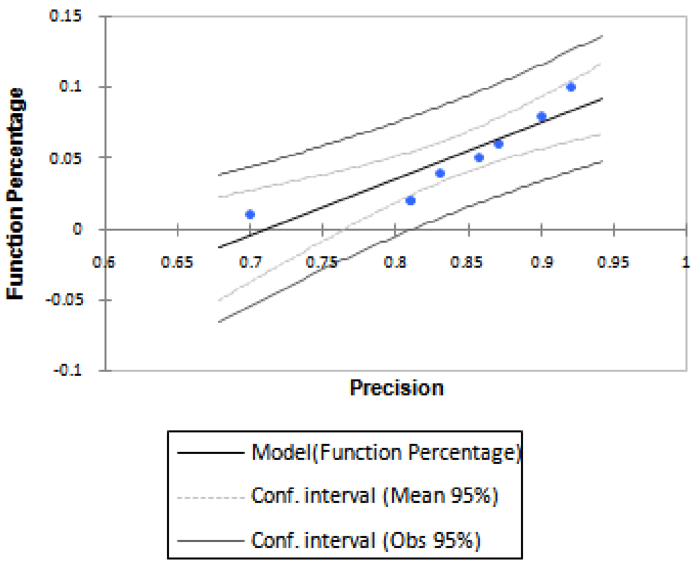

Figure 7: Regression Curve

5. THREATS TO VALIDITY

This current work exhibits some of the following threats to validity.

- First, an internal threat to validity is that the functions reported similar to each may not be clones according to other developers, because different people would have different perception/understanding of the same lines of code.
- Second, work has not been proceeded to detect semantic clones among functions. It is hard to avoid subjectiveness while evaluating open source softwares because of lack of template benchmark.
- Comparison of recall has not been explored, because to check recall in this higher number of functions/methods is practically not possible by considering only our own perception. It would need the time of developer and his acknowledgement to inspect the sampled functions.
- Also, the underlying design and structure of the clone detection tool has a great impact on the clone detection results. Variation in minimum token length by some tools leads to different results. We evaluated various parameters based on a prior study and our demonstration with existing clone detection tools.

6. CONCLUSIONS

1. An efficient and faster approach has been proposed to detect function-level clones based on tokenized method.
2. A novel template has been proposed which is faster, because it is free from parsing issues. Moreover, character encoding has been adopted which makes the execution thread lighter and faster.
3. The work shows shortest encoding scheme ever adopted as compared to literature which shows novelty.

4. The proposed scheme also exhibit less space complexity because of single sequential representation and adopted character encoding scheme (character consumes less space as compared to other data types).
5. Proposed work is capable to detect exact as well as gapped function-level clones.
6. Detection of false positives has also been verified, which shows improved precision.
7. Function-Level clone detection has been performed, which his beneficial to detect reusability in the software and would serve as a basis for refactoring.

REFERENCES

1. Davey N, Barson P, Field S, Frank R, Tansley D. , "The development of a software clone detector", International Journal of Applied Software Technology, pp. 219-236, 1995.
2. Murakami H, Hotta K, Higo Y, Igaki H, Kusumoto S., "Gapped code clone detection with lightweight source code analysis", IEEE 21st International Conference on Program Comprehension (ICPC), pp. 93-102, 2013.
3. Kodhai E, Kanmani S., "Method-level code clone detection through LWH (Light Weight Hybrid) approach", Journal of Software Engineering Research and Development, Vol. 2, no.1, 2014.
4. Ó Cinnéide M, Tratt L, Harman M, Counsell S, Hemati Moghadam I., "Experimental assessment of software metrics using automated refactoring", Proceedings of the ACM-IEEE international symposium on Empirical Software Engineering and Measurement, pp. 49-58, 2012.
5. Chen X, Wang AY, Tempero E., "A replication and reproduction of code clone detection studies", Proceedings of the Thirty-Seventh Australasian Computer Science Conference, pp. 105-114, 2014
6. Mubarak-Ali AF, Syed-Mohamad SM, Sulaiman S., "Enhancing Generic Pipeline Model for Code Clone Detection using Divide and Conquer Approach", Int. Arab J. Inf. Technol., Vol. 12, no. 5, pp 510-517, 2015.
7. White, Martin, et al. "Deep learning code fragments for code clone detection." Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ACM, pp. 87-98, Aug 25, 2018.
8. Li, Z., Sun, J., "An iterative, metric space-based software clone detection approach", 2nd International Conference on Software Engineering and Data Mining (SEDM), pp. 111–116, 2010.
9. Abd-El-Hafiz, S.K., " A metrics-based data mining approach for software clone detection", IEEE 36th Annals Computer Software and Applications Conference (COMPSAC), pp. 35–41, 2012.
10. Saha, R.K., Roy, C.K., Schneider, K.A., Perry, D.E., "Understanding the evolution of type-3 clones: an exploratory study", Proceedings of the 10th Working Conference on Mining Software Repositories, pp. 139–148. IEEE Press, Piscataway, 2013.
11. Bellon S, Koschke R, Antoniol G, Krinke J, Merlo E., "Comparison and evaluation of clone detection tools", IEEE Transactions on software engineering, Sep; ol. 33, no. 9, 2009.
12. Koschke R, Falke R, Frenzel P., "Clone detection using abstract syntax suffix trees", Reverse Engineering, WCRE'06, 13th Working Conference, pp. 253-262, 2006.
13. Cordy Nicad 3.4 The NiCad Clone Detector, 2011
14. Ishihara, T., Hotta, K., Higo, Y., Igaki, H. and Kusumoto, S., "Inter-project functional clone detection toward building libraries-an empirical study on 13,000 projects", IEEE conference on Reverse Engineering (WCRE, pp. 387-391, 2012.

15. Y. Sasaki, T. Yamamoto, Y. Hayase, and K. Inoue, "Finding File Clones in FreeBSD Ports Collection," in Proc. of the 7th Working Conference on Mining Software Repositories, pp. 102–105, 2010.
16. Kamiya, Toshihiro, Shinji Kusumoto, and Katsuro Inoue., "CC Finder: a multilinguistic token-based code clone detection system for large scale source code", IEEE Transactions on Software Engineering, Vol. 28, no. 7, pp. 654-670, 2002.
17. I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in Proceedings of the IEEE International Conference on Software Maintenance (ICSM '98), pp. 368–377, 1998.
18. Patenaude, J.F., Merlo, E., Dagenais, M. and Laguë, B., "Extending software quality assessment techniques to java systems", IEEE Seventh International Workshop on Program Comprehension, pp. 49-56, 1999.
19. The J Hot Draw: http://www.jhotdraw.org/ (June 2006).
20. Basit, Hamid Abdul, and Stan Jarzabek., "Efficient token based clone detection with flexible tokenization." ACM Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT Symposium on The foundations of software engineering, pp: 513-516, 2007.
21. Roy CK., "Detection and Analysis of near-miss software clones", IEEE International Conference on Software Maintenance, ICSM, pp. 447-450, 2009.
22. Ami R, Haga H., "Code Clone Detection Method Based on the Combination of Tree-Based and Token-Based Methods", Journal of Software Engineering and Applications, 10:13, p.891, 2017.
23. Research Article "Software Clone Detection and Refactoring" Hindawi Publishing Corporation, ISRN Software Engineering, Article ID 129437, 8 pages, 2013.
24. Zhao F, Liu Q., "A string matching algorithm based on efficient hash function", IEEE International Conference on Information Engineering and Computer Science, pp. 1-4, 2009.
25. Dhavleesh Rattan, Rajesh Bhatia, Maninider S., "Detecting High-Level Similarities in Source Code and Beyond", International Journal of Energy, Information, and Communications, Vol. 6, no. 2, pp. 1-16, 2015.
26. Marcus and J. I. Maletic, "Identification of high-level concept clones in source code", Proceedings of 16th IEEE International Conference on Automated Software Engineering (ASE'01), pp. 107-114, 2001.
27. H. A. Basit and S. Jarzabek, "A Data mining approach for detecting higher-level clones in software", IEEE Transactions on Software, vol. 35, no. 4, pp. 497-514, 2009.
28. S. Grant and J. R. Cordy, "Vector Space Analysis of Software Clones", Proceedings of 17th IEEE International Conference on Program Comprehension, pp. 233- 237, 2009
29. Hotta, Keisuke, et al. How Accurate Is Coarse-grained Clone Detection?: Comparision with Fine-grained Detectors. Electronic Communications of the EASST, 2014.
30. Roy CK, Cordy JR., "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization", The 16th IEEE International Conference on Program Comprehension, ICPC, pp. 172-181, 2008.
31. Cordy, J.R. and Roy, C.K., "The NiCad clone detector" ,IEEE 19th International Conference on Program Comprehension (ICPC), pp. 219-220, 2011.

32. R. M. Karp and M. O. Rabin, “Efficient randomized pattern matching algorithms,” IBM Journal of Research and Development, vol. 31, no. 2, pp. 249–260, 1985.
33. Kaur, H. and Maini, R., Performance Evaluation and Comparative Analysis of Code-Clone-Detection Techniques and Tools, IJAST (SERSC), 2017.
34. Sheneamer, A. and Kalita, J. A survey of software clone detection techniques. International Journal of Computer Applications, 137(10), pp.1-21, 2016.
35. Mayrand J, Leblanc C, Merlo E., “Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics”, In International Conference on Software Maintenance(icsm) 1996 Nov 4 (Vol. 96, p. 244).
36. Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. DECKARD: scal- ´ able and accurate tree-based detection of code clones. In Proceedings of the 29th International Conference on Software Engineering, pages 96–105, Minneapolis, MN, USA, 2007.
37. Roy et al. Clone Works
38. Svajlenko J, Roy CK. Fast and flexible large-scale clone detection with Clone Works. In Software Engineering Companion (ICSE-C), 2017 IEEE/ACM 39th International Conference on 2017 May 20 (pp. 27-30). IEEE.
39. Strüber D, Plöger J, Acreţoaie V. Clone detection for graph-based model transformation languages. In International Conference on Theory and Practice of Model Transformations 2016 Jul 4 (pp. 191-206). Springer, Cham.
40. Gharehyazie M, Ray B, Keshani M, Zavosht MS, Heydarnoori A, Filkov V. Cross-project code clones in GitHub. Empirical Software Engineering. 2018:1-36.
41. Mondal M, Rahman MS, Roy CK, Schneider KA. Is cloned code really stable?. Empirical Software Engineering. 2018 Apr 1;23(2):693-770.
42. Roy, C.K. and Cordy, J.R., 2018, March. Benchmarks for software clone detection: A ten-year retrospective. In 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER) (pp. 26-37). IEEE.
43. White, M., Tufano, M., Vendome, C. and Poshyvanyk, D., 2016, August. Deep learning code fragments for code clone detection. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (pp. 87-98). ACM
44. Akram, J., Mumtaz, M. and Luo, P., 2020. IBFET: Index-based features extraction technique for scalable code clone detection at file level granularity. Software: Practice and Experience, 50(1), pp.22-46.
45. Fang, C., Liu, Z., Shi, Y., Huang, J. and Shi, Q., 2020, July. Functional code clone detection with syntax and semantics fusion learning. In Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 516-527).
46. Kluban, M., Mannan, M. and Youssef, A., 2022. On Measuring Vulnerable JavaScript Functions in the Wild.