THE PREDICTION OF CODE CLONE QUALITY BY EXTRACTING FAST AND PRECISE CLONE GENEALOGIES IN COMPLEX CODE BASES

Dissertation submitted in fulfilment of the requirements for the Degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

By

GEETIKA

11501877

Supervisor

SANDEEP KAUR

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

Month-April Year- 2017

ABSTRACT

Programming cloning is the present issue in ventures, making an affirmation of clones a key piece of programming examination. Existing written work on the theme of programming or software clones is assembled completely into different characterizations. Use of existing code either by duplication and paste strategies or by performing minor modifications in the present code is known as programming cloning. Programming clones may provoke the bug inducing and genuine support issues.

Duplication is distinguished by taking a gander at highlight film of source parts. The briny occupation for the spotting is that source code is every so often imitated precisely. The area methodology must have the ability to ignore the shallow remaining segment and to dressed metal on the essential law of likeness remembering the true objective to find material duplication. While higher layer information yielded by syntactic and semantic code examination can be put another option to convincing use, the detriments of these trench examination methods are over all the reduced adaptability to different programming related process. Since duplication is an inescapable issue, in any case, support for duplication area and organization is required for each programming language being utilized.

Clone types/sorts, techniques for clones and assorted procedures are joined into this paper. In like manner this paper will fill as a manual for a potential client of clone recognizing evidence philosophies, to help them in picking the benefit devices or systems for their interest.

DECLARATION

I hereby declare that the research work reported in the dissertation entitled "THE PREDICTION OFCODE CLONE QUALITY BYEXTRACTING FAST AND PRECISE CLONE GENEALOGIES IN COMPLEX CODE BASES" 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 Ms.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.

Geetika

11501877

SUPERVISOR'S CERTIFICATE

This is to certify that the work reported in the M.Tech Dissertation entitled **"THE PREDICTION OF CODE CLONE QUALITY BY EXTRACTING FAST AND PRECISE CLONE GENEALOGIES IN COMPLEX CODE BASES"**, submitted by **Geetika**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:

Concerned HOD:

HoD's Signature: _____

HoD Name: _____

Date: _____

Neutral Examiners:

External Examiner

Signature: _____

Name: _____

Affiliation: _____

Date: _____

Internal Examiner

Signature: _____

Name: _____

Date: _____

ACKNOWLDGEMENT

It is not until you undertake research like this one that you realize how massive the effort it really is, or how much you must rely upon the selfless efforts and goodwill of others. I want to thank them all from the core of my heart.

I owe special words of thanks to my supervisor Ms. Sandeep kaur for her vision, thoughtful counselling and encouragement for this research on " **THE PREDICTION OF CODE CLONE QUALITY BY EXTRACTING FAST AND PRECISE CLONE GENEALOGIES IN COMPLEX CODE BASES**". I am also thankful to the teachers of the department for giving me the best knowledge, guidance throughout the study of this research.

And last but not the least, I find no words to acknowledge the financial assistance & moral support rendered by my parents and moral support given by my friends in making the effort a success. All this has become reality because of their blessings and above all by the grace of almighty.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE NO.	FIGURE DESCRIPTION	PAGE NO.

KEYWORDS

1. Code clone
2. Plagiarism
3. Reuse
4. Semantic
5. Syntactic
6. Similarity

CHAPTER 1
INTRODUCTION

Programming code cloning is comprehensively utilized by designers to make code in which they have conviction and which lessens headway expenses and upgrades the item quality. Programming clone investigation in the early years was by and large fixated on the recognition.Some normal practices of the programming change like Copy and paste, reusing the code; it is assessed in the programming field, the codes which are fundamentally similar or semantically tantamount present wherever. The path toward replicating a code is known as code clone and examination of code clones, though an investigation of has late extends to the whole scope of clone organization[1].

Reusing programming through replicating and pasting is a constant pain in programming change paying little heed to the way that it incites sincere maintenance scrapes. The way toward copying a code is known as code clone. A few software engineers perform code cloning purposefully or unexpectedly amid the advancement of an application or programming. It has been studied that 30% of the code in the majority of the product organizations is replicated code. So it is basic to realize that why the code has been copied, why there is a need to copy the code, how the duplicated or cloned code negatively affects the maintenance and advancement.

For support and advancement reason, a few stages like clone discovery, investigation and upkeep have turned into a noteworthy territory of research for some specialists. In spite of the fact that cloning has many focal points in programming ventures. It spares the software engineer's chance, reuse of code is simple for an apprentice in the business. Be that as it may, as we reuse the code, the overhead additionally increments. So cloning has a dull side also.

The immense matter of concern is the maintenance of the created software. Some of the time the cost for maintenance surpasses more than the cost of the improvement. The

bug location, infection acknowledgment may likewise require the extraction of organized or semantically near clones. Each perspective has two confronts like a coin has two sides so as the code cloning.

1.1 CLONE TERMINOLOGIES

1. **Code fragment**: Code area (some part of a code) is any progression of code lines with or without remarks. It is distinguished by code piece filename, code part start line, code section end line.
2. **Code clone:** At the moment that a code some portion of document two is a clone of another code segment of record one.
3. **Clone pair:** One course of action of a code segment is indistinguishable to other whether in a same document or in another record, they are said to be a clone pair.
4. **Clone class:** When many pieces are like each other or on the other hand, there exists a clone-relationship between them then they make a clone class

1.2 TYPES OF CLONES

Type 1: These types of clones are otherwise called as exact clones. In this type of clone, there is a little bit more chance of variation in whitespaces and comments, but as the name suggests they are exact or identical clones.

Table 1.1 Exact clone

ntfooadd(intnum[],int v){
int z=0;//fooadd
for(int p=0;p<v;p++){
z=z+num[p];
}
return z;
}
	intfooadd(intnum[],int v){
	int z=0;
	for(int p=0;p<v;p++){
	z=z+num[p];
	}
	return z;
	}
		intfooadd(intnum[],int v){
		int z=0;//fooadd
		for(int p=0;p<v;p++){
		z=z+num[p];
		}
		return z;
		}

Type 2: These types of clones are known as renamed orparameterized clones. The structure or the syntax of this type of clone is same but there can be exceptions of layouts, variables, literals and in comments.

Table 1.2 Syntactic clone

intfooadd(intnum[], int v){
int z=0;//fooadd
for(int p=0;p<v;p++){
z=z+num[p];
}
return z;
}
	intdofooadd(int no[], int v){
	int z=0;
	for(int p=0;p<x;p++){
	fooadd=fooadd+no[p];
	}
	return fooadd;
	}
		intfooadd(int s[], int v){
		int z=0;//fooadd
		for(int p=0;p<v;p++){
		z=z+s[p];
		}
		return z;
		}

Type 3: These types of clones are known as Near- Miss Clones. Some amendments are done in the code like adding or removing new statements, modification in layouts, modification in literals, changing the name of the variables. If there is deletion of a statement in another code fragment, then they are termed as Near-Miss clones.

Table 1.3 Near-Miss clone

intfooaddition(intnum[], int n){
intfooadd=0;//fooadd
for(inti=0;i<n;i++){
fooadd=fooadd+num[i];
}
return fooadd;
}
	Intd.fooadd(int no[], int n){
	int a=0;
	for(inti=0;i<n;i++){
	a+=no[i];
	}
	return a;
	}
		intfooadd(int a[],int n){
		int x=0;//fooadd
		for(inti=0;i<n;){
		x=x+a[i];
		i++;
		}
		return x;
		}

Type 4: These sorts of clones are called as semantic clones. If two code pieces have similarity in their function or their behavior is similar, then they would be considered as semantic clones. Textual similarity is not the necessity. But it is not necessary in every case that code fragment is copied from the native code.

Table 1.4 Semantic clone

intfooadd(int no[],int n){
intfooadd=0;
for(int q=0;q<n;q++){
fooadd=fooadd+no[i];
}
return fooadd;
}
	intfooadd(int no[],int z){
	if(z==1)
	return no[z-1];
	else
	return no[z-1]+fooadd[no,z-1];
	}

1.3 REASONS OF CLONING

- **Lack of Interpretation of requirements:** Here and there, it is hard to translate and make an orderly approach for every single prerequisite as a result of the high number of determinations in extensive frameworks.
- **Tested code:** As there is always risk associated with new code because programmer can develop the code which might be more mind boggling or more inclined to bugs and errors. So to copy code is always preferable choice.
- **A Matter of chance:** By co-incidence, codes can be similar.
- **Preferring Developer's Credibility:**In most of the company's developer's performance is measured by checking how much number of lines he is producing in one hour.
- **Little knowledge of the new language:**Sometimes programmer does not have the better command over the programming language; at that moment they prefer copy and paste technique.

1.4 ADVANTAGES OF CLONING

- **Quick process**: When a programmer starts a code from the scratch, it takes lots of time and effort. So, copy and paste mechanisms are easier to develop a system.
- **Foundation for templates**: Template building is supported by code cloning. For example: same types of design are followed in all pages of many websites.

- **Encouraging reuse:** To achieve already existing Functionality of the tested code, reusing is done by copy and paste technique.

1.5 DISADVANTAGES OF CLONING

- **Rise in the need of resources:** Program becomes bigger and complex with the cloning. The number of hardware and software are needed to meet the requirements.
- **Likelihood of poor design increases:** Modular and structural programming approach is not being followed. When a clone is used in the program, it leads to poor design and ultimately it hampers the quality.
- **Maintenance becomes a tedious task:** To maintain the cloned code which complicates the understanding of the code, becomes a difficult work for the maintenance team.
- **Rise in cost and time:** If a bug is detected, then to remove it in the entire code takes a huge amount of time and effort as well as the cost increases for modification.

1.6 TECHNIQUES OF CLONE DETECTION

- **Text based clone detection technique:** Detection is not performed on the premise of syntactic and semantic similarity. Line by line comparison will be done on the two code fragments.If textual similarity exists between them, then they are counted as clones.
- **Abstract-syntax tree based clone detection technique:** Codes are parsed into a tree based algorithm [21] or tree based matching, if a match is detected then the result would be a clone. Generally, near- miss clones are represented in the abstract syntax tree and then on the result, pattern matching is applied.
- **Token-based clone detection technique:** By using the concept of lexical analysis or study, source code is converted into the tokens. Exact clones and syntactic clones are traced out with this technique.

- **Graph-based clone detection technique:** From the source code, the program dependency graph is acquired which includes control flow and data flow. It contains behavior or semantic information of a two codes.
- **Metric-based code clone detection technique:** Distinctive measurements of codes are computed. Measurements contain data about the name of strategies, formats, literals and control of the project. The parts of code which will demonstrate comparable metric qualities are considered as clones.
- **Hybrid clone detection technique:** By mixing an using two or more above mentioned stechniques clones can be detected. This technique holds better value than normal technique. For example: graph and metrics technique can be used in a combination for best results.

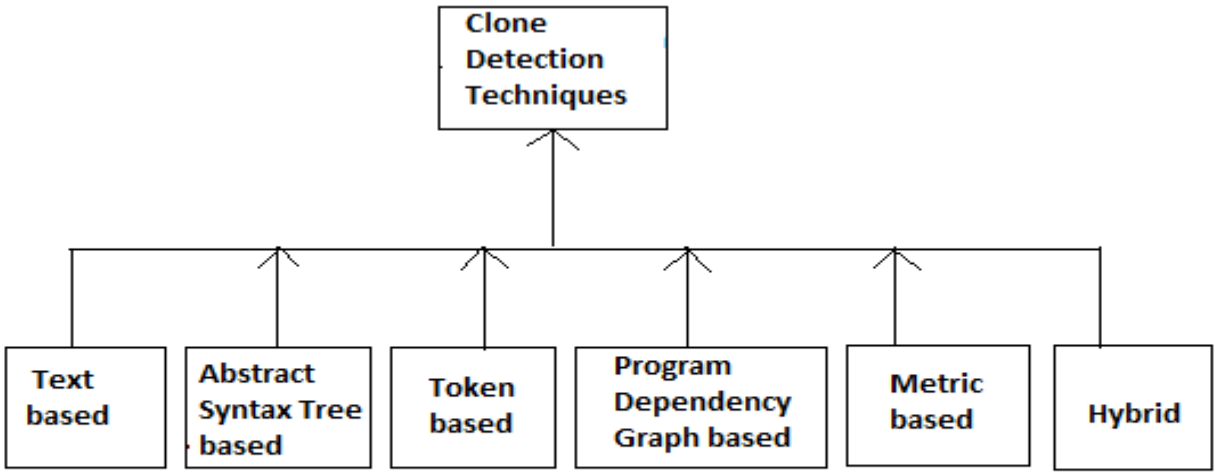

Figure 1.1 Clone Detection Techniques

1.7 GENERIC CLONE DETECTION PROCESS

There are the generic steps involved in detecting clones whether they are actual clones or not. This process is quite expensive, requires fast computation speed. On the basis of similarity, clones are detected from the clone pairs.

1) **Pre-processing:** This phase follows two steps: one is divided, the source code into the sections also known as segmentation. Secondly, figure out the area of comparison. There are certain objectives of this phase:
   - **Elimination of unwanted parts**:Source code is segmented and uninterested parts are removed, which may generate false positive values. Reckoning of further steps would be easy.
   - **Figure out source units:**Once the removal of unwanted code is completed, then the rest of the source code is partitioned in such a way so that common portion can be obtained. For an instance: in a program; files, classes, functions/methods, start finish blocks, or source line sequence.
   - **Figure out comparison units:** Segmentation of the source units to further obtain smaller units for the comparison purpose.
2) **Transformation:** For the comparison purpose, the main motive of this phase is to convert the source code units into peculiar intermediate representations. This process is called as extraction. This step is further subdivided into following:
   - **Extraction:** To make source code appropriate as input to the real algorithm, conversion of source code has done.
   - **Tokenization:** Every line of source code is isolated into tokens.
   - **Parsing:** To indicate the clones in syntactic approach, abstract syntax tree is used to compare algorithms for the same sub-trees.The Metric-based approach can also be used.
3) **Match detection:** Transformed code which is obtained from the above steps is put into comparison algorithm where all the transformed comparison units are evaluated on the basis of similarity to determine the matches. A set of candidate clone pairs will be obtained. The algorithms used in this phase are: suffix tree dynamic pattern matching and hash esteem examination.
4) **Formatting:** The clone pair list for the changed code obtained by the comparison algorithm is transformed over to a relating clone pair list for the first code base.
5) **Post processing/filtering:** This step is further subdivided into two parts:
   - **Manual analysis:** Here false positives are filtered out by human experts.

- **Automated heuristic***:* Few parameters are already set according to filtering purposes. For example: length, frequency, diversity etc.

6) **Aggregation:** With a specific end goal to expel the information, perform ensuing examination or accumulate outline measurements, clones might be collected into clone classes.

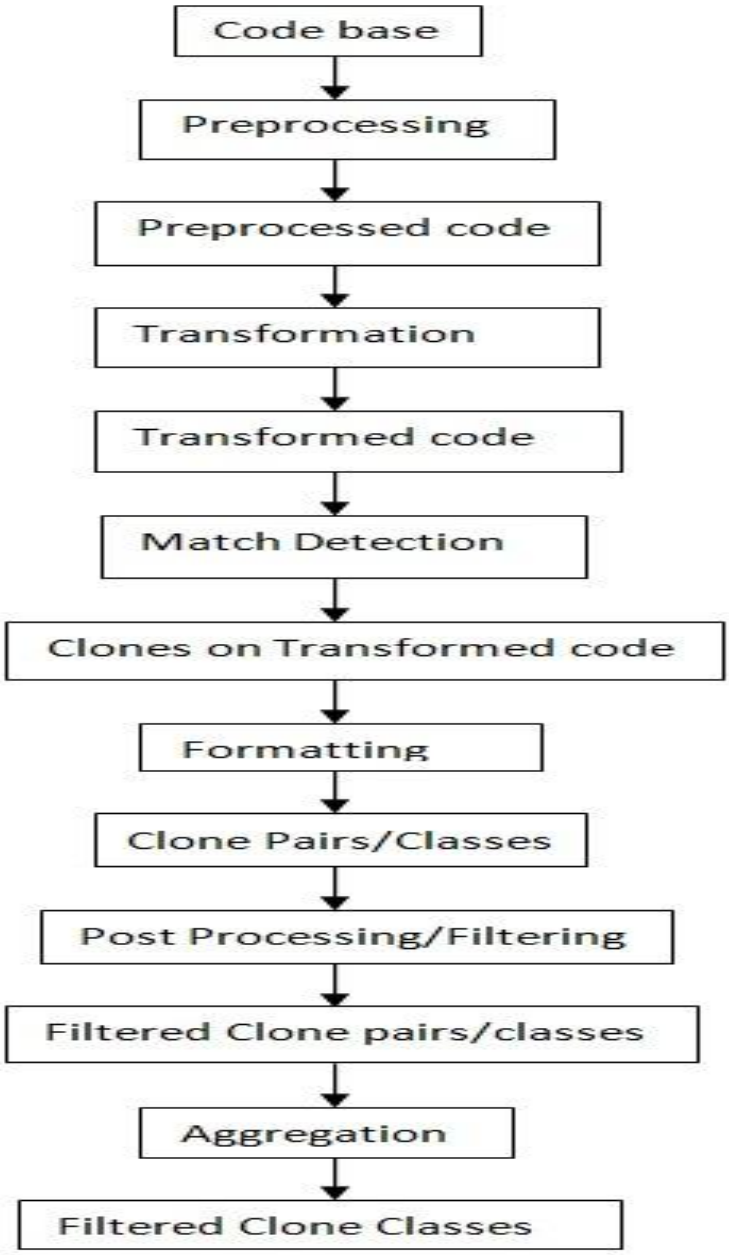

Figure1. 1.2 Generic clone detection process

SUMMARY

This chapter is to present the field of programming, building in which programming cloning and its discovery. The part started with the essential wordings of imperative terms. To comprehend the pertinence of discovery of clones, the nonexclusive clone recognition prepares to exhibit. The inspiration for this postulation happens to be of clones to decrease the upkeep procedure at the later phase of the advancement of programming or any venture.

CHAPTER 2
REVIEW OF LITERATURE

Software or code cloninghas become a major area of research these days. Many researchers diligently exploring this topic and so many approaches have been developed to probe duplicate codes. These approaches are syntax-based[2], text- based[3], graph based [4] and metric based [5].

2.1Survey on clones

Chanchal K. Roy *et al.*[6]have performed the comparison and evaluations of different techniques and tools. First of all the reorganizations and then evaluations of different approaches are being performed on the basis of some restrictions and on the basis of types of clones. This paper aids to detect different clone detectors.

Robert Tairas,JeffyGray[7]shows the expanding clone up support by consolidating clone identification and rewriting an existing source to improve its readability, reusability (refactoring) activities simply by modifying the structure of the code yet without changing the conduct of code. They have proposed CeDAR (clone recognition, investigation and refactoring) code yet without changing the conduct of code. This tool focuses only on Type1 and Type 2 clones. The results of clone detection techniques and refactoring activities for eliminating duplicate code and for maintenance of code clones have been accumulated.

Ripon K. Saha*et al*[8] have shown automaticdetection of evolution pattern of both exact and near-miss clones by constructing their groups and they have developed a prototype "gCad" which is scalable to various clone detection tools. For detecting the change in pattern some of the key similarity factors have been used. They have built up a prototype clone genealogy (bunch) extractor, which is further connected on three open source ventures including the Linux kernel.

Dhavleesh Rattan *et al.*[9] have reviewed the programming clones. In theirliterature review, near about 100 studies from literatures were based on software clone detection. The result of these studies is also categorized as types of clones, internal

representation of clones, semantic clones, model clones, code clone management, different approaches of clone detection.

Yaowenchen*et.al.* [10]havepresented an experimental study on code cloning in more than twenty open source games by applying abest in class clone locator, NiCad. They additionally utilized VisCadtool for perception and investigation of clones. This exploration has demonstrated that cloning happens at between inter-project level, as well as at an intra-project. On the premise of various measurements, for example, language category, clone density and the clone area, they broke down an arrangement of measurements and prerequisite of embracing clone management frameworks for game improvement..

Balazinska *et al.* [11] brought out a more refined course of action for limit clones as delineated in Table 1. This request looks good to choose a sensible framework for clone ejection. For instance, the arrangement plan Template Method may be used to compute out complexities the sorts used as a piece of different code segments or the arrangement outline scheme can be used to make sense of algorithmic complexities.

2.2 Text-based approach

Rieger *et al.* [3] looked at entire lines to each other literarily . To expand execution, lines are apportioned utilizing a hash work for strings. Just lines in a similar parcel are thought about. The outcome is visualized as a dotplot, where every spot demonstrates a couple of cloned lines. Clones might be found as specific examples in those dotplots outwardly. Continuous lines can be outlined to bigger cloned arrangements consequently as continuous diagonals or dislodged diagonals in the dotplot.

Marcus*et al*.[12] looked at specific bits of content, in particular, identifiers utilizing idle semantic ordering, a strategy from data recovery. The thought here was to distinguish sections in which comparative names happen as potential clones.

2.3 Metric-based approach

Madhulina Sarkar*et al.* [13]used clone detection technique to forecast the resource requirements, feedback guided by automatic jobmodelingmethodology which has been founded on the metric based clone discovery. When the job is entered in a system, its execution is bolstered and assets are included or evacuated on the premise of a

versatile execution plan displayed in. A tool called PRAGMA is used to implement this scheme.

2.4 Token Based approach

Baker *et al.*[14]has presented in his paper about Dup is mix of content based and token based technique which partition program in parameterized and non-parameterized tokens to find Type I and Type II clones. It uses hashing limit as a piece of demand to find Type I clone and position file forType II clones.

Toshihiro Kamiya*et al.* [15] has suggested about CCFinder is one of the effective token based instruments which can distinguish code clones from Java, C, C++, COBOL and numerous other source program records. This apparatus change over source record into arrangement of tokens and afterward correlation of these tokens areformed with the assistance of addition tree calculation. It likewise gives clone measurements to discover clone combines and clone class. In addition for more correct perception disperse chart and plot outlines are utilized.

2.5 Graph-based approach

Jens krinke*et al.*[16] has identified similar codes with fine-grained program dependence graphs and this approach works not only on the syntax of a program but also on the semantics. Prototype model is used with the non-polynomial complexities which yields high precision and recall. This approach has not worked well with a polynomial time limit.

Mark Gabel *et al.* [17] have performed scalable detection of clones on the basis of semantic clones. Millions of lines of code have been evaluated using their algorithm. The program dependence graphs (PDG) [18] problem which has been used to implement program slicing [19], have been reduced to a simple tree similarity problem. Some of the productive clone recognition methods which are utilised to discover fundamentally comparative clones are DECKARD [20], CP-Miner [21], and CCFinder[15] .

2.6Abstract-syntax based approach

Ira D.Baxter*et al.*[2]havepresented a realistic strategy for recognizing near-miss and arrangement clones on scale has been introduced. The approach has been depended

on varieties of strategies for compiler regular sub expression disposal utilizing hashing. The strategy was direct to actualize, utilizing standard parsing innovation, recognized clones in discretionary dialect builds, what's more, registers macros that permit expulsion of the clones without influencing the operation of the program. They have connected the strategy to a genuine use of direct scale, and affirmed past appraisals of clone thickness of 7-15%, proposing there was a "manual" programming building prepare "repetition" steady.

2.7 Hybrid approach

Chanchal K. Roy[22]has performed discovery and examination of near-miss software clones. They have executed their task in four steps. They have developed a hybrid clone discovery technique, proposed a Meta model of clone sorts, furthermore, they have given a situation based examination of clone recognition procedures and instruments. They have used NICAD tool which was not able to detect Type 4 semantic clone.

Gehan M.K. Selim *et al*.[23] represented an enhancement in clone detection, which are based on source-based by using intermediate representation. They have used a hybrid approach for detection of Type 3 clones. In clone genealogies, their technique has higher accuracy on the correlation with standalone string based and token based clone detector.

Abdullah Sheneamer*et al.*[24] have introduced a hybrid clone recognition strategy that first uses a coarse-grained method to break down clones adequately to enhance exactness. Subsequently,they have utilized a fine-grained indicator to get extra data about the clones and to enhance review. Their strategy has distinguished Type-I and Type-2 clones utilizing hash values for pieces, and gapped code clones (Type-3) utilizing piece recognition and ensuing correlation between them utilizing Levenshtein separation and Cosine measures with shifting edges.

Madhulina Sarkar *et al*[25]**.**have been augmented the scientific classification of clones proposed by different analysts keeping in mind the end goal to make asset necessity expectation more compelling. It additionally shows a hybrid clone-location

system, compri sing of measurements based, PDG-based and AST-based clone location, to make the clone recognition prepare more solid furthermore, hearty.

Bhagwan*et al*.[26]have shown a hybrid technique that recognizes programming code clones for Java programs on the preface of estimations and substance based methodologies. Their proposed approach scans for clones in the code at the file level, class level, and methodology level. Their approach has recognized potential clones on metric-based match. Potential clones are furthermore analyzed line by line using a text based approach to manage check whether the potential clones perceived using metric based examination are truly clones or not.

Table 2.1 Generic survey on clones

S. No.	Title Of Paper	Authors	Contribution Of Paper
1.	Comparison and evaluation of code clone detection techniques and tools: A qualitative approach	Roy, Chanchal K. Cordy, James R. Koschke, Rainer	First of all the reorganizations and then evaluations of different approaches are being performed on the basis of some restrictions and on the basis of types of clones. This paper aids, to detect different clone detectors.
2.	Increasing clone maintenance support by unifying clone detection and refactoring activities	Tairas, Robert Gray, Jeff	CeDAR (clone recognition, investigation and refactoring) code yet without changing the conduct of code. This tool focuses only on Type1 and Type 2 clones. The results ofclone detection techniques and refactoring activities for eliminating duplicate code and for maintenance of code clones have been accumulated
3.	Software clone detection : A systematic review	Rattan, Dhavleesh Bhatia, Rajesh Singh, Maninder	In theirliterature review, near about 100 studies from literatures were based on software clone detection. The result of these studies is also categorized as types of clones, internal representation of clones, semantic clones, model

			clones, code clone management.
4.	Near-miss Software Clones in Open Source Games : An Empirical Study	Chen, Yaowen Roy, Chanchal K	This research has showed that cloning happens not only at inter-project level, but also at an intra-project. On the basis of different dimensions, such as language category, clone density and the clone location, theyanalyzed a set of metrics and requirement of adopting clone management systems for game development
5.	Advanced clone-analysis to support object-oriented system refactoring	Balazinska, Magdalena Merlo, Ettore Dagenais, Michel Lagiie, Bruno Kontogiannis, Kostas	This paper has introduced a more refined course of action for limit clones This request looks good to choose a sensible framework for clone ejection. For instance, the arrangement plan TemplateMethod may be used to compute out complexities the sorts used as a piece of different code segments or the arrangement outline Strategy can be used to make sense of algorithmic complexities

Table 2.2 Textual survey on clones

S. No.	**Title Of Paper**	**Authors**	**Contribution Of Paper**
1.	A language independent approach for detecting duplicated code	Ducasse, S. Rieger, M. Demeyer, S.	The outcome is visualized as a dotplot, where every spot demonstrates a couple of cloned lines. The clones might be found as specific examples in those dotplots outwardly.
2.	Identification of high-level concept clones in source code	Marcus, A. Maletic, J.I.	This paper looked at specific bits of content, in particular, identifiers utilizing idle semantic ordering, a strategy from data recovery. The thought here was to distinguish sections in which comparative names happen as potential clones

Table 2.3 Metric based survey on clone

S. No.	Title Of Paper	Authors	Contribution Of Paper
1.	A hybrid clone detection technique for estimation of resource requirements of a job	Sarkar, Madhulina Chudamani, Sameeta	This paper has used clone detection technique to forecast the resource requirements, feedback guided by automatic job modelling methodology which has been founded on the metric based clone discovery.

Table 2.4 Token based survey on clone

S. No.	Title Of Paper	Authors	Contribution Of Paper
1.	A Program for Identifying Duplicated Code	Baker, Brenda S	It utilizes hashing capacity as a part of request to discover Type I clone and position file for sort II clones.
2.	CCFinder: A multilinguistic token-based code clone detection system for large scale source code	Kamiya, Toshihiro Kusumoto, Shinji Inoue, Katsuro	This apparatus change over the source record in the arrangement of tokens and afterward correlation of these tokens are made with the assistance of addition tree calculation. It likewise gives clone measurements to discover clone combines and clone class. In addition,for better perception disperse chart and plot outlines are utilized.

Table 2.5 Graph based survey on clone

S. No.	Title Of Paper	Authors	Contribution Of Paper
1.	Identifying Similar Code with Program Dependence Graphs	Krinke, Jens	This paper has identified similar codes with fine-grained program dependence graphs and this approach works not only on the syntax of a program but also on the semantics. Prototype model is used with the non polynomial complexities which yields high precision and recall.
2.	Scalable Detection of Semantic Clones	Gabel, Mark	This paper has contributed in performing scalable detection of clones on the basis of semantic clones. Millions of lines of code have been evaluated using their algorithm. The program dependence graphs (PDG) problem which has been used to implement program slicing , have been reduced to a simple tree similarity problem
3.	Code clone detection using coarse and fine-grained hybrid approaches	heneamer, Abdullah Kalita, Jugal	This paper has introduced a hybrid clone recognition strategy that first uses a coarse-grained method to break down clones adequately to enhance exactness. Subsequently,they have utilized a fine-grained indicator to get extra data about the clones and to enhance the review
4.	Software clone detection : A systematic review	Rattan, Dhavleesh Bhatia, Rajesh Singh, Maninder	In theirliterature review, near about 100 studies from literatures were based on software clone detection. The result of these studies is also categorized astypes of clones, internal representation of clones, semantic clones, model clones, code clone management, different approaches of clone detection.

5.	A hybrid clone detection technique for estimation of resource requirements of a job	Sarkar, Madhulina Chudamani, Sameeta Roy, Sarbani Mukherjee, Nandini	It has showed a hybrid clone-location system, comprising of measurements based, PDG-based and AST-based clone location, to make the clone recognition prepare more solid furthermore, hearty.
6.	Design and Analysis of a Hybrid Technique for Code Clone Detection	Bhagwan, Jai Pramila, Kumari	They have displayed a hybrid method that distinguishes programming code clones for Java programs on the premise of measurements and content based methodologies. Their approach has distinguished potential clones on metric-based match.

Table 2.6 AST based survey clones

S. No.	Title Of Paper	Authors	Contribution Of Paper
1.	Detection and Analysis of Near-Miss Software Clones	Roy, Chanchal K	They have developed a hybrid clone discovery technique, proposed a Meta model of clone sorts.They have used NICAD tool which was not able to detect Type 4 semantic clone.
2.	Enhancing source-based clone detection using intermediate representation	Selim, Gehan M K Foo, King Chun Zou, Ying	They have used a hybrid approach for detection of Type 3 clones. In clone genealogies, their technique has higher accuracy.

SUMMARY

This part exhibits the detail of all the writing overviewed and evaluated for the comprehension of subject. Papers identified with code clone recognition and the sorts of clones distinguished have been given in this part. For getting a handle on the information about hybrid approach, different papers exhibiting distinctive advances developed.It has been found amid the writing, review that there is still more work to do on crossover way to deal with get fast,precise clones. Part 3 represents to the extent of hybrid approach to deal with show signs of improvement results with high recall value.

CHAPTER 3

PRESENT WORK

3.1 PROBLEM FORMULATION

Clone Detection is to inconceivable stress for finer support and the nature of the programming system. Systems comprising code clones are significantly unprotected to bugs and inconsistencies and move toward getting to be a hindrance for finer improvement of programming structure.Subsequently, it is an open territory of research for many years and results into different clone detection procedures and devices in light of them. Yet, as talked about in writing, study certain impediments are related with each clone discovery technique and device.

Tools in light of text based methods are connected specifically with respect to source code can ready to recognize just Type I clones though devices in view of tokens have done lexical examination of the source code and ready to distinguish Type II clones moreover. In any case, both these systems can recognize just syntactic comparative codes and comes up short if any alteration is done inside the statements. Even so, a few tools utilize the idea of abstract syntax tree all together discover Type III clones yet this approach necessitates complex calculations and parser to discover comparable sub trees. Metrics based tools are somewhat straightforward and reasonable for the vast programming framework, however it is not effective if connected straightforwardly in source code.

Program Dependence Graph based system is the main strategy which can identify code clones both grammatically and in addition semantically.

So taking after issues are distinguished in existing work:

1) A novel advancement is expected to identify code clones effectively. Both syntactic to semantic clones ought to be distinguished by a device.

2) Numerous false positive clones are identified by tools which ought to be evacuated to get high accuracy values.
3) As the clone recognition process has many stages, so tools ought to be automatic and weightless so that each stage is executed with no extra computational assets.
4) There is a requirement of hybrid approach which utilizes diverse procedures other than tree based clone detection strategy.

This proposal displays a hybrid way to deal with identifying actual code clones. It consolidates token based method with program dependency graph based technique to discover code clones for Java and C++ programs. Initially tokens are produced utilizing pre-processed documents in which remarks and void areas are expelled. The created tokens by documents experiences, Smith-Waterman calculation for comparison purpose and are contrasted with discover potential clones.

In the wake of getting potential clones, PDG is gotten utilizing pre-processed code, which conveys semantic data of programs and with PDGtechnique we get control and data dependencies which are computed to confirm potential clones as actual clones.

A clone detector tool is proposed to discover genuine clones for Java and C++ programs by utilizing hybrid approach. This is a mechanized tool with easy to understand interface which shows acquire token and PDG and gives results, whether tried projects are actual clones or not.

3.2 OBJECTIVES OF THE STUDY

1. To propose and enhanced hybrid technique for software code clone detection using tokenization and program dependency graph based techniques.
   1.1. To perform preprocessing of input files
   1.2. To perform lexical analysis on preprocessed files for generating token sequence.
   1.3. To execute Smith-Waterman algorithm for comparison of generated token sequence and to check potential clones in the files.
   1.4. To extract number of control dependent nodes and data dependent nodes from preprocessed files using program dependency graph based technique.
   1.5. To use Smith-Waterman algorithm for validating potential clones.
2. To implement a tool using the proposed technique for object-oriented languages like Java and C++.
3. To compare precision, recall and accuracy of the proposed approach with the existing base paper approach.

3.3 RESEARCH METHODOLOGY

Code clones are considered as risky to the maintenance and rightness of programming systems if they are not dependably managed. It is evaluated that 85% of total programming cost spent on the support issues. Refactoring and other re-designing activities are used to empty them, however , not beneficial for an extensive variety of clones and it requires high cost. From now on various clone detection techniques and devices are proposed in writing are remembering the ultimate objective to recognize clones yet certain limitations are identified with them. Furthermore, it's not useful to track code clones physically. So clone detection is an open research area and there is a requirement of a mechanized tool which can identify clones successfully.

The proposed work demonstrates a robotized clone detection tool for Java and c++ programs. This tool is a hybrid approach based tool which solidifies token based method and program dependency graph based framework to recognize code clones beneficially.

3.3.1 Token-Based Approach

Token based [27]code portrayal gives a reasonable deliberation to clone recognition. It has both simplicity of flexibility to various languages, and awareness and control of the fundamental language tokens. Near reviews, including diverse clone identification methods have demonstrated that token based clone recognition tools perform well as far as accuracy and recall of the detected clones.

A token arrangement of the input code through a lexical analyzer (similar table representation is given below), and applies the rule-based changes to the succession. The design is to change code divides in a custom frame, to distinguish cloned code portions that have diverse linguistic structure however have similar importance. Another reason for existing is to sift through code portions with determined structure designs.

The tokens of all source files are connected into a solitary token arrangement to consistently recognize clones inside a document and crosswise over files. A few changes are additionally connected to this token string, contingent on the language, to limit the contrasts between comparative code pieces. For instance, evacuating the namespace attribution, introduction lists,stamping of function definition limits, expulsion of availability keywords and so forth for the C++ and Java code.

Here the sample code of C++ is used to explain the working of Token-based approach.

Table 3.1 Source Files

1.#include<iostream>
2.void main(){
3.int x;
4.int y=5;
5.int fact=1;
6.x=1;
7.for(x=1;x<=y;x=x+1){
8.fact=fact*x;
9.}
10.cout>>"value of fact: "+fact;
11.}
	1.#include<iostream>
	2.void main(){
	3.int x;
	4.int y=5;
	5.int fact=1;
	6.x=1;
	7.while(x<=y){
	8.fact=fact*x;
	9.x++;
	10.}
	11.count>>"value of fact: "+fact;
	12.}

File 1 contains factorial code using for loop whereas file 2 contains the same code using while loop.

Table 3.2 List Of Token

Line No.	No.of Token	Line No.	No.of Token
0	5	0	5
1	4	1	4
2	2	2	2
3	4	3	4
4	4	4	4
5	3	5	3
6	5	6	7
6	4	7	5
6	6	8	3
7	5	9	0
8	0	10	6
9	6	11	0
10	0		

List of tokens for file 1 and file 2 including their line number. It tells on a particular line how many numbers of token exist and this is done by the lexical analyzer. By comparing the token sequence further clones can be calculated.

Token-based approach is good to detect Type-1 i.e. exact clones and Type-2 i.e. parametrized or renamed clones.

3.3.2 Program dependency Graph

Program dependency Graph: It is a coordinated diagram which decides two sorts of dependency that exists between the statements of the source code. These two conditions are Control Dependency and Data dependency.

Control Dependency: It exists between two statements exactly when first articulation is restrictive phrase and execution of the second statement depend on upon the aftereffect of the main statement.

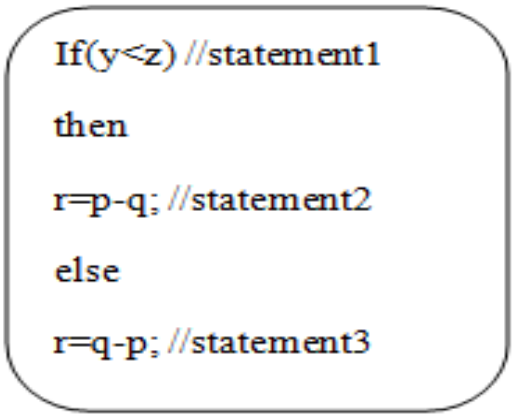

Figure 3.1 Control Dependency

In the above code execution, statement 2 and statement 3 depends upon the outcome of the predicate expression at statement 1. Subsequently, control dependency exists between them.

Data Dependency: It holds between two statements just when the first statement incorporates the meaning of the variable and the second statement utilizes the variable without redefinition of the variable. This can be clarified with the assistance of taking after the code:

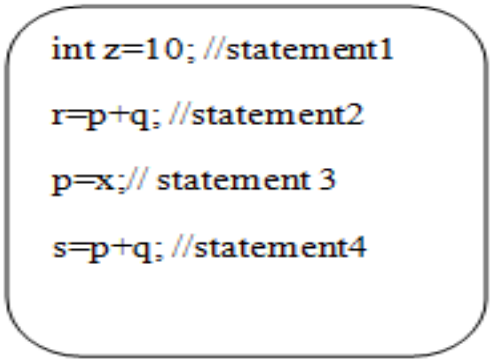

Figure 3.2 Data Dependency

Data dependent while statement 3 is not on the grounds that it contains the redefinition of the variable and statement 4 is dependent on statement 3 only.

3.3.3 Smith-Waterman algorithm

The Smith-Waterman calculation [28] is a calculation for recognizing comparative arrangements between two base groupings. This calculation has leverage that it can recognize comparative arrangements regardless of the possibility that they incorporate a few gaps**.** The Smith-Waterman algorithm comprises of the accompanying five stages[29].

Stage 1 Making a table: The (X+2)×(Y+2) table is made, where X is the length of one grouping(c1, c2, ・・・, cX)and Y is the length of the other arrangement (d1, d2, ・・・, dY).

Stage 2 Introducing the table: The top line and furthest left section of the table made in Step 1 are loaded with two base arrangements as headers. The second line and segment are introduced to zero.

Stage 3 Computing scores of each and every cells in the table: Scores of each and every the rest of the cells are figured by utilizing the accompanying equation.

$$P_{x,y}(2 \le x,\ 2 \le y) = \text{maximum} \quad (1)$$

$$\begin{cases} p_{x-1,y-1} + z(c_i, d_j), & (2) \\ p_{i-1,y} + gap, & (3) \\ p_{x,y-1} + gap, & (4) \\ 0 \end{cases}$$

$$z(c_x, d_y) = \begin{cases} match\ (c_x = d_y), & (5) \\ mismach(c_x \neq d_y) & (6) \end{cases}$$

where $P_{x,y}$ is the estimation of $s_{x,y}$; $s_{x,y}$ is the cell situated at the*xth* row and the *yth* column; *z(cx,dy)* is a closeness of coordinating *cx* with *dy ; cx* is the *xth* estimation of one arrangement and*dy* is the *yth* estimation of the other arrangement.

Stage 4 Follow back of the table: Follow back means the moving operation from $s_{x,y}$ to $s_{x-1,y}$, $s_{x,y-1}$ or $s_{x-1,y-1}$ utilizing the pointer made in Step 3. Following the pointer contrarily speaks to follow back. Follow back starts at the cell whose score is greatest in the table. This proceeds until cell values diminished to zero.

Stage 5 Distinguishing comparative arrangements: The exhibit components pointed by the follow back way are recognized as similar local arrangements.

3.3.4 Steps of proposed methodology:-

1. Create a database to store source files.
2. Loading source files into clone detector tool.
3. Lexical analysis is performed at this step, tokens will be calculated on the predefined length of statements. This phase is a tokenization phase.
4. Smith-Waterman algorithm works on the backend for comparing two files, potential clones are being detected at this stage, if any exist.
5. Formation of the clone group is done and we are extracting out potential clones from this round.
6. Up to this step, we can find the Type-1, Type -2 clones with high recall and precision value.
7. Further source files in a preprocessed form have again loaded to generate program dependency graph for getting semantic information of the files. This phase is also known as normalization phase, control and data dependencies are generated.
8. For comparison purpose, Smith-Waterman algorithm is again used to validate the potential clones obtained from tokenization phase.
9. Potential clones will be detected and extracted out and clone groups will be formed at this round.
10. Clone groups from both the rounds will be compared, potential clones will be checked whether they are actual clones or not.

11. We obtain final results and fine quality clones are being calculated.

12. Cloning percentage for each file is obtained and the cloned lines of the respective files are displayed on the results screen.

13. The addition and subtraction in the second clone fragment are compared with first cloned fragment, and results are displayed in percentage.

14.PDG approach is good for calculating type-3 and type-4 clones.

3.3.5 Algorithm of proposed methodology

Algorithm1 :Preprocessing and generate Tokens (file1, file 2)

Input: Either two java .class files or two c++ files

Output: Tokens

1.if(comments, whitespaces)

2.do

3.preprocessing of files

4.display(file1)

5.display(file2)

6.use preprocessed files.getTokensbyMethodSignature to create tokens

7.int b= SvToken.countTotalNumberOfTokens()

8.if tokens exist

9. print->line number with their token number

10.else

11.print ->there is no tokens in a file.

12.End

Algorithm 2. Smith –Waterman algorithm for Comparison(file 1.tokens , file 2.tokens)

Input: Two generated token files with token sequence (sequence of 4 tokens is used)

Output: Finds the potential clones exist or not

1. if (file 1.token= file 2.token)

2. Compare each local assignment of the files

3. if match found then print->potential clone

4. else if (file1.token != file 2.token)

5.print-> no potential clones exist

6.End

Smith-Waterman algorithm is used for comparison purpose and hereto recognize potential clones. In this token to token comparison is made as for clones programs;textual and syntactic similarity should exist between programs.

Algorithm 3: Generate_PDG (file1, file2)

Input: Either two Java .class files or two c++ files

Output: PDG

1. Use preprocessed files.getPDGByMethodSignatureto create PDG
2. int b = SvPDG.countTotalNumberOfNode()
3. Count all control and data dependency modes.
4. Print all control and data dependency on the GUI.
5. End

The program dependency graph is used to calculate Type-3 and Type-4 clones. Control and data dependent nodes are being calculated for the source files. PDG checks the behavioral similarity between the files.

Algorithm 4. Smith –Waterman algorithm for Comparison(file 1, file 2)

Input: Two generated PDG files with data and control nodes

Output: Finds the actual clones exist or not

1. if (file 1= file 2)
2. Compare each local assignment of the files
3. if match found then print->actual clone
4. else if (file1 != file 2)
5. print-> no actual clones exist

6 .End

Here, Smith-Waterman algorithm is used for comparison purpose and to detect actual clones by comparing generated PDG with potential clones.

3.3.6 Roadmap To Our Proposal

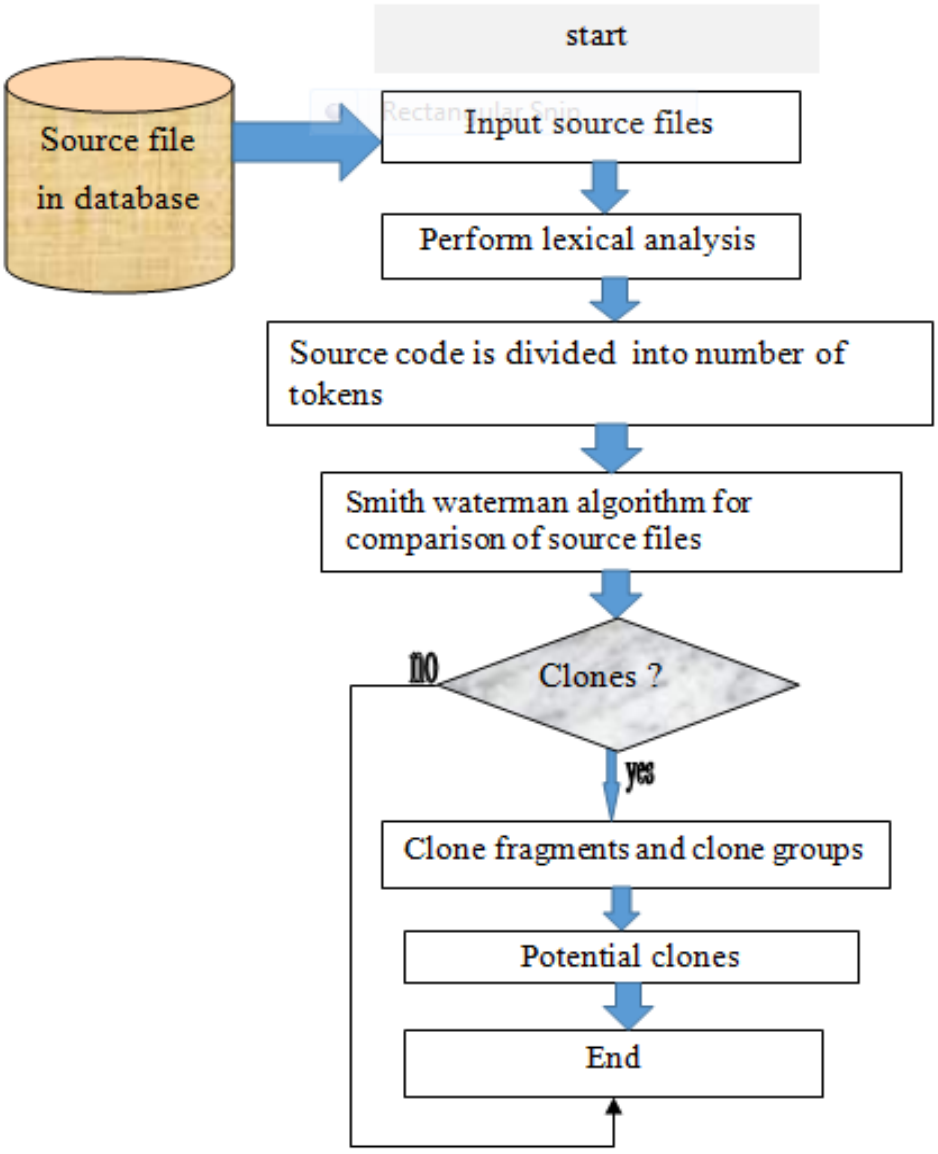

Figure 3.3 Clone detection by tokenization

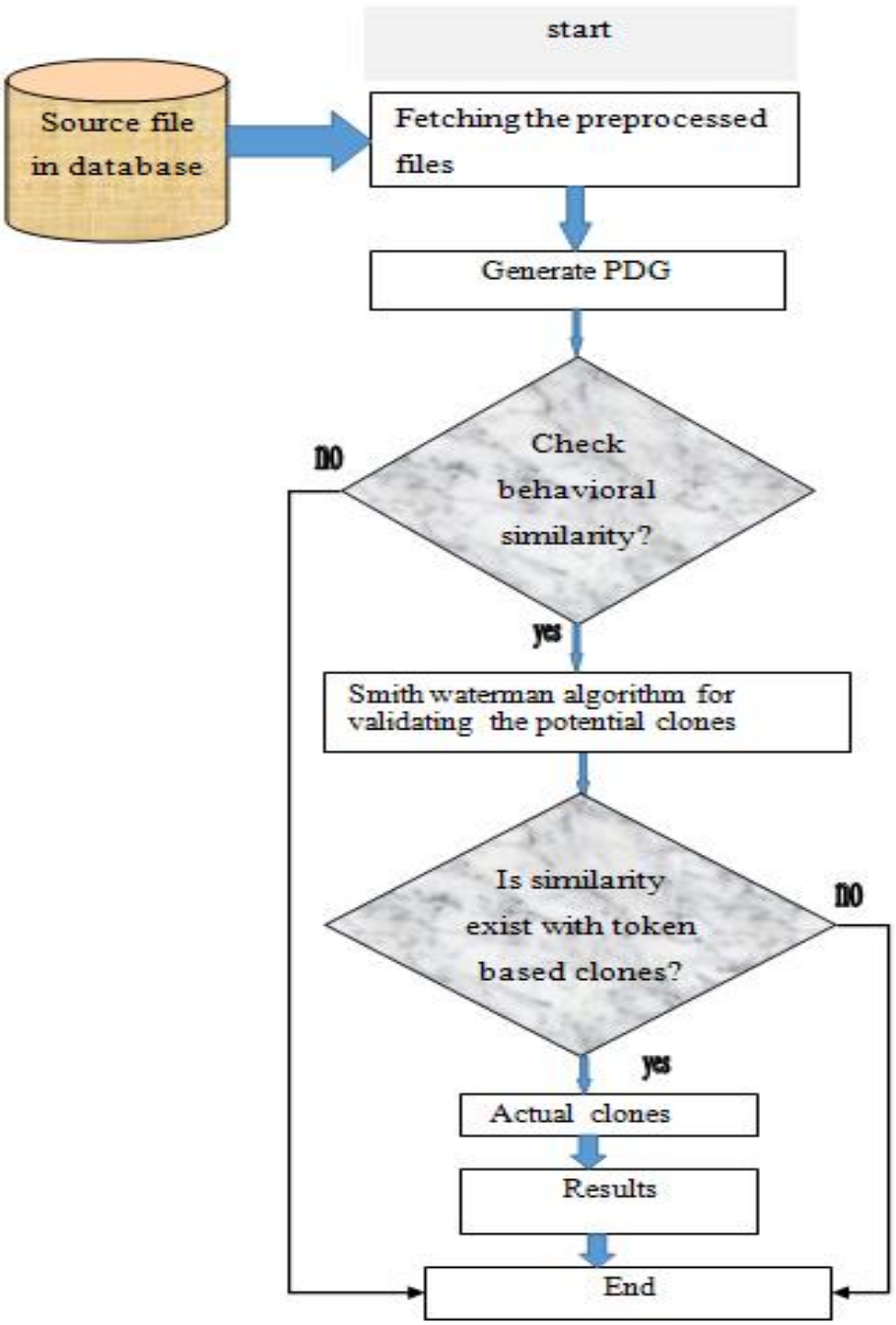

Figure 3.4 Clone detection by PDG approach

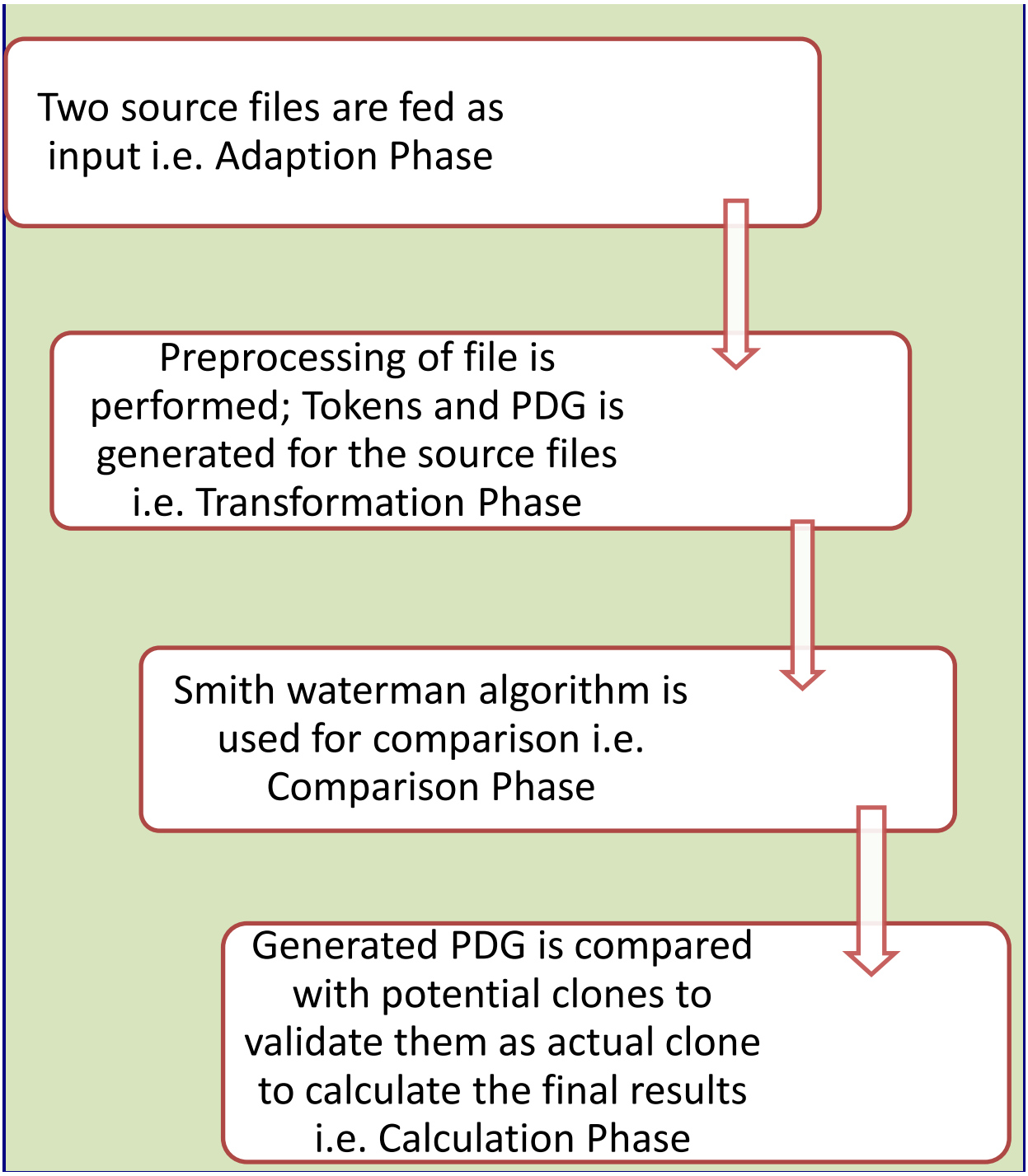

Figure 3.5 Phases involved in code clone detector tool

CHAPTER 4

RESULTS AND DISCUSSION

The functioning of the proposed tool begins with Adaption Phase, i.e. by giving either two Java or c++ code records as contribution with the assistance of the client. For this reason the startup page of code clone Detector is made by utilizing Java frames. To pick records with the help of client, Java FileChooser capacity is included which permit choosing just .class documents for Java programs and for c++programs;it chooses .cpp records. Figure 11 demonstrates the primary page which is the Netbeans IDE to run the project. Then the next page, displays our cone detector tool, files are selected there and then the event which pass the source document names to the function which produces token which are utilized for recognition of potential clones and afterward Program Dependence Graph for both the projects which are utilized for approval of potential clones in a program.

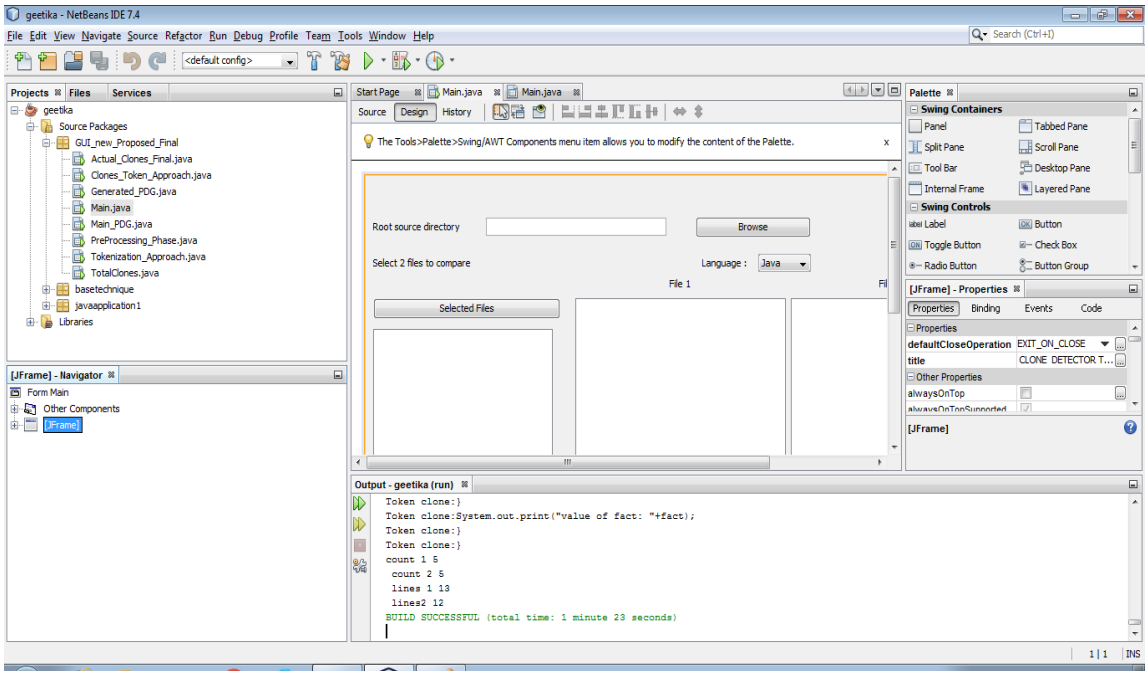

Figure 4.1 Netbeans IDE

This is the Netbeans IDE 7.4 version to open the project,first of all goto files, create a new project and give the desired name and select Java, a new Java application will be created; under the source packages we can open the projects to run the files.

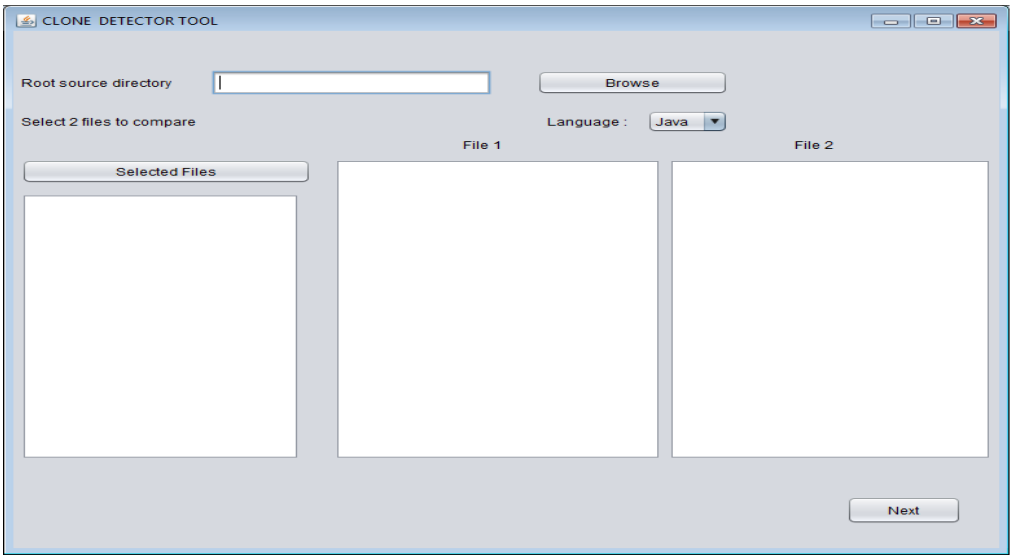

Figure 4.2 Start-up page

This is the initial window of our clone detector tool where browses, language selection, selected files and next buttons are given.

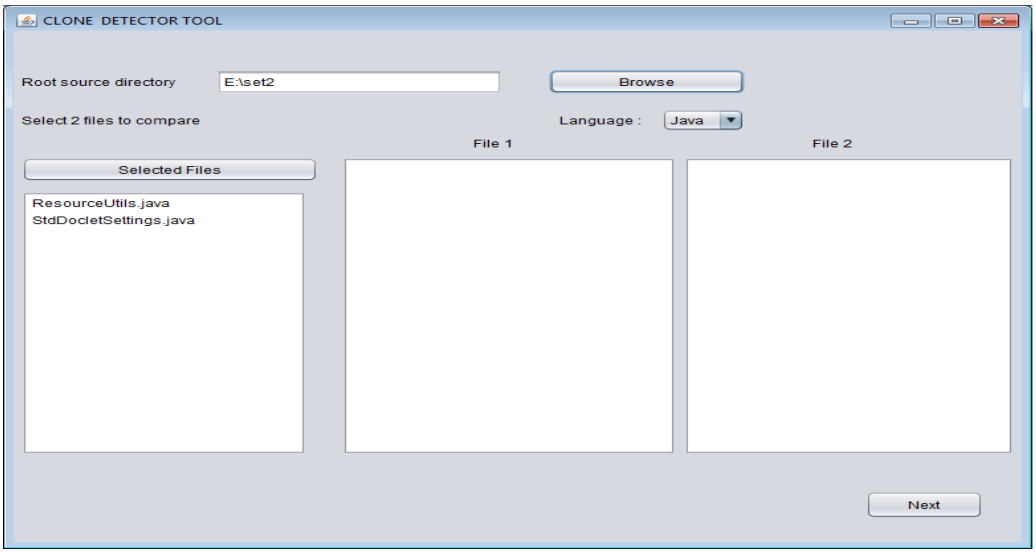

Figure 4.3 Browse files

Bowse the files location which are to be detected and choose the language,files will be shown below the window.

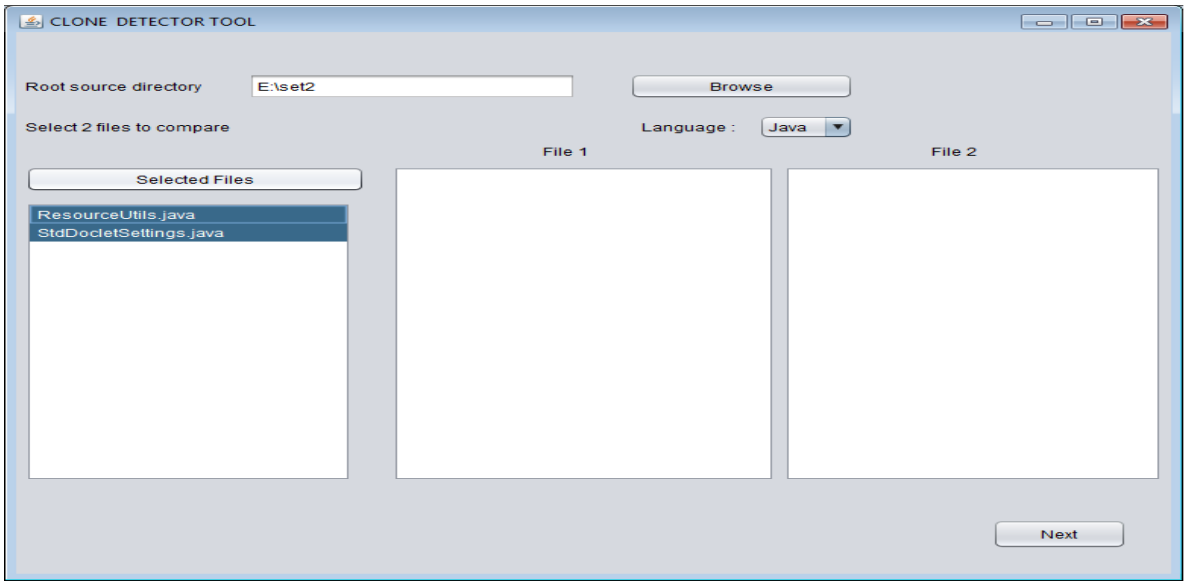

Figure 4.4 Select files

Select the files, which are displayed on Clone Detector Tool window.

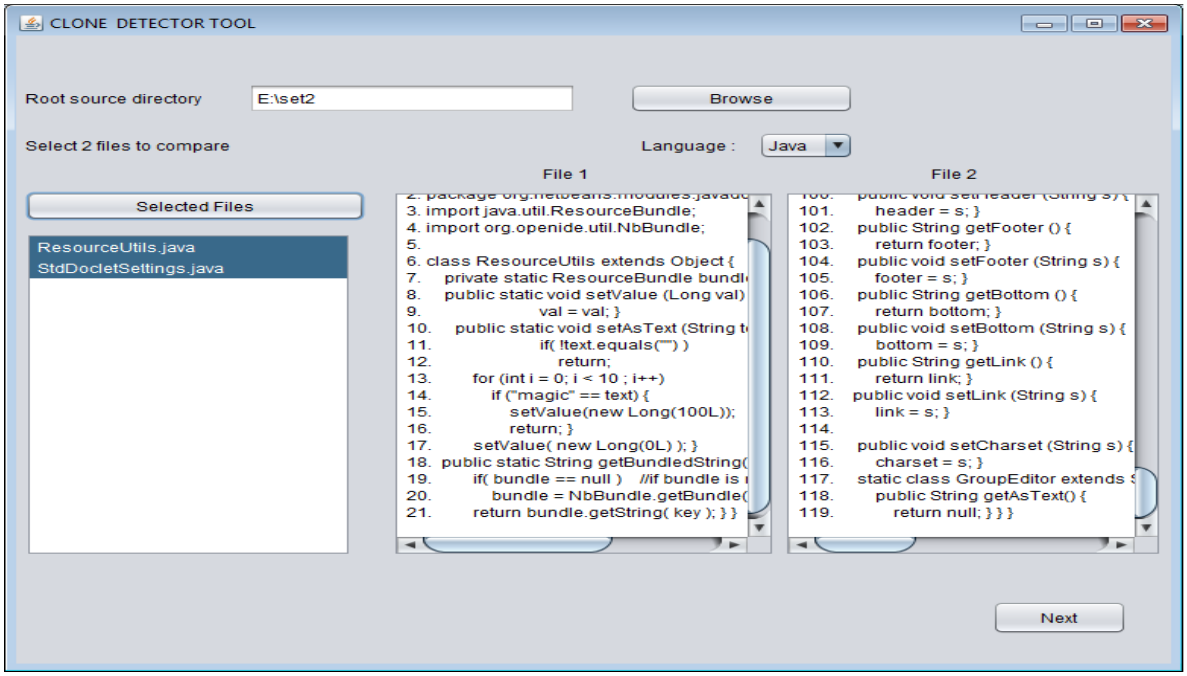

Figure 4.5 Displaying Source Files

By pressing the selected file button, source files will display in the same window, click the next button for the further process.

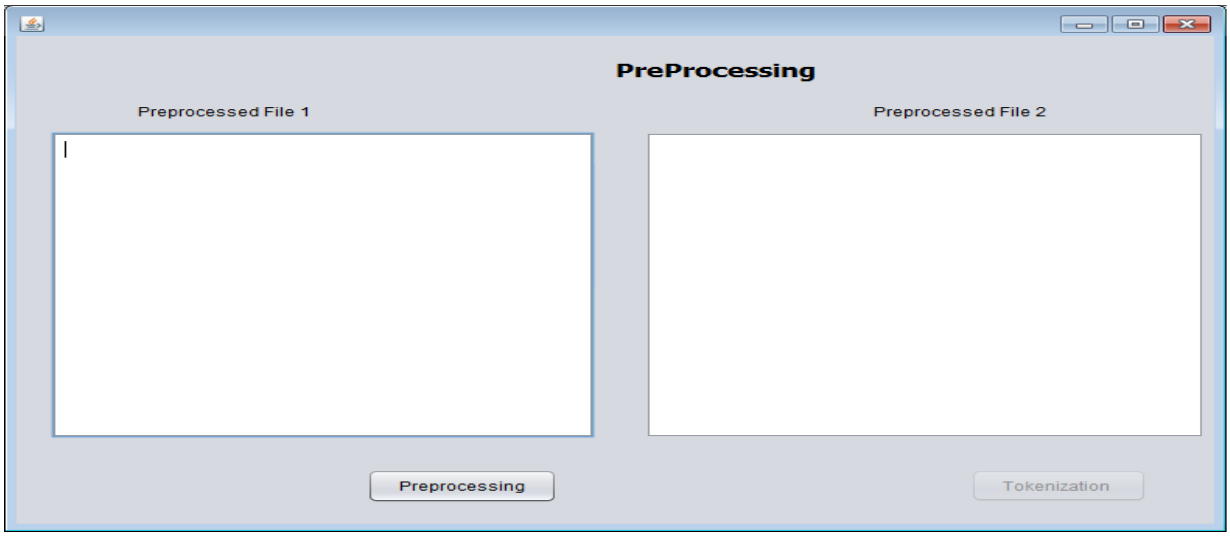

Figure 4.6 Preprocessing window

This is the preprocessing window view.

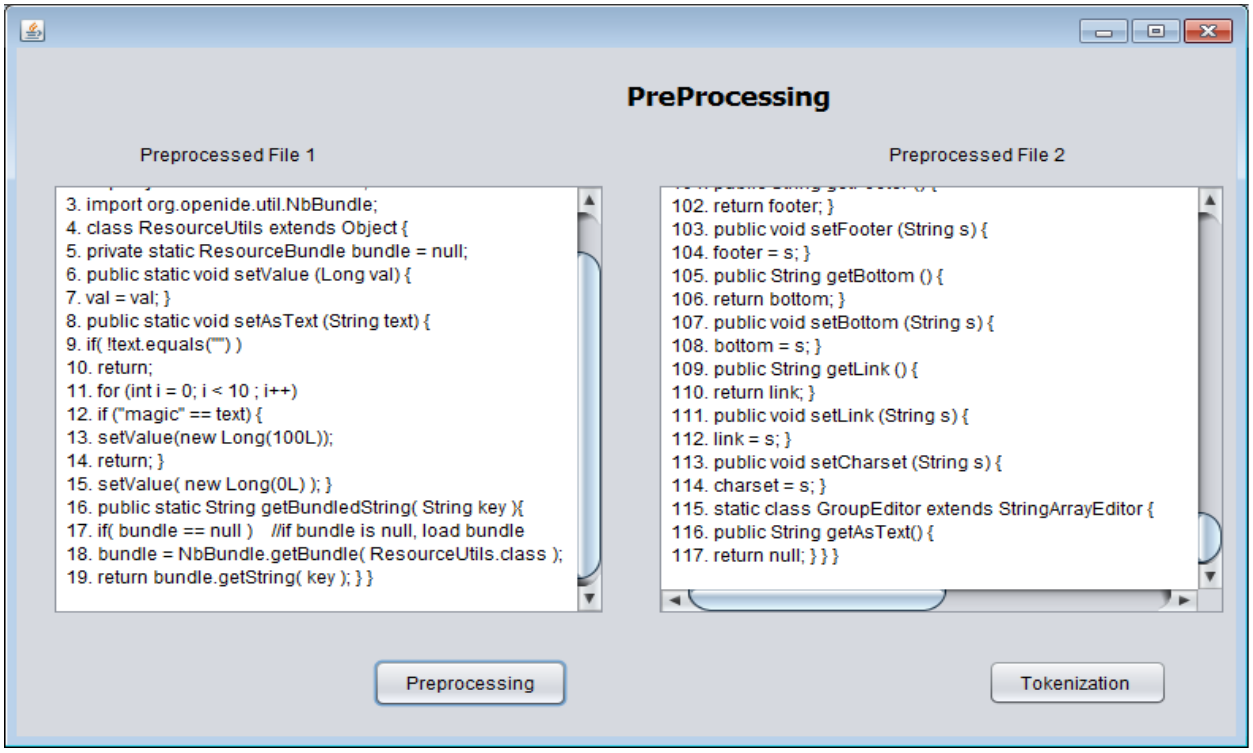

Figure 4.7 Preprocessed files

By clicking on preprocessing button; white-spaces, comments will be removed and after that press tokenization button to start tokenization technique.

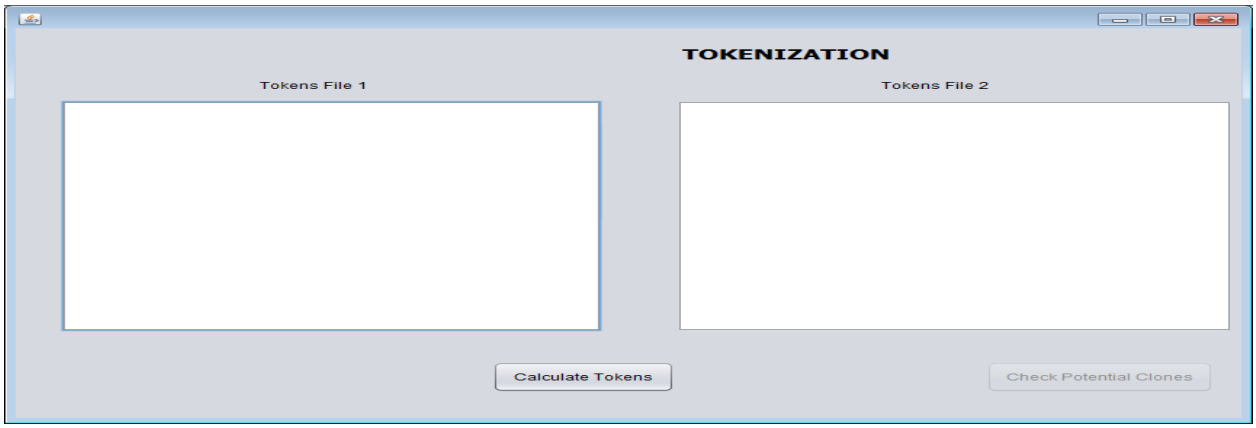

Figure 4.8 Tokenization window

This is the Tokenization window view.

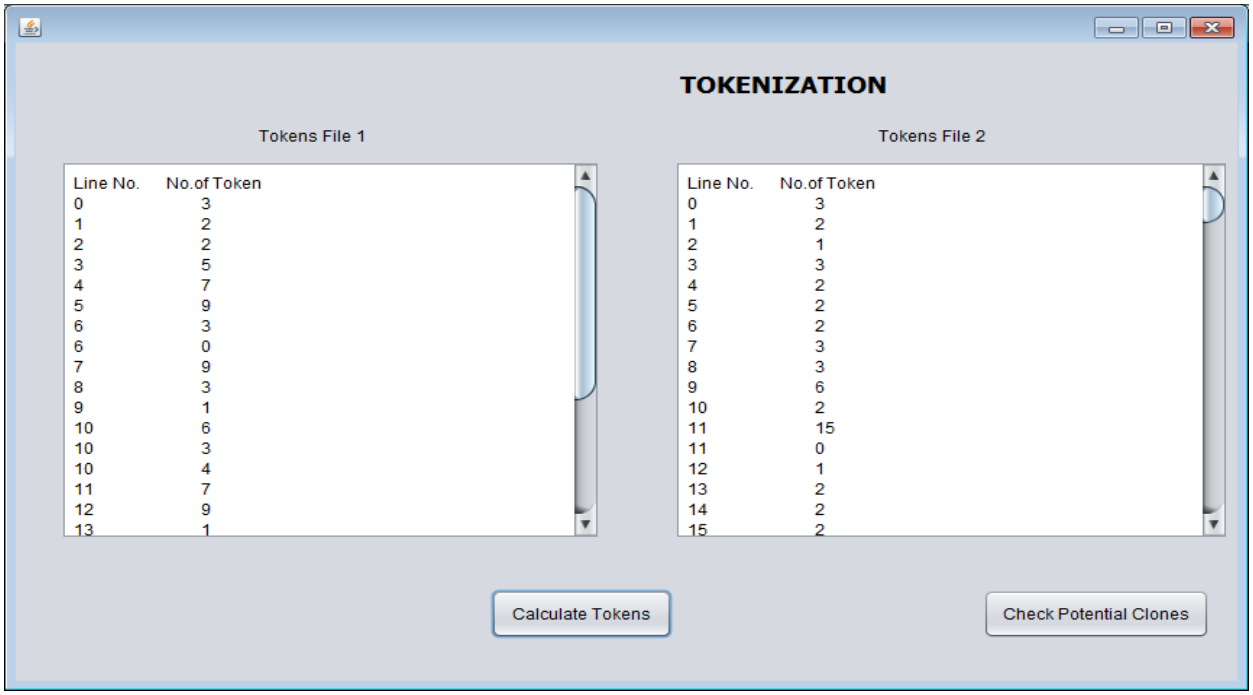

Figure 4.9 Caulate Tokens

Click on the calculate tokens button; token of file 1 and file 2 along with their line numbers will generate,after that click on check potential clones button.

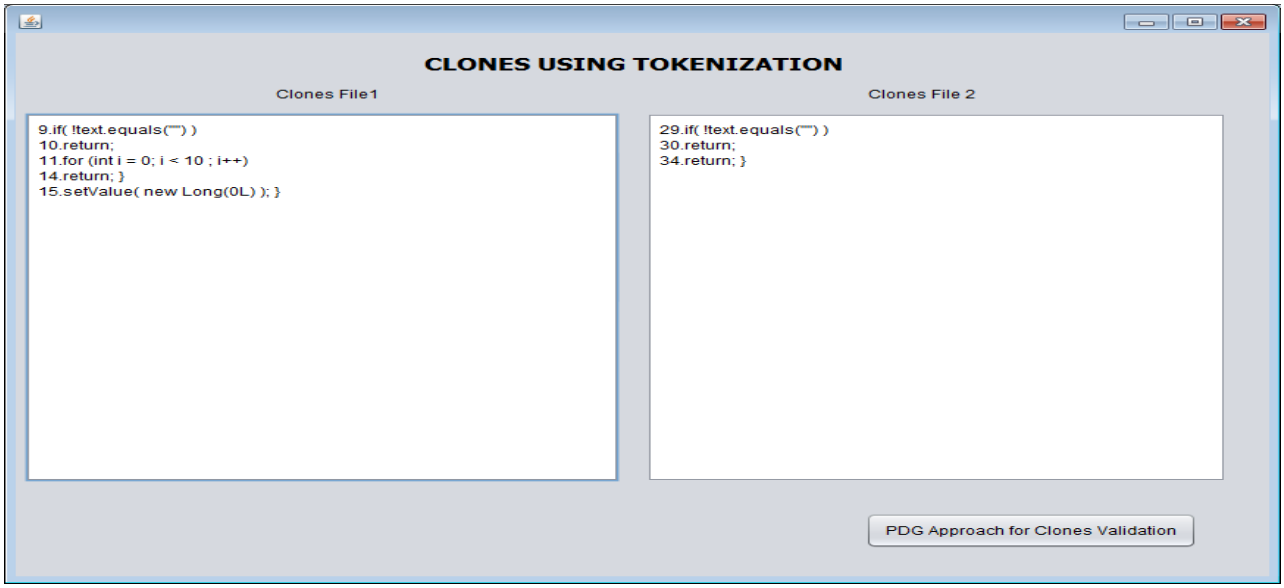

Figure 4.10 Potential clones

File 1 and file 2 potential clones are generated using tokenization approach, after that click on PDG approach for validating the potential clones.

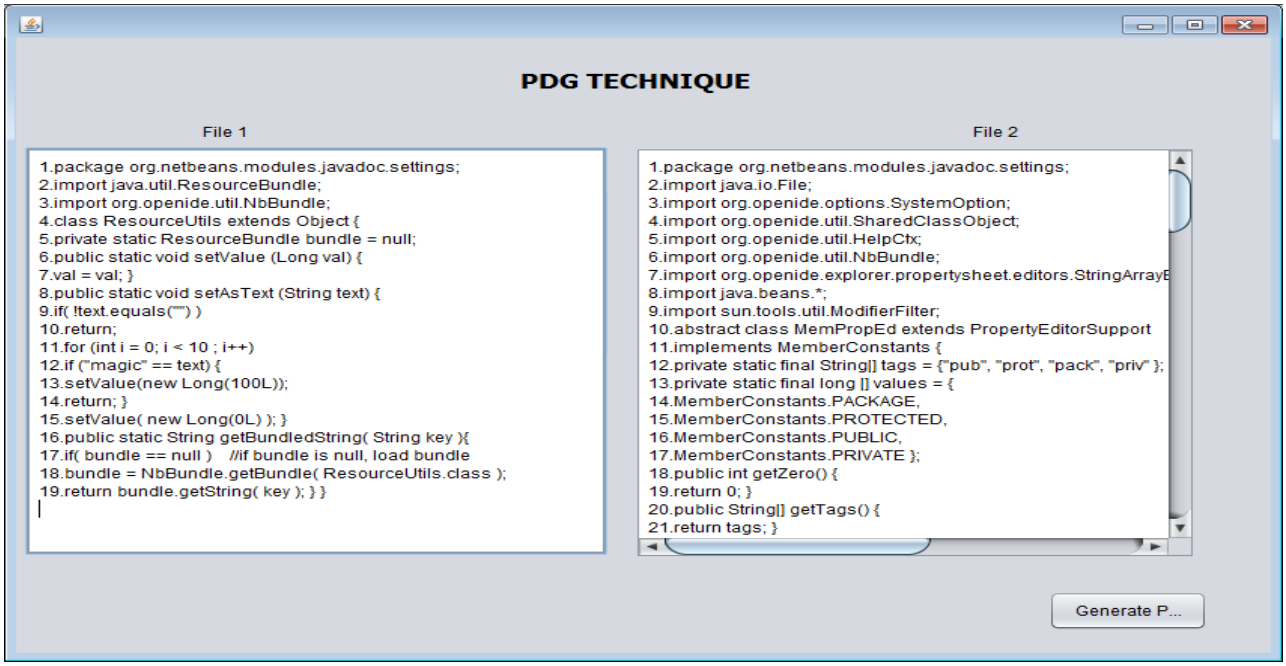

Figure 4.11 Preprocessed files for PDG

In this window pain again preprocessed files are fetched to generate PDG

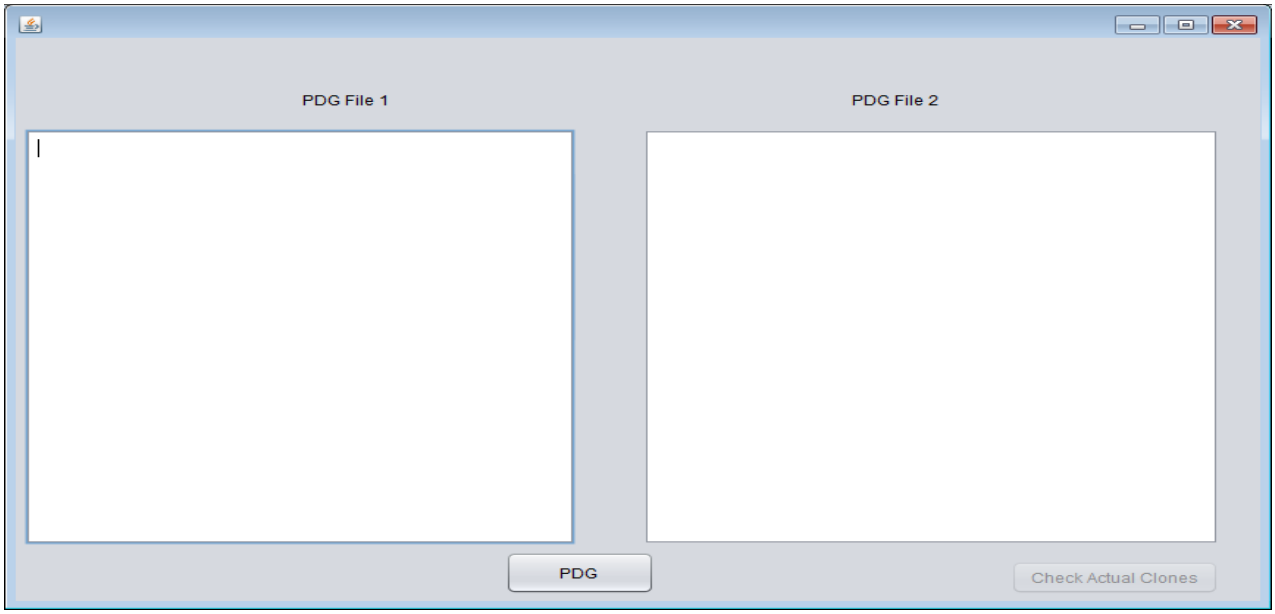

Figure 4.12 PDG Window

This view of the window is for Program dependency graph technique.

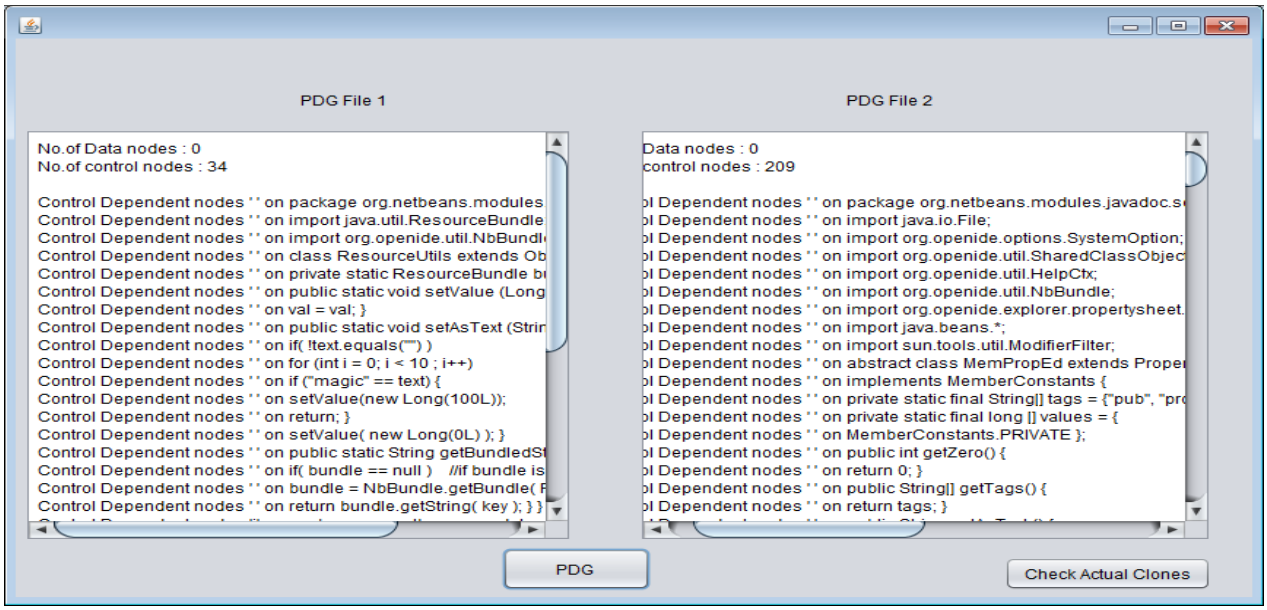

Figure 4.13 Generated PDG

This window panel will display data nodes and control nodes generated by the PDG technique for file 1 and file 2 and it will check whether logical similarity exist between two files,and then press check actual clone button.

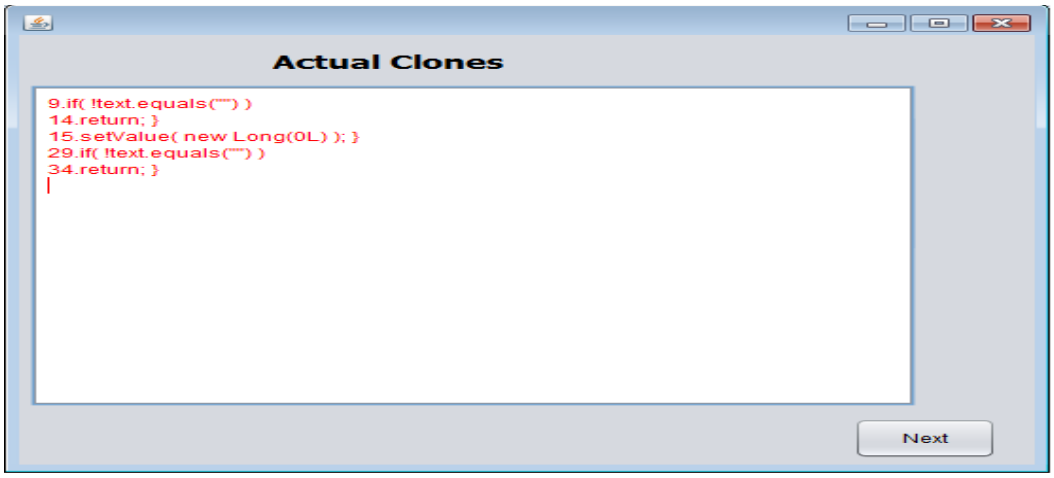

Figure 4.14 Actual Clones

This window panel display result of actual clones of validating potential clones which are generated by tokenization technique, Smith Waterman algorithm is working for comparison.

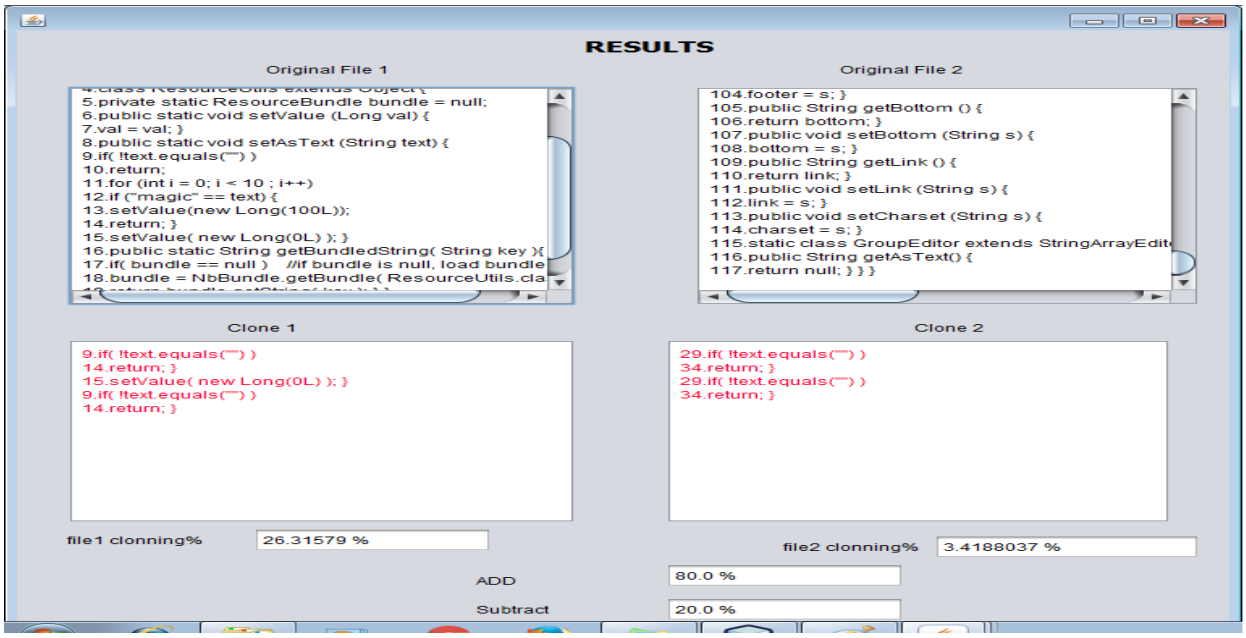

Figure 4.15 Results

On result window, original files and clones of respective files are displayed with the cloning percentage of the respective files and add, subtract percentage of lines are also displayed of clone groups.

4.1 RESULTS OF TRIALS

We have done trials with the existing and proposed system to recognize the clones. The aftereffects of both methodologies have been tried on two Java source code records and furthermore on the C++ code documents and their resultants are appearing in table 9 and table 10.

The outcomes demonstrate that our proposed approach is superior to anything the current one as far as parameters precision rate, recall, and accuracy rate values which are gotten by utilizing the equation (7), (8) and (9) examined in the next segment. In Table 9 and Table 10:

- **(TP)**is a contraction for genuine positive, i.e. these are the real clones which are distinguished by the device.
- **(TN)** is a contraction for genuine negative, i.e. these are the genuine clones which are not distinguished by the apparatus.
- **(FP)**is a truncation for false positive, i.e. these are not the genuine clones, but rather are recognized as clones by the instrument.
- **(FN)** is a shortened form for false negative, i.e. these are not the real clones and furthermore the instrument didn't recognize these.
- **(P)** is the entirety of TP and FN.
- **(N)** is the total of FP and TN.

We have done trials on sixty files from which forty files are actual cloned files and twenty are non cloned files.Some files are taken from Bellon’s data set which is for Java files and others are sample files for testing the results of existing and proposed techniques.

Table 4.1 Results of existing technique

Existing technique	Predicted Negative	Predicted Positive
Negative Cases	TN:16	FP: 4
Positive Cases	FN: 10	TP: 30

Table 4.2 Results of proposed technique

Proposed technique	**Predicted Negative**	**Predicted Positive**
Negative Cases	TN: 16	FP: 4
Positive Cases	FN: 6	TP: 34

4.2 Execution Measures

For clone identification, the parameter precision, recall and accuracy are gotten utilizing the conditions given beneath[26]:

$$\text{Precision} = {}^{TP}/_{TP + FP} \quad (7)$$

$$\text{Recall} = {}^{TP}/_{P} \quad (8)$$

$$\text{Accuracy} = {}^{TP + TN}/_{P + N} \quad (9)$$

Utilizing the over four conditions we have looked at the execution of our proposed approach and existing methodology in view of table 9 and table 10. The got results are appearing in table 11.

Table 4.3 Results comparison

Parameters to compare	Existing Approach	Proposed Approach
Precision	0.88	0.89
Recall	0.75	0.85
Accuracy	0.76	0.83

Summary

We have done trials withbellon data set and on some sample files to calculate the precision, recall and accuracy values. We haveseen that our tool is detecting morenumber of clones as compared to the existing technique, so cloning percentage calculated by our clone detector tool is more. Also addition i.e., no less than one clone section in new Group is recently included and subtraction i.e.,no less than one clone section in old group has changed or expelled, in this manner it doesn't show up in new group percentage from one clone group [30]than the other clone group is also being detected by our tool.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSION

This thesis report puts a light on all the types of clones and various techniques for the detection of clones. We have also presented the reasons of cloning along with its pros and cons and the process involved in detection of clones.We have presented a hybrid technique that recognizes software code clones for Java and c++ codes on the basis of token and program dependency graph based approaches.

The Program dependency graph procedure is utilized to discover potential clones in the framework while measurements based strategy is utilized to check them as genuine clones. As PDG conveys semantic data of framework, thus proposed device can distinguish both syntactic and semantic comparative code clones. The proposed device has discovered code clones just for projects written in object oriented languages. This device experiences five stages amid its clone discovery life cycle.

The proposed approach distinguishes potential clones on the token-based match. Potential clones are additionally contrasted utilizing a PDG-based approach with check whether the potential clones identified utilizing token based examinations are really clones or not. We have actualized the current and proposed systems as an instrument written in Java. In light of the outcomes from this apparatus, we have watched that our proposed technique is superior to existing one as far as parameters, for example, precision, recall, accuracy,,0.88, 0.89, ,,0.75, 0.85, ,,0.76, 0.83 individually to exist and proposed strategy.

5.2 FUTURE SCOPE

Since the last decade, there has been a wonderful contribution of numerous researchers in the field of software cloning. This field has still a lot of scope for new researchers to work upon code clone genealogies, investigating potential clones from the actual clones, detecting type 4(Semantic) clones with more accuracy and precision, refactoring of clones and of course the maintenance of a project which is the most costly phase of SDLC.

Our method is fit for all types of clones, and the programming language is only fit for ava and c++. However, the method is extended to another programming language and detector types.Productivity of tool can be enhanced for type IV clones where the rearrangement of control and data dependent proclamation is related. This tool can be additionally upgraded by utilizing clones evacuation strategies subsequent to recognizing real clones.

REFERENCES

[1] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," *Inf. Softw. Technol.*, vol. 55, no. 7, pp. 1165–1199, 2013.

[2] I. D. Baxter, A. Yahin, L. Moura, M. S. Anna, L. Bier, and S. Drive, "Clone Detection Using Abstract Syntax Trees," 1998.

[3] S. Ducasse, M. Rieger, and S. Demeyer, "A language independent approach for detecting duplicated code," *Proc. IEEE Int. Conf. Softw. Maint. - 1999 (ICSM'99). 'Software Maint. Bus. Chang. (Cat. No.99CB36360)*, no. c, pp. 109–118, 1999.

[4] R. Komondoor and S. Horwitz, "Using Slicing to Identify Duplication in Source Code," *SAS '01 Proc. 8th Int. Symp. Static Anal.*, vol. 2126, pp. 40–56, 2001.

[5] J. Mayrand, C. Leblanc, and E. M. Merlo, "Experiment on the automatic detection of function clones in a\nsoftware system using metrics," *Softw. Maint. 1996, Proceedings., Int. Conf.*, pp. 244–253, 1996.

[6] 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, no. 7, pp. 470–495, 2009.

[7] R. Tairas and J. Gray, "Increasing clone maintenance support by unifying clone detection and refactoring activities," *Inf. Softw. Technol.*, vol. 54, no. 12, pp. 1297–1307, 2012.

[8] R. K.Saha, C. K. Roy, and K. A. Schneider, "Department of Computer Sience,University of Saskatchewan,Canada{ ripon.saha, chanchal.roy, kevin.schneider}@usask.ca," in *An automatic Framework for Extracting and Classifying Near-Miss Clone Genealogies*, 2011, pp. 293–302.

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

[10] Y. Chen and C. K. Roy, "Near-miss Software Clones in Open Source Games : An Empirical Study," pp. 1–7, 2014.

[11] M. Balazinska, E. Merlo, M. Dagenais, B. Lagiie, and K. Kontogiannis, "Advanced clone-analysis to support object-oriented system refact oring," pp. 98–107, 2000.

[12] A. Marcus and J. I. Maletic, “Identification of high-level concept clones in source code,” *Proc. 16th Annu. Int. Conf. Autom. Softw. Eng. (ASE 2001)*, no. 0, pp. 107–114, 2001.

[13] M. Sarkar, T. Mondal, S. Roy, and N. Mukherjee, “Resource requirement prediction using clone detection technique,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 4, pp. 936–952, 2013.

[14] B. S. Baker, “A Program for Identifying Duplicated Code,” *Comput. Sci. Stat.*, vol. 24, pp. 49–57, 1992.

[15] 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.*, vol. 28, no. 7, pp. 654–670, 2002.

[16] J. Krinke, “Identifying Similar Code with Program Dependence Graphs,” 2001.

[17] M. Gabel, “Scalable Detection of Semantic Clones,” pp. 321–330, 2008.

[18] C. Liu, C. Chen, J. Han, and P. S. Yu, “GPLAG: detection of software plagiarism by program dependence graph analysis,” *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discov. data Min.*, pp. 872–881, 2006.

[19] M. Weiser, “Program slicing,” *Proc. 5th Int. Conf. Softw. Eng.*, pp. 439–449, 1981.

[20] L. Jiang, G. Misherghi, and Z. Su, “D ECKARD : Scalable and Accurate Tree-based Detection of Code Clones ∗,” no. 0520320, 2007.

[21] S. Lu, Z. Li, F. Qin, L. Tan, P. Zhou, and Y. Zhou, “BugBench: Benchmarks for Evaluating Bug Detection Tools,” *Proc Work. Eval. Softw. Defect Detect. Tools*, no. 3, pp. 1–5, 2005.

[22] C. K. Roy, “Detection and Analysis of Near-Miss Software Clones,” pp. 447–450, 2009.

[23] G. M. K. Selim, K. C. Foo, and Y. Zou, “Enhancing source-based clone detection using intermediate representation,” *Proc. - Work. Conf. Reverse Eng. WCRE*, pp. 227–236, 2010.

[24] A. Sheneamer and J. Kalita, “Code clone detection using coarse and fine-grained hybrid approaches,” *2015 IEEE 7th Int. Conf. Intell. Comput. Inf. Syst. ICICIS 2015*, pp. 472–480, 2016.

[25] M. Sarkar, S. Chudamani, S. Roy, and N. Mukherjee, “A hybrid clone detection

technique for estimation of resource requirements of a job," *Int. Conf. Adv. Comput. Commun. Technol. ACCT*, pp. 174–181, 2013.

[26] J. Bhagwan and K. Pramila, "Design and Analysis of a Hybrid Technique for Code Clone Detection," vol. 5, no. 11, pp. 380–385, 2016.

[27] 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.

[28] M. S. Waterman, T. F. Smith, and W. A. Beyer, "Some biological sequence metrics," *Adv. Math. (N. Y).*, vol. 20, no. 3, pp. 367–387, 1976.

[29] 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.

[30] C. H. Wang, Y. Tu, L. P. Zhang, and D. S. Liu, "Extracting clone genealogies for tracking code clone changes," *Int. J. Secur. its Appl.*, vol. 10, no. 3, pp. 21–30, 2016.

APPENDIX

Abbreviations

AST : Abstract Syntax Tree

FP: False Positive

FN: False Negative

PDG: Program Dependency Graph

SDLC: Software Development Lifecycle

TP: True Positive

TN: True Negative