ATO ENHANCE THE CODE CLONE DETECTION ALGORTHIM BY USING HYBRID APPROACH FOR DETECTION OF CODE CLONES.

Dissertation submitted in fulfilment of the requirements for the Degree of

MASTER OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

By

ROOPAM

11506892

Supervisor

MR. GUPREET SINGH

School of Computer Science and Engineering

Lovely Professional University

Phagwara, Punjab (India)

May, 2017

May, 2017

LOVELY PROFESSIONAL UNIVERSITY
Transforming Education, Transforming India

TOPIC APPROVAL PERFORMA

School of Computer Science and Engineering

Program : P172::M. Tech. (Computer Science and Engineering) [Full Time]

COURSE CODE : CSE545 **REGULAR/BACKLOG :** Regular **GROUP NUMBER :** CSERGD0006

Supervisor Name : Gurpreet Singh **UID :** 16523 **Designation :** Assistant Professor

Qualification : ME Soft. Engg. **Research Experience :** 5 years

SR.NO.	NAME OF STUDENT	REGISTRATION NO	BATCH	SECTION	CONTACT NUMBER
1	Roopam	11506892	2015	K1519	9872883299

SPECIALIZATION AREA : Networking and Security **Supervisor Signature:** _____

PROPOSED TOPIC : A novel approach for code clone detection and removal

Qualitative Assessment of Proposed Topic by PAC		
Sr.No.	**Parameter**	**Rating (out of 10)**
1	Project Novelty: Potential of the project to create new knowledge	7.40
2	Project Feasibility: Project can be timely carried out in-house with low-cost and available resources in the University by the students.	6.60
3	Project Academic Inputs: Project topic is relevant and makes extensive use of academic inputs in UG program and serves as a culminating effort for core study area of the degree program.	6.60
4	Project Supervision: Project supervisor's is technically competent to guide students, resolve any issues, and impart necessary skills.	7.20
5	Social Applicability: Project work intends to solve a practical problem.	6.80
6	Future Scope: Project has potential to become basis of future research work, publication or patent.	7.00

PAC Committee Members		
PAC Member 1 Name: Prateek Agrawal	UID: 13714	Recommended (Y/N): Yes
PAC Member 2 Name: Pushpendra Kumar Pateriya	UID: 14623	Recommended (Y/N): Yes
PAC Member 3 Name: Deepak Prashar	UID: 13897	Recommended (Y/N): Yes
PAC Member 4 Name: Kewal Krishan	UID: 11179	Recommended (Y/N): Yes
PAC Member 5 Name: Dr. Ashish Kumar	UID: 19584	Recommended (Y/N): Yes
DAA Nominee Name: Kanwar Preet Singh	UID: 15367	Recommended (Y/N): NA

Final Topic Approved by PAC: **A novel approach for code clone detection and removal**

Overall Remarks: Approved

PAC CHAIRPERSON Name: 11011::Rajeev Sobti **Approval Date:** 26 Oct 2016

ABSTRACT

Code clones are easy and quick way to add some existing logic from one section to another section. Code clones are different fragments of code that are very similar. Clone is a persistent form of software reuse that effects on maintenance of large software. In previous research, the researchers emphasize on detecting type 1, type 2, and type 3 and type 4 types of clones. The existing code clone detection techniques like text based, token based, abstract syntax tree, program dependency graph and metric based are used to detect clone in source code. In this research, the enhancement in code clone detection algorithm has been proposed which detects code clones by HYBRID algorithm that is combination of program dependency graph and Metric based clone detection techniques. In this work, firstly implementation of code clone detection will be done by hybrid approach on various datasets. Then, comparison of existing technique will be done with the hybrid technique in terms of achieving enhancement in performance, efficiency and accuracy in results. This method is considered to be the least complex and is to provide a most accurate and efficient way of Clone Detection. The results obtained have been compared with an existing tool on various datasets.

DECLARATION

I am Roopam hereby declare that the research work reported in the dissertation entitled "TO ENHANCE THE CODE CLONE DETECTION ALGORTHIM BY USING HYBRID APPROACH FOR DETECTION OF CODE CLONES" 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 Mr. Gurpreet Singh I have not submitted this work elsewhere for any degree or diploma.

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

Signature of Candidate

Roopam

11506892

SUPERVISOR'S CERTIFICATE

This is to certify that the work reported in the M.Tech Dissertation entitled "**TO ENHANCE THE CODE CLONE DETECTION ALGORTHIM BY USING HYBRID APPROACH FOR DETECTION OF CODE CLONES"**, submitted by **Roopam** 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

Mr. Gurpreet Singh

Date:

Counter Signed by:

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

   HoD Name: _____

   Date: _____
2) **Neutral Examiners:**

   External Examiner

   Signature: _____

   Name: _____

   Affiliation: _____

   Date: _____

   Internal Examiner

   Signature: _____

   Name: _____

   Date: _____

ACKNOWLEDGEMENT

I would like to express my special thanks to God to give me this opportunity of writing this thesis and providing such nice peoples who was there always to help me in my dissertation. Secondly, big thanks goes to my mentor "Gurpreet Singh" who gave me this topic "***To Enhance Code Clone detection using Hybrid approach for detection of code clones"*** for my dissertation work, I am heartily thankful to Gurpreet Sir for being my mentor and helped me in doing a lot of Research and I came to know about so many new things. Very special thanks to all the authors whose paper I referred for this dissertation. Their effort made me to think about new ideas and due to what I am able to implement them in my research. "Source: Internet" gave me so many short definitions that's included here.

At last, I would also like to thank my parents and friends who helped me a lot in my research within the limited time frame. This is just 40% completion of my work, I hope my mentor, parents and friends will help me till my thesis is under working.

Roopam

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Checklist for Dissertation-III Supervisor

Name: _____ UID: _____ Domain: _____
Registration No: _____Name of student:_____

Title of Dissertation:

_____

- ☐ Front pages are as per the format.
- ☐ Topic on the PAC form and title page are same.
- ☐ Front page numbers are in roman and for report, it is like 1, 2, 3…….
- ☐ TOC, List of Figures, etc. are matching with the actual page numbers in the report.
- ☐ Font, Font Size, Margins, line Spacing, Alignment, etc. are as per the guidelines.
- ☐ Color prints are used for images and implementation snapshots.
- ☐ Captions and citations are provided for all the figures, tables etc. and are numbered and center aligned.
- ☐ All the equations used in the report are numbered.
- ☐ Citations are provided for all the references.
- ☐ **Objectives are clearly defined.**
- ☐ Minimum total number of pages of report is 50.
- ☐ Minimum references in report are 30.

Here by, I declare that I had verified the above mentioned points in the final dissertation report.

Signature of Supervisor with UID

CHAPTER 1

INTRODUCTION

Software engineering is all about building, evolving and maintaining software systems. It is a set of problem solving skills, techniques, technology and methods applied upon a variety of domains to evolve and create useful systems that solve many problems like practical problems. Software Engineering is the practice of computer science which applies engineering fundamentals to build, accomplish, customize and maintain of software components. Software engineer is required to handle software engineering projects which discover, create, build software and tells its behavior. System engineering is different from software engineering. System engineering is concern with deployment, architectural design and integration where as software engineering is concern with development, quality and testing and control of the system.

The main goals of software engineering are as follows.

- To produce high quality software with less cost.
- To achieve higher accuracy.
- To achieve reliability.
- To improve efficiency.

1.1 SOFTWARE TESTING:

This technique is to find out error or faults in a system to make it correctness, completeness and to identify the quality of existing software. It is an internal part of software development and closely related to software quality. The main aim of software testing is to fulfill user's requirements and make the system error free. So software testing is mainly to find outs the error or bugs to raise the quality of the system. This technique is used to catch the bugs and uncover it. Software testing is a process and discipline also. It is different from software development. It should be considered that is part of software development.

1.1.1 Types of Testing:

These testing are as follow:-

- **Black-Box Testing**: It is also known as functional testing. It is a testing which is based on the output and does not require any knowledge of internal structure of a program. This testing ignores the internal mechanism of the system. It is a testing in which it's working or process is not understood by its user. It has no knowledge of processing or working of code but only concentrate upon the output. It is used in the validation process. It is based on the functionality and specification. Sometimes it is also known behavioral testing. It has two techniques.
  - Equivalence class portioning
  - Boundary value analysis.
- **White-Box Testing:** This testing also called as open box testing or clear box testing and structural testing. In this, testing code is visible. It has knowledge of the internal mechanism of the components. It is opposite to black box testing. White-box testers are aware about the internal structure and also know how code is looks like. It is used in the validation process. It is a clear box testing because code can be easily visible in this type of testing. It has many techniques.
  - Branch coverage.
  - Statement testing.
  - Path coverage.
  - Condition coverage.
- **Clone Testing:** To assure the quality of the product is the main target of software engineering. It detects the faults and prevents the system from faults by analysis or testing. At the time of developing any software for saving effort and time, software developer might copy paste program code again and again in different places. So if one section has fault, it will be reproduced in every section. There are many copies of code present but no record of such copies is present. This will make hard to prevent such faults and maintenance of existing software. By concluded that the clone result comes from adding some extra functionality, which is same but not identical to existing logic. Code cloning is close to a process in which some parts of code traced and then pasting it with or without some slight modifications in other section of the code so that we can reuse the section of the code. The fixed or pasted code segment is known as code clone. During the development phase of software code cloning is very common. As per present research

study 8% to 22% of code in a software system is cloned code. For the maintenance stage of the software it is tricky because if the cloned data consists of faults, then it should need to identify and correct the same fault from the clones of that code. It will increase the cost of development software and lead to poor quality of software system as it results in maximizing in software size [10]. If more than two code sections in a software system's code-base are closely similar or exactly similar to each other then we call them as code clones [7]. As far as removal of duplicated code is concerned, the art proposes refactoring technique which is a method to gradually raise the structure of programs while preserving their external behavior [11].

1.2 TYPES OF CODE CLONES:

Code clones are of four types:

- **Type 1**: It is also called exact or same copy code clone. These code clones can be detect by every technique and they are easy to detect. These code clones are identical in nature. Type 1 clones deals with white spaces and comments.[6]

Table 1.1: Type 1

int addn(int num[],int x){
int A=0;//addn
for(int m=0;m<x;m++){
A=A+num[m];
}
return A;
}
	int addn(int num[], int x){
	int A=0;
	for(int m=0;m<x;m++){
	A=A+num[m];
	}
	return A;
	}
		int addn(int num[],int x){
		int A=0;//addn
		for(int m=0;m<x;m++){
		A=A+num[m];
		}
		return A;
		}

- **Type-2**: These are code clones which are symmetrically and syntactically very same. Literals are changed. E.g. Name of variables and functions. It is difficult to detect from type1.these code clones can be detected by token based, abstract syntax tree, program dependency graph, metric based and hybrid approach. they can not be detect by text based approach.[6]

Table 1.2: Type 2

int additionn (int num[], int n){
int sumx=0;//sumx
for(int i=0;i<n;i++){
sumx=sumx+num[i];
}
return sumx;
}
	int doaddn(int no[], int n){
	int S=0;
	for(int i=0;i<n;i++){
	S+=no[i];
	}
	return S;
	}
		int sumX(int a[],int n){
		int P=0;//sumX
		for(int i=0;i<n;){
		P=P+a[i];
		i++;
		}
		return P;
		}

- **Type-3**: These code clones are copied fragment by deleting or adding and interchanged lines. Type 3 code clones are detected by PDG, AST, MTB and Hybrid approach [1].

Table 1.3: Type 3

int addn(int num[],int x){
int A=0;//addn
for(int m=0;m<x;m++){
A=A+num[m]; }
return A;
}
	int addn(int num[], int x){
	int A=0;
	for(int m=0;m<x;m++){
	A=A+num[m];
	}
	return A;
	}
		int addn(int num[],int x){
		int A=0;//addn
		for(int m=0;m<x;m++){
		A=A+num[m];
		}
		return A;
		}

- **Type-4**- These code clones are different in syntax but functionality and logical is same. This code clones are very difficult to detect. It is code clone which is not created intentionally .they are only detected by PDG and Hybrid approach. [9]

Table 1.4: Type 4

int addn(int no[],int n){
int sumx=0;
for(int p=0;p<n;p++){
sumx=sumx+no[i];
}
return sumx;
}
	int addn(int no[],int n){
	if(n==1)
	return no[n-1];
	else
	return no[n-1]+addn[no,n-1];
	}

1.3 TECHNIQUES FOR CODE CLONES:

Code clone techniques are basically of four types:

- **Textual approach**- In this line to line measure is done, it is based on two types one is simple line matching and other is parameterized line matching. [7] This technique is string based.
- **Lexical approach:** - In this we modify source code into tokens using lexical rules. These tokens are matched with each other.
- **Syntactic approach:** - In this an abstract tree is developed. Using parser source code is change into parse tree.
- **Semantic approach:** - In this approach, source code is served as program dependency graph. Nodes define the statements and expressions and, edges define control and data dependencies.

1.4 CODE CLONE PROCESS:

1.4.1 PROCESSING PHASE:

This phase follows two steps- one is dividing the source code into the sections also known as segmentation. Secondly, figure out the area of comparison. There are certain objectives of this phase [7]:

- **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, classes, functions/methods, files, 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.

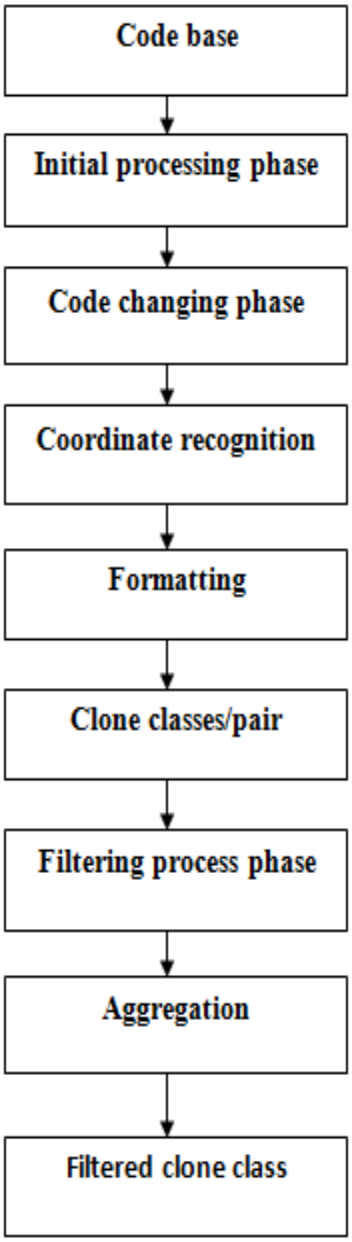

Figure 1.1: Code Clone process

1.4.2 TRANSFORMATION PHASE: 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 in tokens.

- **Parsing**: To indicate clones in syntactic approach, abstract syntax tree is used to compare algorithms for same sub-trees. Metric-based approach can also be used.

1.4.3 COORDINATE RECOGNITION: 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.

1.4.4 FORMATTING: The clone pair list for the changed code acquired by the comparison algorithm is transformed over to a relating clone pair list for the original code base.

1.4.5 FILTERING PROCESS PHASE: This step is further subdivided into two parts.

1.4.6 MANUAL ANALYSIS: Here false positives are filtered out by human experts.

1.4.7 AUTOMATED HEURISTIC: Few parameters are already set according to filtering purposes. For example: length, frequency, diversity etc.

1.4.8 AGGREGATION: With an end goal to expel the information, perform ensuing examination or accumulate outline measurements, clones might be collected into clone classes.

1.5 VARIOUS CODE CLONE DETECTION TECHNIQUES:

The code clone detection techniques in software cloning are [8]:

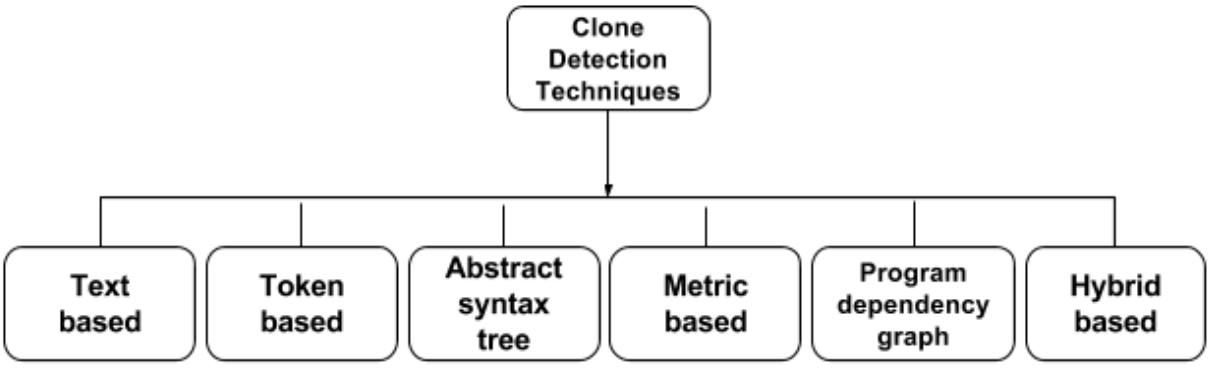

Figure 1.2: Code clone detection techniques

- **Text based-** it needs minimum transformation. In this portion of code is considered as sequences of strings and then these are co-related with each other in order to find the same string. It is also called as string based approach. Line by line comparison will be performed on the two code fragments. [6] If textual similarity exists between them, then they are counted as clones.

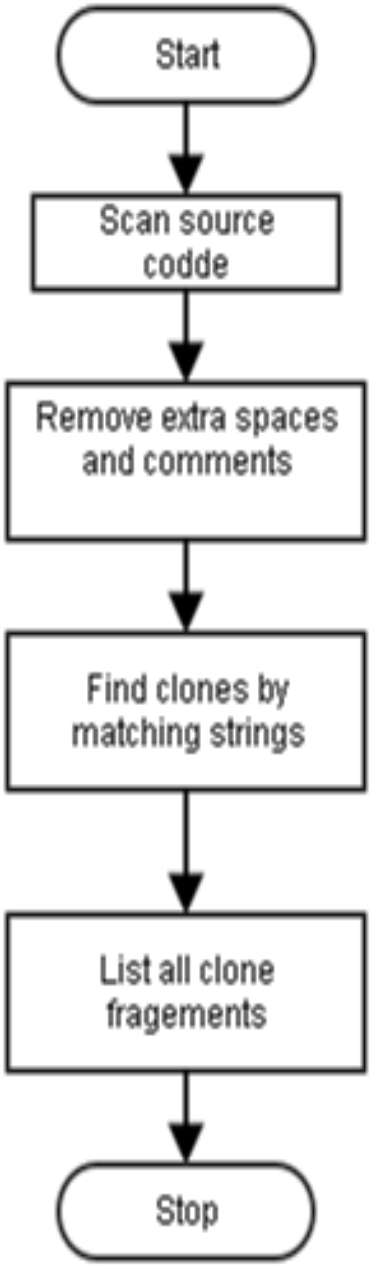

Figure 1.3: Text based

- **Token based-** it is also known as lexical approach. This approach uses parser or lexer for the transformation of source code into a sequence of tokens. This approach is more efficient than text based approach. Parameterized matching with suffix tress one of the technique of token based approach.[10]

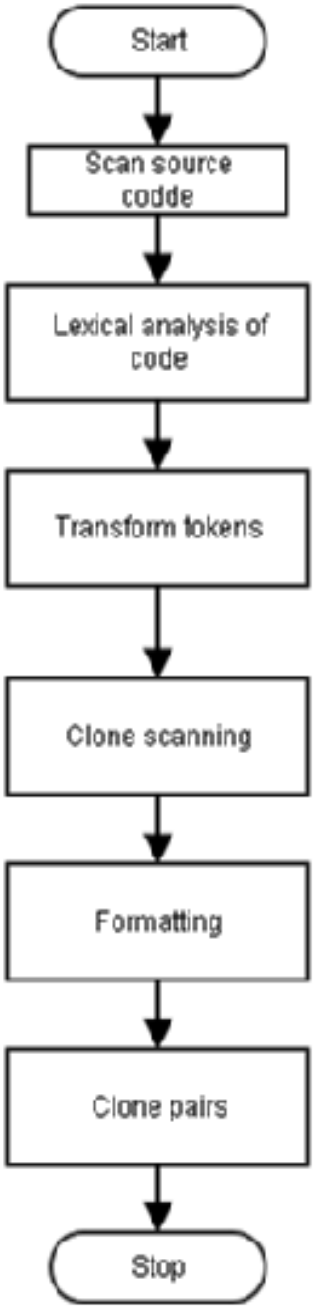

Figure 1.4: Token based

- **Abstract syntax tree based approach**- in this, code clones are searched by searching for same or similar sub trees which denotes presence of code clone. The level of accuracy [7] is best but it results in unstable scalability as it depends on the algorithm that is being used to make (build) and correlate of the trees.

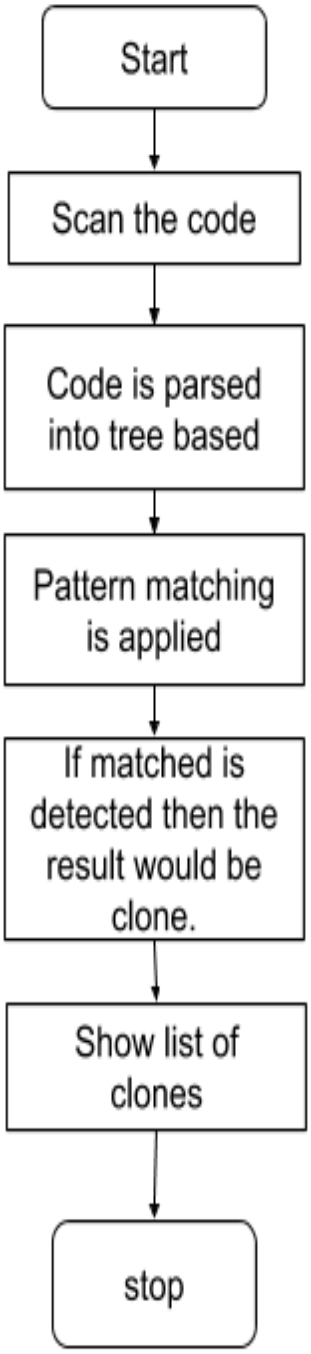

Figure 1.5: Abstract syntax tree

- **Metric based-** metric are used to measure clones in software after the calculations of metrics from source code. [3] For the calculations of metric this approach parses source code to its AST/PDG representation. This approach provides high accuracy and scalability level. It helps us to detect type 3 code clones.

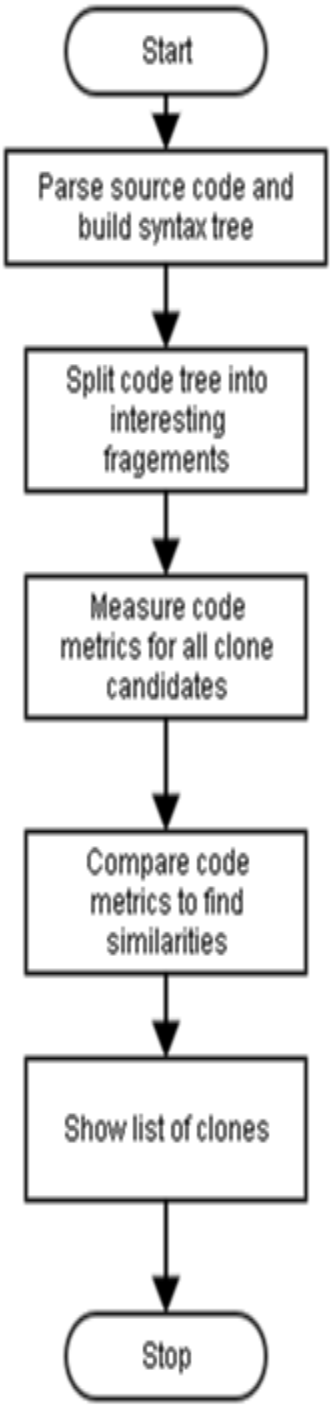

Figure 1.6: Metric based

- **Program dependency graph**- this approach emphasis on data dependency and control flow. After execute PDG algorithm is used to retrieve code clones. This helps in the finding of the clones. The dependency graphs needed to be make for this approach and the accuracy of these graphs have to be extremely taken care, PDG based detection approach is very effective as it can detect non-contiguous code clones. [4] But it is costly process to obtain PDG for large software's.

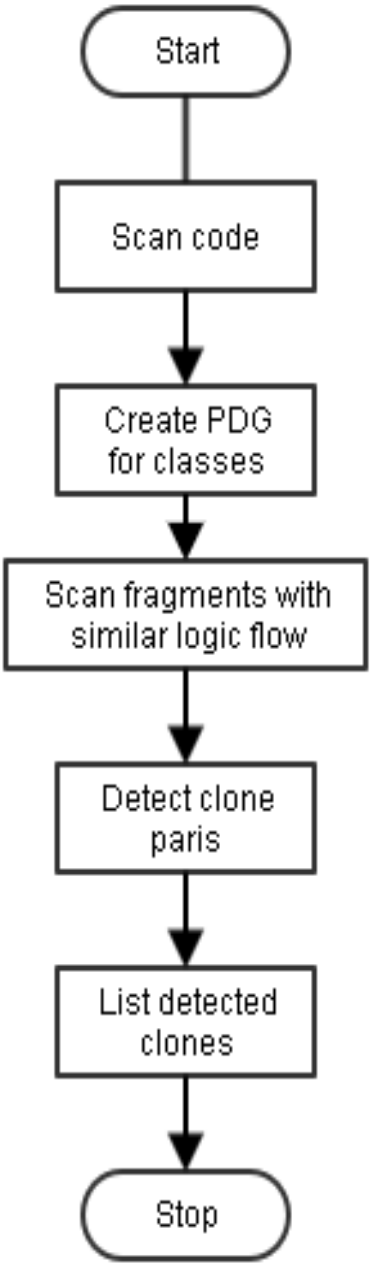

Figure 1.7: Program dependency graph

- **Hybrid based-** this technique is basically combination of two techniques by merging two or more techniques hybrid can be formed and then clones can be detected by this technique. This technique holds better value than normal technique. It provides better results in term of accuracy, efficiency, performance, etc. For example: program dependency graph and metric based technique can be used in a combination for best resultsin term of various parameters.

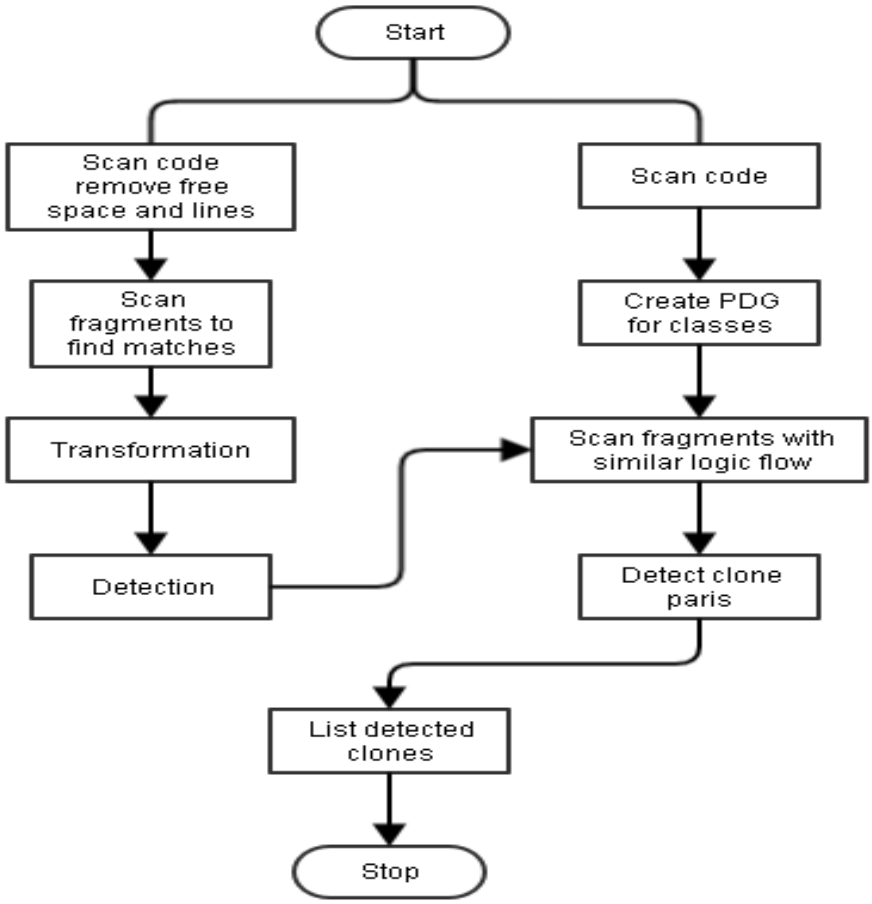

Figure 1.8: Hybrid based

1.6 MERITS OF CODE CLONING:

The various advantages for software cloning are [7]:

- **Help software growth research:** In software growth analysis Software code clone detection techniques are successfully used. as it helps in looking at the dynamic nature of different clones in different forms of a system.

- **Helps in Code Compression**: Code clone detection techniques can be used to outline the device by reducing the source code size.
- **Helps in Program learning**- As all files have a copy so they must implement a data structure with dynamically allocated space. Thus a code piece is used to regulate it.

1.7 DEMERITS OF CODE CLONING:

The various disadvantages for software cloning are:

- **Bug Propagation May Increase** -If a code segment holding a bug and that is reused by coping and pasting without or with minor transformation, the fault of the original section may remain in all the pasted sections in the system and accordingly the probability of bug propagation may increase.
- **Maximization in Probability of Bad Design** -Software code Cloning may lead to introduction of bad design, which may cause lack of good inheritance structure or abstraction.
- **Maximize the Maintenance Cost** -Due to presence of copied code, if a bug is found in one segment of code, there will be requirement to find similar bug in all cloned segments which is a difficult task. It becomes time consuming task.
- **Resource Requirement Maximization-** Code cloning proposes higher growth rate of the size of system. System size may not be a big problem In some domains.

1.8 APPLICATIONS OF CODE CLONING:

The various applications for software cloning are:

- **Detection of hold Fragments and Plagiarism**- Finding similar code is also useful in tracking down plagiarism and copyright infringement.
- **Discover Usage Patterns**- If the copied fragments of a same source fragment have been detected, the functional usage patterns of that fragment can be determined.
- **Detects mischievous software**- By correlating one malignant (malicious) software family to another evidences can be found where parts of one software system match part of another.

1.9 COMPARISONS OF CLONE CODE DETECTION TECHNIQUES [9]:

Table 1.5: Comparisons of clone code detection

Properties	**Text Based**	**Token Based**	**Tree Based**	**PDG Based**	**Metrics Based**	**Hybrid**
Transformation	Ignores white spaces and comments	Token generated from source	AST generated from source	PDG generated from source	To find metrics values AST is generate from the source code	Uses AST and PDG generated from source
Representation	normalized source code	In the form of tokens	Represent in the form of abstract syntax tree	Set of program dependency graph	Set of program dependency graph	Set of program dependency graphs
Comparison based	tokens of line	Token	Node of tree	Node of program dependency graph	Metrics value	Metrics value and Node program
Computational complexity	Depends on algorithm	Linear	Quadratic	Quadratic	Linear	Quadratic
Refactoring opportunities	Good for exact matches	Some Post processing needed	It is good for refactoring because Find syntactic clones	Good for refactoring	Manual inspection is required	Good for refactoring
Language in dependency	Easily adaptable	It needs a lexer but there is no syntactic knowledge required	Parser is required	syntactic knowledge of edge and PDG is required	Parser is required	Parser and syntactic knowledge of edge and PDG is required
Clone Types	Can find only Type I clones	Can detect type I & II clones	Can detect type I, II,III	Can detect type I, II,III & IV clones	Can detect type I, II,III	Can detect type I, II,III
Suffix tree comparison	No	No	Yes	Yes	No	Yes
AST	No	No	Yes	Yes	No	Yes

CHAPTER 2

REVIEW OF LITERATURE

Ritesh v. patil*et.al* in 2015 [8] proposed significantly detection of duplicate code and accomplish address type-3 and various other clones that has challenging condition in the research, the aim of this approach is to abate comparison and rectify precision, the decentralizes system, with multiple smart move smoothly executes the task. The Faster detection of clones has been designated by code reduction method.

Geetika bansal et.al in 2014 [3] proposed the uses set of metrics in metric based clone detection. On the basis of precision and recall values group of matrices are estimated. "Count Path" is used in proposed approach because it increases the precision value. It works on type-1, type-2 errors and from large set of matrices some matrices are selected which are having less correlation.

Sonam gupta et.al in 2014 [10] proposed that there are many code clone detection technique like text based and token based or abstract syntax tree or PDG and metric based. Many algorithms are developed on the basis of detection technique to detect type1, 2, 3, 4 clones, some are clone, but still none is able to find the clone with accuracy and efficiency. So this paper proposed "clone chunk algorithm" which will find all types of clones with accuracy and efficiency.

Judith F. Islam et.al in 2016 [5] proposed that the main focus is on finding the bug-replication tendencies of diverse clone types. The real impact of cloning on software maintenance and evolution cannot be understood without this as 55% bugs are replicated bugs. Method calls and if conditions are having higher tendency of containing replicated bugs, type 2 and type 3 clones have higher tendencies of replicated bugs then type 1 clone

Mena bharti et.al in 2014 [7] proposed about suited code clone detection techniques, merits, demerits, application of code clones due to the presence of code clone maintenance of software has become extremely difficult.

Bayu priyambadha et.al in 2014 [4] proposed that the main concern is how to evaluate input and output and their effects on void and method without parameters as

it is performed using PDG (program dependency graph). The semantic clone detection method is used to increase accuration of detection by 89% as trail is performed on each method.

Serge demeyer *et.al* in 2014 [11] proposed the correlation of three techniques that are parameterized matching, simple line matching, and metric finger prints. Simple line matching is best suited or overviewed for the duplicate code and metric fingerprint is best suited for combination with refactoring tool i.e. Able to remove duplication subroutines. Parameterizes matching works in fine-grained refactoring tool that works on the statement level.

Robin Sharma*et.al* in 2013 [1] proposed that this paper contends detection of functional clone from the source code using object oriented paradigm concept. This paper describe the hybrid approach which uses two techniques:-textual approach and metric based approach for detecting code clone in open source system. At the end of the paper, this approach is compared with the existing tool to detect the clone in the software and give accurate result which is being less complex.

Rubal sivakumar*et.al* in 2012 [6] proposed that this paper detects all types of clone in web application by using hybrid approach by the combination of textual and metric analysis. This method is least complex and provides efficient way of clone detection. The conclusion of this paper is to find functional clone and eliminate the duplicate code in web application and improve the poorly designed web applications.

Kuldeep Kaur et.al in 2015 [2] proposed that while making any software for preserving time and effort, program code is copied and pasted again and again. So if one module has bug, it is reproduced in every copy. Code cloning is a process in which some part of code is traced and then fixed with or without some minor modifications into another part of the code so that the part of the code can be reused. The pasted code segment is called a code clone. It has been observed that text based technique can detect only Type 1 clone. Token based technique detects Type1, Type II clone. Tree based approach detect Type1, Type II, Type III clone, program dependency graph approach is used for detecting Type IV clones but it is difficult to develop a syntax tree because its complexity is very high. It has been observed that on

the basis of precision, recall, robustness and scalability no technique is found optimum.

Prajila Prem et.al in 2013 [9] proposed that Code cloning means a process in which some part of code is traced and then fixed with or without some minor modifications into another part of the code so that it can be reused for the code. The fixed code section is known as code clone. In this paper various code clone detection methods, tools for detection approaches and technique used for that and the code analysis will be discussed. The textual approach gives a rough overview of the duplicated code that is quite easy to obtain, the token-based approach provides a precise picture of a given piece of duplicated code and is robust against rename operations. Syntactic techniques are very good at revealing duplicated subroutines and irrespective of small differences, as it works best with refactoring tools which works on the method level.

Fang-Hsiang Su, Jonathan Bell *et.al* in 2016 [19] reviewed in this research many software engineering tasks like program understanding and software refactoring which can be assisted by Identifying similar code in software system. Whereas most of the approaches focus on identifying code which looks alike and some aims at detecting code which functions in similar way. A novel method, In-Vivo Clone identification has been proposed that by detecting functional clones in arbitrary programs and mining their Inputs and outputs. The major concern is to use current workloads to execute programs and to measure amount of functional similarities between programs HitoshiIO, an open source and widely available. The pilot's results show that HitoshiIO can identify more than 800 practical clones over a corpus of 118 programs. In-Vivo Clone technique can effectively detect functional clones so it is used in system like Java, HitoshiIO. HitoshiIO applies static data flow analysis instead of fixing the definitions of program to identify potential inputs and outputs of individual methods. As a result further research that will leverage the information of function clones has enabled.

Norihiro Yoshida, Takashi Ishio*et.al* in 2011 [23] proposed in this research big number of code clones has been described whereas software developers are interested in only a subset of code clones which are relevant to software development tasks such as refactoring. In this paper, a method is proposed in which code clones for refactoring

activity are extracted by combining clone metrics. Japanese Software Company developed a study on web application which is being conducted by us. The result indicates that to extract refactoring candidates of clone metric are most effective than individual clone metric. A method is proposed in which sets for refactoring are extracted by using the result of CCFinder. We presented the advantage of the proposed method with an industrial case study using the source code developed by NEC. The case study shows that to extract clone sets of refactoring the proposed method is more useful than using sole clone metric.

Toshihiro Kamiya, Shinji Kusumoto*et.al* in 2016 [12] propsed that code clone is a code part of the source data (files) which is exact or similar to another. Several code clone detection tools and techniques have been proposed as code clones generally trim maintainability of software. New clone detection is proposed in which the conversion of input source data and token-by-token similarities are available. CCFinder is being developed on basis of proposed clone detection technique in which code clones extracted in java source files and C/C++. As well metrics for code clones were suggest. Transformation rules and a Token-based comparison are presented with clone detecting techniques. A metrics is suggested to select interesting clones. Which were applied to various industrial-size software systems in the experiments.

Geetika, raj Kumar*et.al* in 2014 [15] proposed that To rework a code segment by copying it from one section of the software and pasting it with or without some slight alterations into other sections of the software is called code cloning. It is a basic means of software reuse. A count of code clone detection techniques have been suggested so far. In this thesis, an approach is put forward which is metric based which is code clone detection technique for selection of a set of relevant metrics to detect code clones. The suggested approach evaluates a set of independent metrics that are assessed on the basis of the precision and recall values in clone detection starting from all aggregation of one metric to number of Metrics in the combinations until the complete set of metrics involved are evaluated in approach. Count Path is a new metric that has been used in the approach. The result of implementation of the way that is proposed on a C language software system shows that the use of the metric CountPath has enlarged the precision value to a great extent. A major condition of

metrics based techniques of code clone detection that is less Precise has overcome by this technique which is also computationally efficient.

Geetika chatley, Sandeep Kaur*et.al* in 2016 [24] proposed the current issue in industries is software cloning which is making recognition of clones a key of programming examination. The software or programming clones has been grouped comprehensively into various classifications. software cloning is Usage of actual code either by duplication and paste methods or by performing minor changes in the current code .through this paper , guide to a potential client of clone identification strategies and its help in choosing the right apparatuses or methods for their interests has been studied. Thus putting light on all the types of clones and techniques for the detection of clones that are practiced. The reasons of cloning along with its pros and cons and the process involved in detection of clones were presented by the paper. Since the last decade, numerous researchers in the field of software cloning have contributed magnificently.

Nam H. Pham, Hoan Anh Nguyen*et.al* in 2009 [30] proposed in this research for large scale software Model-Driven Engineering (MDE) has become a crucial development skeleton. Earlier research has reported that cloning also appeared in MDE as in traditional code-based development. This Paper represents ModelCD, which is a novel clone detection tool for Matlab/Simulink models that can fluently and precisely detect both exactly matched and approximate model clones. The unique graph-based twin identification methods is the origin of ModelCD that are able to methodically and incrementally invent clones with a high degree of accuracy, completeness, and scalability. The central ideas include the systematic generation of the candidate clones with the best techniques, and the accurate structural trait extraction for candidate sub graphs that have been implemented into ModelCD.

Gehan M. K. Selim, King Chun Foe*et.al* in 2010 [20] proposed in this research software clones detection in large scale projects helps improve the feasibility of large code bases. The source code representation (e.g., Java or C files) of a software system has been traditionally used for clone detection. In this paper, technique has been proposed which help transforms the source code to an intermediate representation, and reuses established source-based clone detection techniques to detect clones in the

intermediate representation. Thus a hybrid clone detection technique has presented. The technique complements string or token-based clone detectors which detects Type 3 clones by leveraging the intermediate representation .The recollection of the approach is higher than source based clone detectors with minimal drop in the precision using Bellon corpus which has incomplete clone groups. In the further, this approach can be applied on bigger systems and evaluated the time performance and scalability of the approach.

Chengnian Sun, Siau-Cheng Khoo*et.al* in 2011 [17] proposed that a simple practice is done to employ third-party libraries in software projects. Software libraries encapsulate a large number of useful, well-tested and robust functions to interact with libraries, as programmers only need to invoke Application Programming Interfaces (APIs) exported from libraries. This paper suggested a novel approach based on trace relation of data dependency graphs to detect imitations of library APIs for acquiring improved software maintainability. A prototype has made and investigated its utility on ten sizable open-source software projects. The experiment presents that technique can report 313 valid checks in total with high precision average of 82% for explicitly carried library APIs, and 116 valid checks with precision average of 75% for static library APIs.

Raghavan Komondoor, Susan Horwitz*et.al* in 2001 [26] proposed that cloned codes in program make both understanding and maintenance difficult. This flaw can be minimized by detecting cloned codes and selecting them into a independent new policy, and replacing all the clones by calls to the new policy. This paper describes the design and initial implementation of a tool that finds clones and displays them to the programmer. The novel aspect of this approach is the use of program dependence graphs (PDGs) and program slicing to find isomorphic PDG sub graphs that show clones. The implementation shows that the technique is a good one and real code does include the kinds of clones that our tool is well-suited to handle (and that most previous approaches to clone detection would not be able) and the tool does the clones that would be identified by a human.

Mark Gabel, Lingxiao Jiang *et.al* in 2008 [25] proposed the similar code fragments in programs have been identified by various approaches which have been refined.

These same fragments, called code clones they are used to identify redundant code .This paper, presents the scalable clone detection algorithm which is based on this definition of semantic clones. The impression is the discount of the tough graph comparison problem to a simpler tree comparison problem by scaling carefully selected PDG sub graphs which are related to their structured syntax. The extended definition of a code clone to comprise semantically related code which provides an approximate algorithm for locating these clone pairs. A practical tool based on our algorithm has implemented that scales to millions of lines of code. An intraprocedural analysis frame has been refined that could aid in creating PDGs more instantly and for other languages in future.

Priyanka Batta, miss Himanshi *et.al* in 2012 [21] proposed that software Clone detection is one of the main research area as duplicate code from an applications is detected by it. 5% to 20% of software systems contain duplicated code that is generated due to simply copying of the existing program code. The aim of this study is to analyze the working of hybrid clone detection technique that can design and analyze a hybrid technique for detecting software clone in an application. Metric approach with text base (line of code) technique is combined for that. A code clone detection system is designed and implemented which helps in detecting clones in the code efficiently and productively. The proposed system removed the problems occurred due to software clone like increase maintenance work and cost, defect probability and resource requirement.

Gurvinder Singh, Jahid Ali *et.al* in 2015 [29] proposed in this research the branch of Clone Detection has undergone a great advancement. This rise is due to the development of various solutions, which involves the implementation of complex algorithms and tool chains to offer clone detection. The intention is to present a survey of the various existing techniques and to develop a tool which is user friendly, easy to maintain and is not limited to small or big software. Clone detection method can be used for more complex applications like web based applications. This paper presents a review of the detection techniques and proposes an approach to deal with code clones in any situation. Enhancement in research scenario can be exaggerated with advanced algorithm in future to extend work.

Lingxiao Jiang, Ghassan Misherghi*et al* in 2007 [18] proposed that code clones detecting has many software engineering applications. Current techniques either do not scale to large code bases or are not robust against minor code alterations. In this paper, an efficient algorithm to find similar sub trees is presented to tree representations of source code. The algorithm is based on a novel characterization of sub trees with numerical vectors in the Euclidean space Rn and an efficient algorithm to cluster these vectors w.r.t. the Euclidean distance metric. The algorithm is implemented in a tool that is DECLARD which is language independent and highly configurable.

Saif Ur Rehman, Kamran Khan*et.al* in 2012 [16] proposed as the theory of code reuse is very common in software engineering. By code reuse the code is copied and pasted in many places in the same or different software without any alterations. For capturing duplicated redundant code, numerous code clone techniques and tools have been raised in last few decades. Each of these techniques attempts to find out the cloned code that is known as software clone. These approaches include Kclone, CP-Miner, CC-Finder, CReN etc. The technique is capable of diagnosing clones within wide source codes and is distinctive in its ability to identify code duplication independent of the source language. One of the prototype tool LSC Miner which takes a file of source code as input and the tool tokenizes it, storing it in a two-dimensional array. In the final stage, these tokens are correlated to each other to find clones.

Perumal. A, Kanmani. S *et.al* in 2010 [28] proposed that copying a code fragment and reusing it by pasting with or without minor modifications is a common practice in software development environments. Various techniques have been developed to find duplicated redundant code. Our proposal is a new technique for finding similar code blocks and for quantifying their similarity. Clone clusters, sets of code blocks can be found by technique all within a user supplied similarity. Similar clones can be detected using metrics for type 1, type 2 of clones. In our proposed system, after clone detection the system does two functions. In the clone clustering phase, the detected clones are grouped together after successful clone detection.

Noble kumara, **Anju saha** et.al in 2014 [14] proposed the effectiveness of different refactoring methods on quality attributes and to categories according to the results that they obtained on specific quality attributes. This paper concentrates on the reusability,

complexity, maintainability, testability, adaptability, understandability and completeness quality attributes. Here fourteen refactoring techniques were used to obtain the result for various quality attributes using the refactoring techniques. These quality attributes and refactoring methods used to enhance the quality of the software. Basically those refactoring methods were used which enhances the software metrics which integrate attribute, method, coupling, cohesion and inheritance of the software.

Anshu rani, **Harpreet Kaur** et.al in 2012 [27] proposed some refactoring techniques, tools and some features for code refactoring. Enhancement in the internal quality, maintainability and reliability without affecting external structure was done by using refactoring. Some steps were proposed to perform refactoring on code like identifying the code where it should be applied or determining the methods that can be used for particular place and assurance about maintaining behavior, applying refactoring technique and accessing the results of refactoring code. Some refactoring techniques were used like composing method that includes extract method that replaces temp with query, inline refactoring methods that moves feature between object includes move method, inline class for organizing code uses replace type code with class, change value to reference, and replace array with object for refactoring code. Eclipse and intellij Idea tools were used.

Ioana verebi et.al in 2015 [22] proposed a model based approach on the code refactoring was developed that gives better way to investigate reconstructing alternatives. For this approach a tool was implemented named as refractor that is used to fill the gap between structure flaw correction and detection. The historical data was validated by using some solutions that can be used to create high level structure, and compared the process to standard one in phase of speed, effectiveness and features of the code which is refractor. Transformed model was uses to compute weighted count of the code or design computed using baseline model and design flaw model.

Yoshio kataoka, Takeo Imai et.al in 2002 [31] proposed a quantities assessment method which helps calculating the improved maintainability results of code refactoring. Focus was on the coupling metrics to assess the effect of refactoring on code. In this paper the comparison between the coupling before and after refactoring techniques was made to improve the quality and assess the maintainability

improvement. The author used three coupling metrics along with different code refactoring method and combined these three coupling metrics to evaluate the code. This paper uses refactoring methods to improve the maintainability and software implementation of targeted software in order to choose a safe process. Refactoring Assistant was used as a refactoring tool for implementation of this method

Anam shahjahan, Wasi haider Buttet.al in 2015 [13] proposed in the paper which enhances the features of the code by using graph theory techniques and the procedure of enhancing the quality of code without changing its internal structure and external part is refactoring. Hypothesis techniques were used to correlate the results produced. Response time got improved through this study. The four main attributes that are Analyzability, changeability, time behavior and resource utilization were used to improve code quality.

CHAPTER 3

PRESENT WORK

3.1 PROBLEM DEFINATION:

Code clones are easy and quick way to add some existing logic from one section/module to another section or module. Code Clones make software code more bug prone. Clone is a persistent form of software reuses that effect on maintenance of large software. Another issue with code clones is that to change logic in application all clones has to be updated otherwise there is a risk of increased clone count in the application.

Developer might copy wrong code in different places or may simply skip updating all code clones whenever there is a change. In this research focus would be on increasing efficiency of clone detectors in to order detect clone count and thus increased customer satisfaction. By using hybrid approach higher accuracy, better efficiency and higher performance can be achieved as compared to existing approaches.

3.2 OBJECTIVE OF THE STUDY:

The main aim of the study is to achieve efficient results, higher accuracy and better performance than existing technique. To achieve this, objectives are as follows:

- To explore the code clone detection techniques and tools used for detection.
- To Developed and design an algorithm for enhancing code clones detection.
- To implement the propsed algorithm by merging two detection techniques.
- To Evaluate and authenticate the performance of propsed algorithm on standard data sets.

3.3 RESEARCH METHODLOGY:

Code cloning is done for saving time. The code is copied and fixed or pasted in multiple places in the software or different software without any alterations by code reuse Code Clones make software code more bug prone. Detecting code clones in software may save future efforts very easily.

Researchers have proposed a number of approaches for code clone detection techniques.

This approach is calculating code metrics first. Initially set of metrics vector are prepared for code fragments and then code units will give an idea of code clones. Candidate clone metrics will compared using program dependency graph which will detect code clones even if variables have been renamed and a few changes are done in logic. Combined results from metrics based detection and PDG would provide ample data for code clones detected with higher rate of accuracy. Methodology of the Research would be in following phases:

1. There are various techniques for Code Clone Detection. The main focus of this research is on Metric Based and Program Dependency Graph technique.
2. An Algorithm has been designed by merging two techniques (MTB, PDG) named as Hybrid technique.

   The structure of algorithm is defined as:

   a) Download latest code from repository

   b) Parse code and identify following items:

   - Calculate code metrics for code
   - Iterate code metrics and identify code clone fragment sections
   - Run second pass to match code semantics and filter unnecessary code clone candidates

   c) Repeat for all clone fragments:

   - Generate code metrics and assign score for each clone set
   - Score each candidate :
     i. Generate PDG for each clone candidate set
     ii. Compare PDG of the candidate set
     iii. Look for logical differences
       1. If LD <=2 Then
         a) Flag segment as code clone
       2. Else decrease score of the candidate fragment
     iv. If PDG match < 80% then
       1. Remove clone candidate from clone set
     v. Update score of remaining candidates
   - Prepare final list of clones

d) Show results.

3. This approach is calculating code metrics first. Initially Set of metrics vector prepared for code fragments and then code units will give an idea of code clones. Candidate clone metrics will compared using program dependency graph which will detect code clones. Combined results would provide higher rate of accuracy.

   The flowchart has been designed depicting methodology to be followed for research:

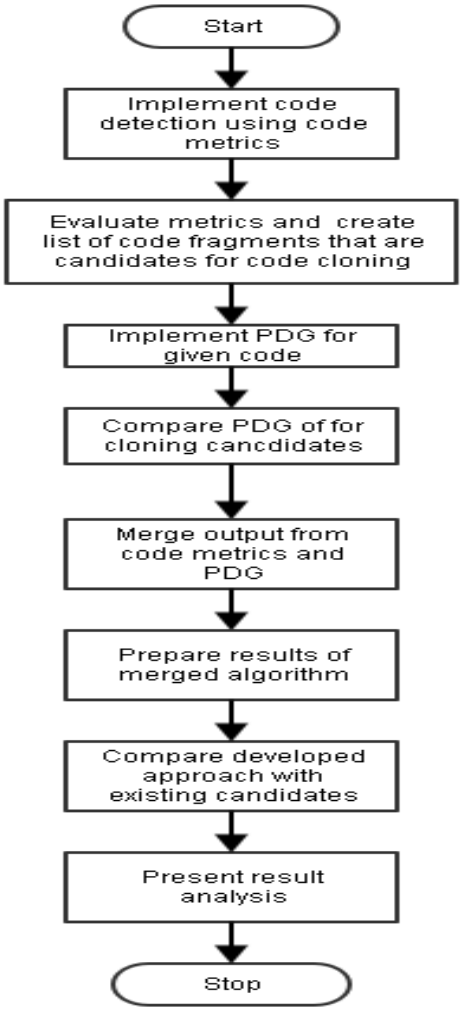

Figure 3.1: Proposed Flowchart

4. Comparison has been done between hybrid and existing technique on the basis of accuracy and performance and graphs are generated which are differentiating these two approaches

CHAPTER 4

RESULTS AND DISCUSSIONS

Developer might copy wrong code in different places or may simply skip updating all code clones whenever there is a change. In this research the focus would be on increasing efficiency of clone detectors in order to detect more code clone count and thus increased customer satisfaction. In order to resolve the issues discussed above, algorithm has been designed (called hybrid technique) by merging two approaches (MTB, PDG). Hybrid approach in achieving clone detection can help us to improve clone detection.

4.1 DATA CONSTRUCTION:

The dataset contains information release version of Junit open source java framework accessible in GitHub which gives details about projects. This project was discovered perfect for research reason due to the adequate number of releases adaptation and the measure of code between two adjacent releases. Junit's 3-4 arrivals will be examined.

4.2 EXPERIMENTAL RESULTS:

These results will show us, how system is working on the interface from project selection to cloned detection.

4.2.1 Eclipse view added: Eclipse is used as platform for this research. Algorithm For code clone detection will be running over eclipse. Below image shows initial default view when there is no calculation done. Code clone detection can be initiated by selecting the project.

Figure 4.1: Eclipse view

4.2.2 Initiating code clone detection: To calculate code clones, project selection must be done for starting code scan. Figure 4.2 depicts that data set is selected for metric based, program dependency graph and hybrid approach same way.

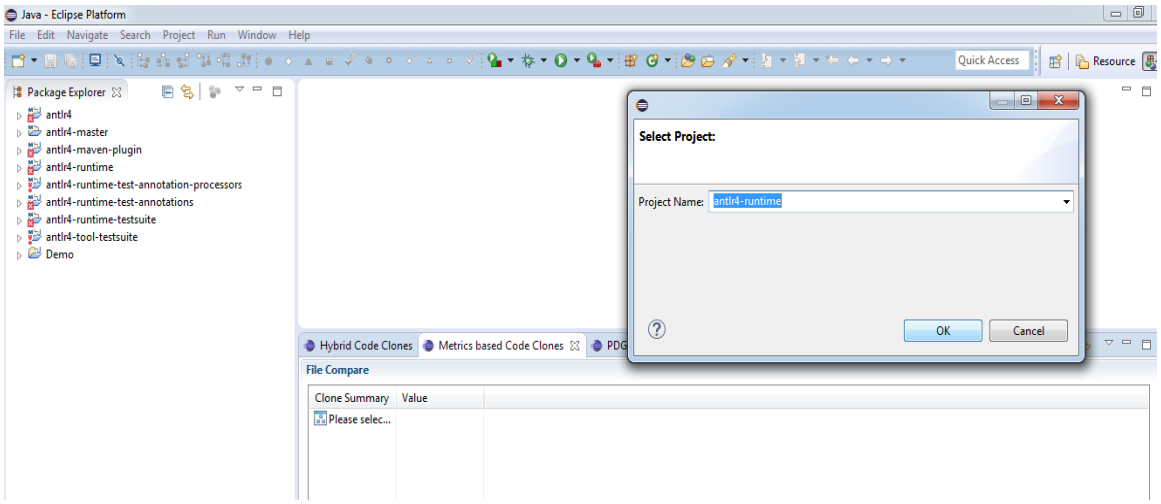

Figure 4.2: initiating clone detection

4.2.3 Calculation progress: After the selection of data set, all the classes will be progressed and it shows total no of files, duplicate files, raw lines and significant lines. Below figure shows that now we can select any of the file for viewing code clones. Click on opening differences.

a) **For metric based approach:** It finds 46107 number of code clones from the data set in metric based approach.

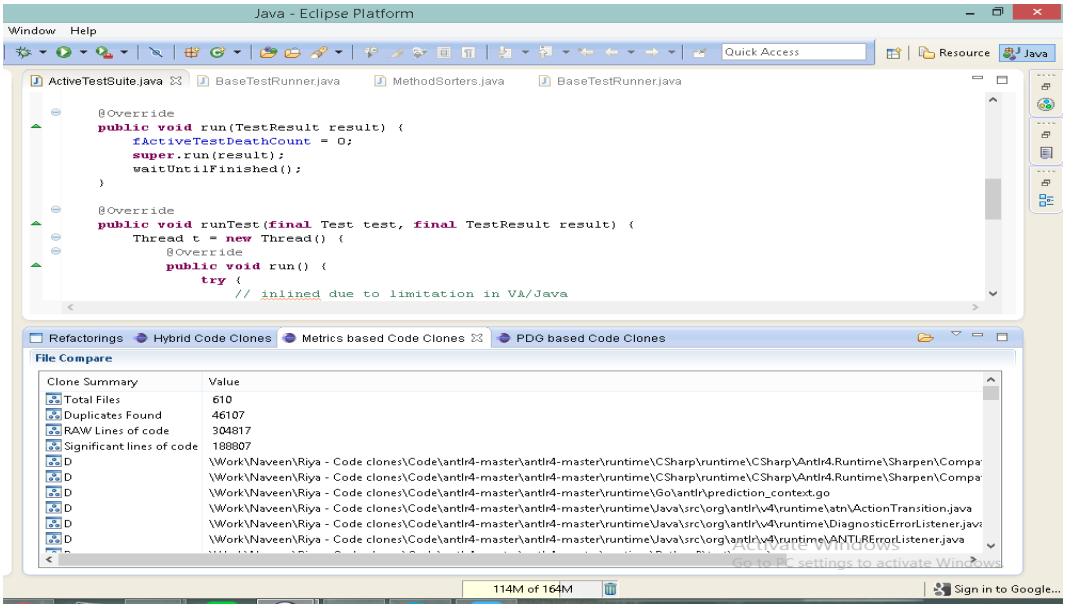

Figure 4.3: calculation progress for metric based

b) For PDG approach: It finds 49883 number of code clones from the data set. This is greater in number as compared to metric based approach.

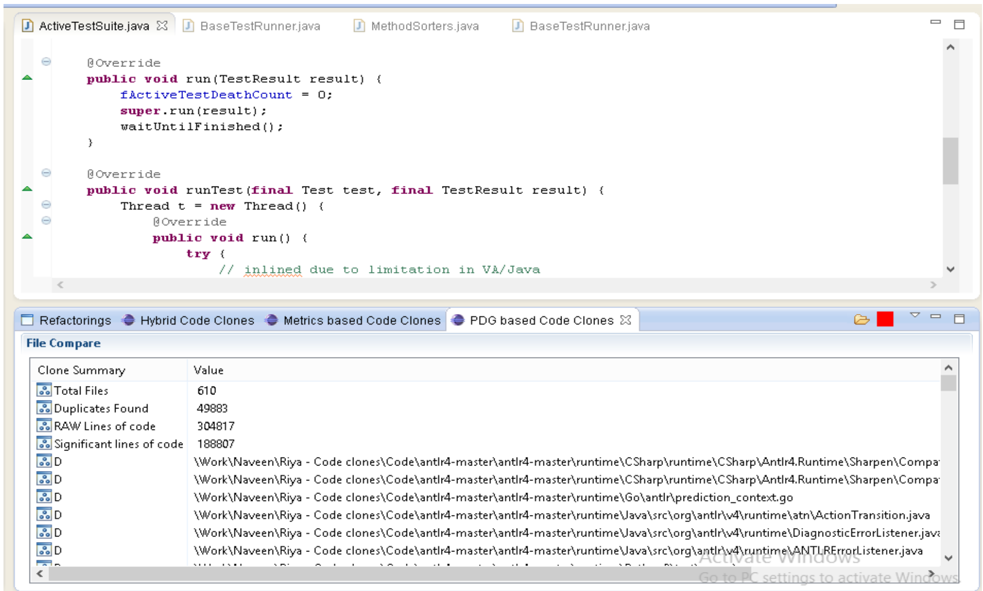

Figure 4.4: calculation progress for PDG

c) For Hybrid based approach: It finds 56723 maximum number of code clones in the data set. This is greater in number as compared to existing approach i.e. Metric based and PDG approach.

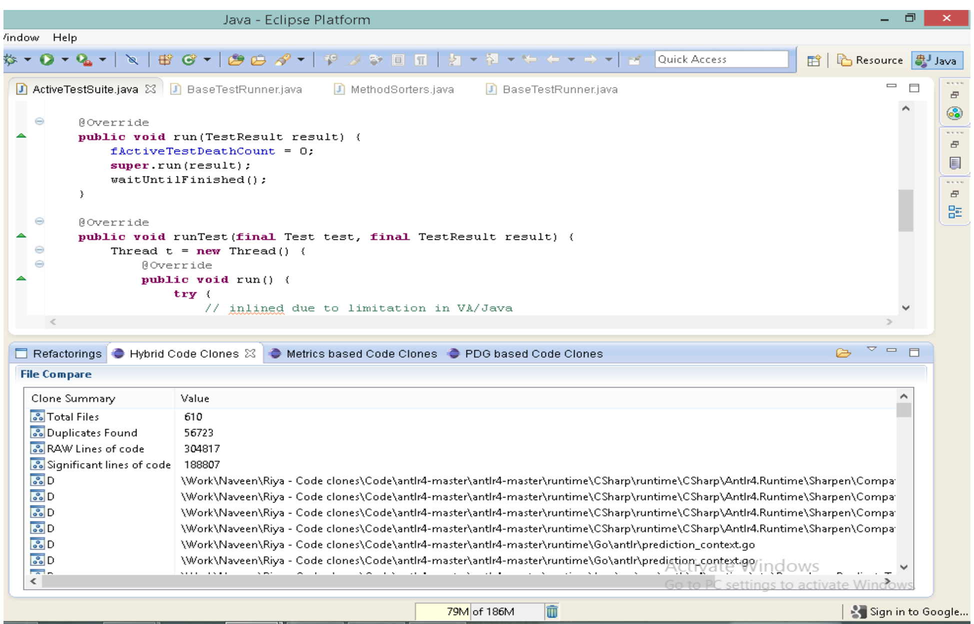

Figure 4.5: calculation progress for Hybrid

4.2.4 Comparer window: Below screen shows progress of the operation. It automatically fetches files from project and scans them to find code clones. In this it shows similar code clones found in two files. It shows code clones detected by tool and listed in eclipse view. Double click on any of the code clone entry in list opens up shows code block containing code clone. In expanded view eclipse editor shows the files opened in 2 columns of a screen and location where code clone has been detected.

a) **For metric based approach:** In metrics based approach code metrics are converted and stored into vectorized form for comparison. Code metrics vectors contain list of code metrics directed by code fragments. Comparison of code metrics fragments is used to find similarity between code fragments. We are using list of similar code fragments to calculate PDG and then comparing data and logic dependcy of the code metrics.

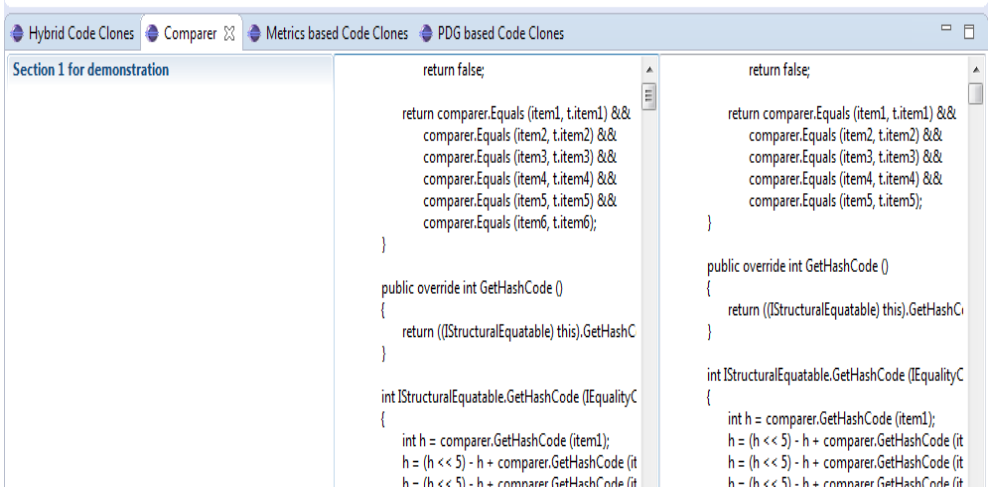

Figure 4.6: comparer window for metric based

b) **For PDG approach:** PDG uses program dependence graph for finding similarity between code clones. Here PDG is used for generated to view candidate clones for finding code clones. Data and logic dependence helps in identifying code clones even if variables have been renamed.

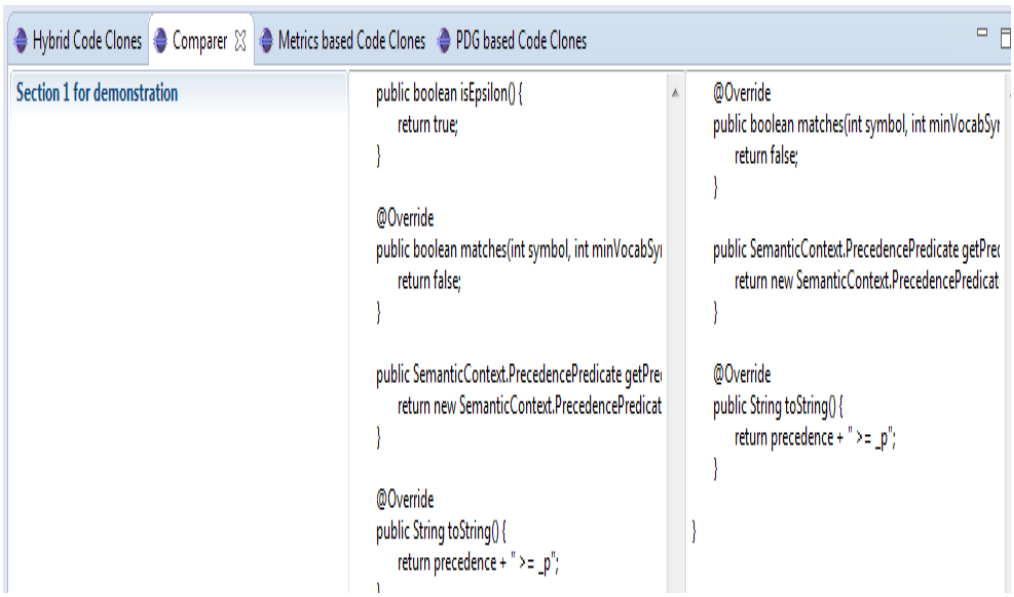

Figure 4.7: comparer window for PDG

c) **For Hybrid based approach:** Hybrid approach uses both code metrics and PDG based comparison of code clones. It starts with scanning code of all java files that are included in project. These code metrics are stored in vectors for finding similarity of code fragments. PDG for code fragments are generated to enhance the comparison. Complete list of code clone candidates to refine results obtained from both techniques.

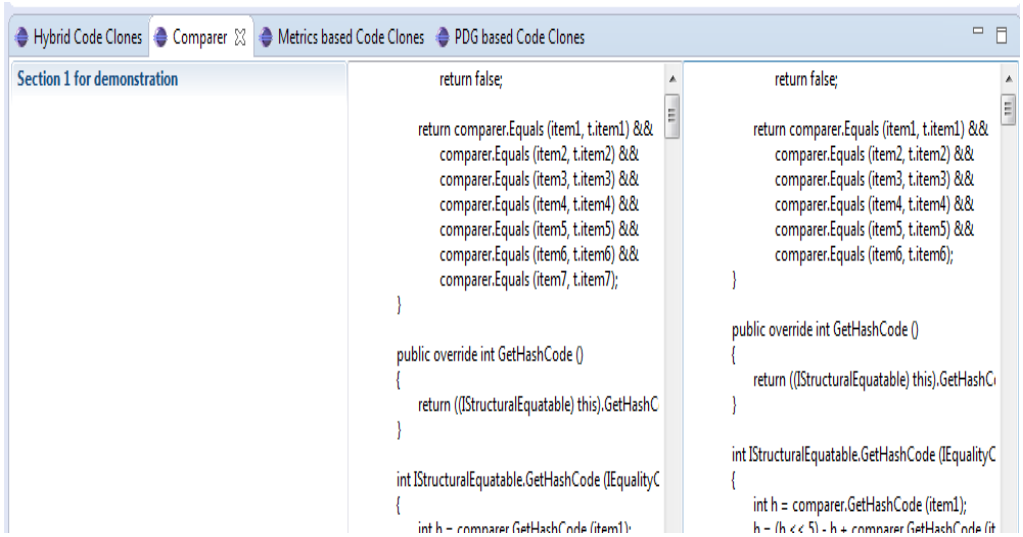

Figure 4.8: comparer window for Hybrid

4.2.5 Accuracy obtained: The values for Accuracy, false positives and true negatives are obtained for existing and proposed approach. Here false positives are defined in below section 4.3.2, true negatives in 4.3.3 and accuracy is defined in 4.3.6.

Accuracy= Accuracy of code clone is calculated by equation below. Where A is accuracy is total number of clones. CC is set of valid clones that it subset of total clones

Formula $$A = \left[\sum_{i=0}^{N} CC \Big/ N \right]$$ where N is total number of code clones found and CC is correct clones. A is accuracy

a) Metric Based

The below figure describes the obtained values for Total True negatives found 20.32%, Accuracy: 28.23% and False Positives: 71.75%.

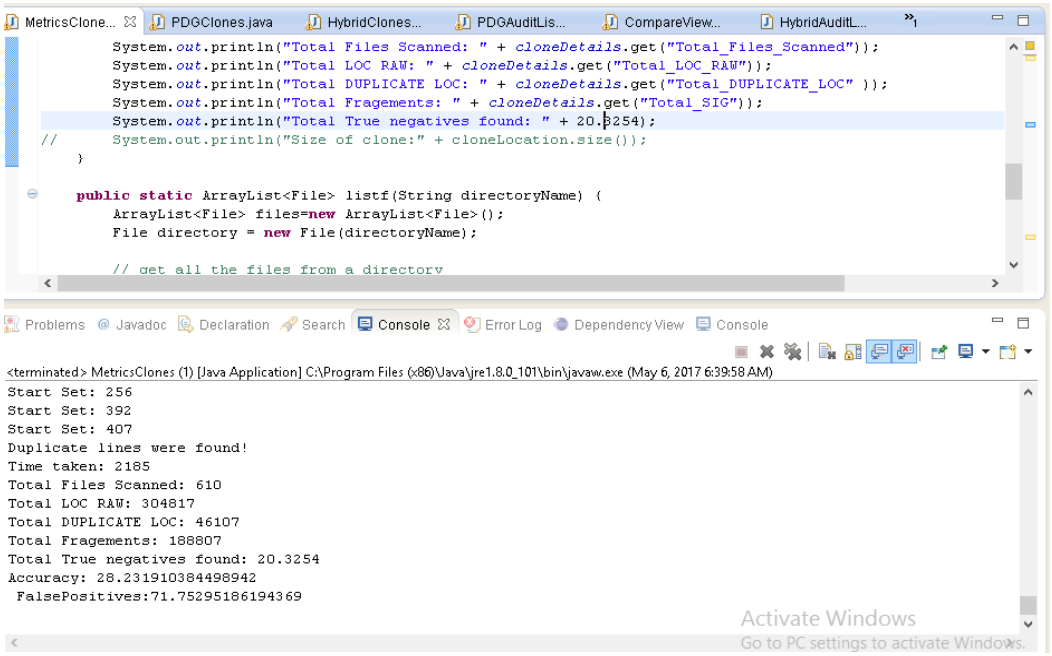

Figure 4.9: Accuracy for metric based

b) PDG

The below figure describes the obtained values for Total True negatives found 20.03%, Accuracy 30.01% and False Positives 69.96%.

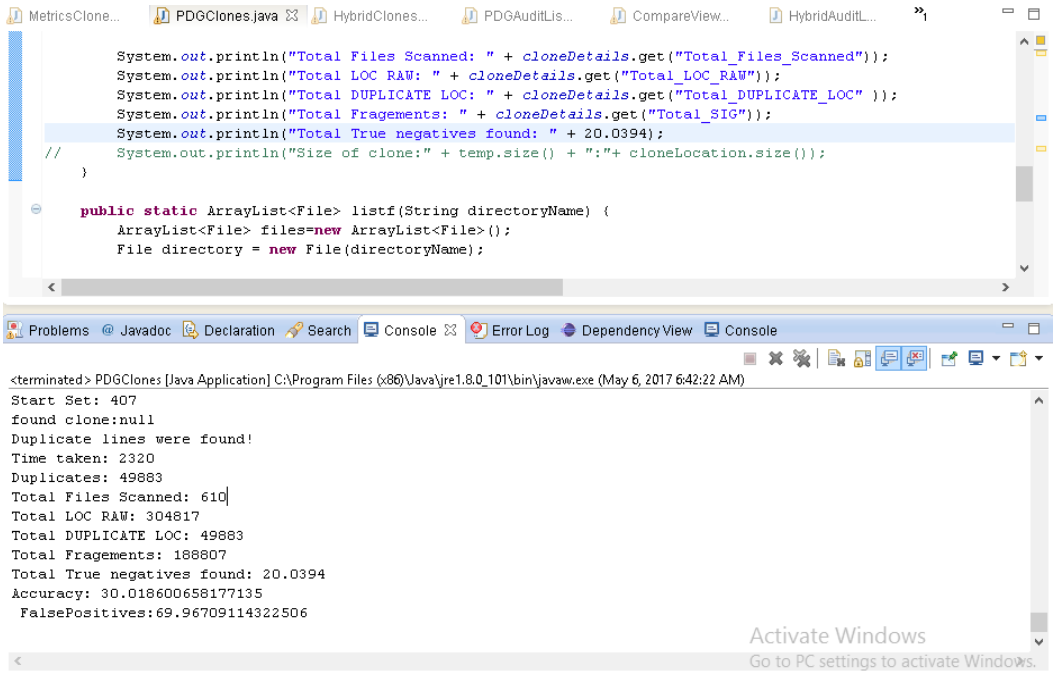

Figure 4.10: Accuracy for PDG

c) Hybrid based

The below figure describes the obtained values for Total True negatives found 8.35%, Accuracy 31.57% and False Positives 68.40%.

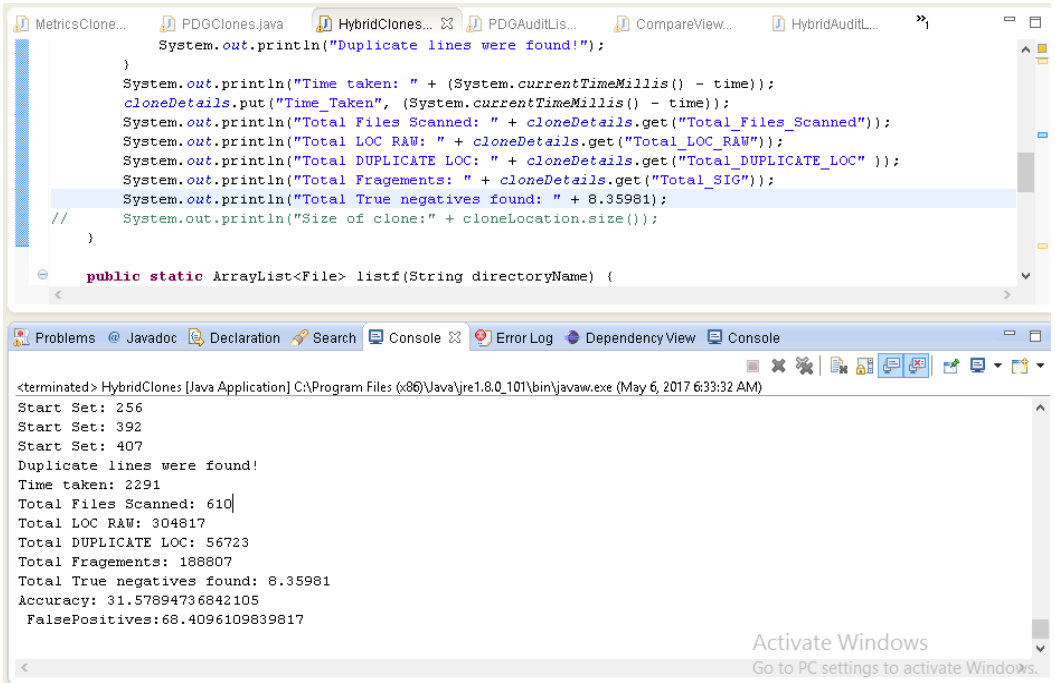

Figure 4.11: Accuracy for Hybrid

4.3COMPARISION WITH EXISTING TECHNIQUE:

4.3.1 Clones detection: number of clones found in antlr4 project

The table below represents number of code clones detected in data set by individual algorithms. Performance of algorithm shown by count of code clones is detected. Higher the code clones detected, more is the algorithm's detection power. Hybrid method adds to comparison of blocks with metrics and PDG which helps in better comparison of clone candidates.

Formula: Clone code detection= Total no of code clone found/ Total code fragments.

Table 4.1: clones detection

TECHNIQUES	%AGE OF CLONE FOUND
METRIC BASED	24.4
PDG	26.42
HYBRID	30.04

A graph is generated on the basis of above mentioned table 4.1. Which represents percentage of clones found in Hybrid approach and is compared with the existing approaches.

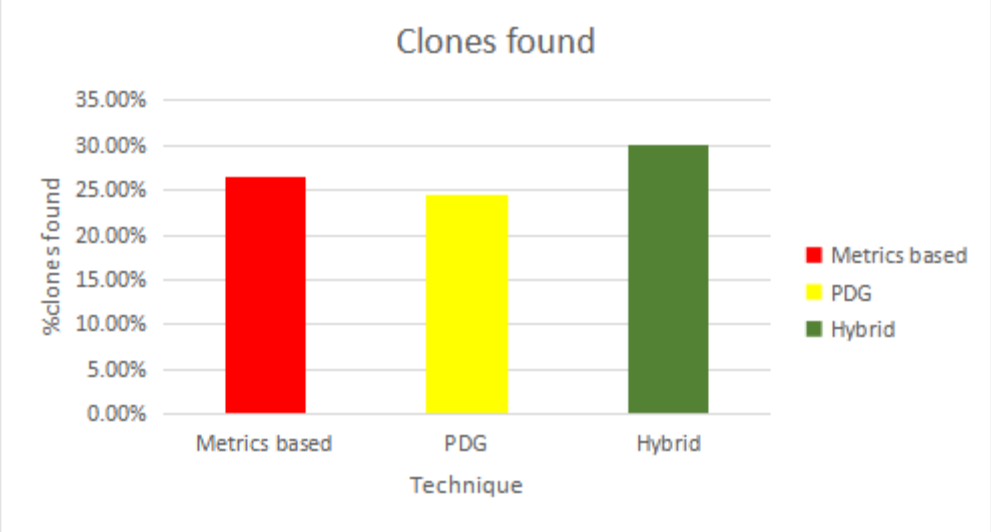

Figure 4.12: Clones found

4.3.2 False Positives: false positives are clones marked by algorithm but actually which are not code clones.

False positives are wrong clones detected by the algorithm. Higher number of false positives show lesser accuracy of algorithm and thus reduced efficiency. Using hybrid method helps in reducing false positives by adding PDG comparison to metrics based comparison of code.. One formula gives false positives FN where NC is set of incorrect clones and N is total number of clones that were not detected.

Formula:

$$FN = \left[\sum_{i=0}^{N} NC \Big/ N \right]$$ where N is total number of code clones found and NC is number of incorrect clones.

Table 4.2: false positives

TECHNIQUES	%AGE OF CLONE FOUND
METRIC BASED	72
PDG	70
HYBRID	68

A graph is generated on the basis of above mentioned table 4.2. Which represents percentage of false positive clones found in Hybrid approach and is compared with the existing approaches.

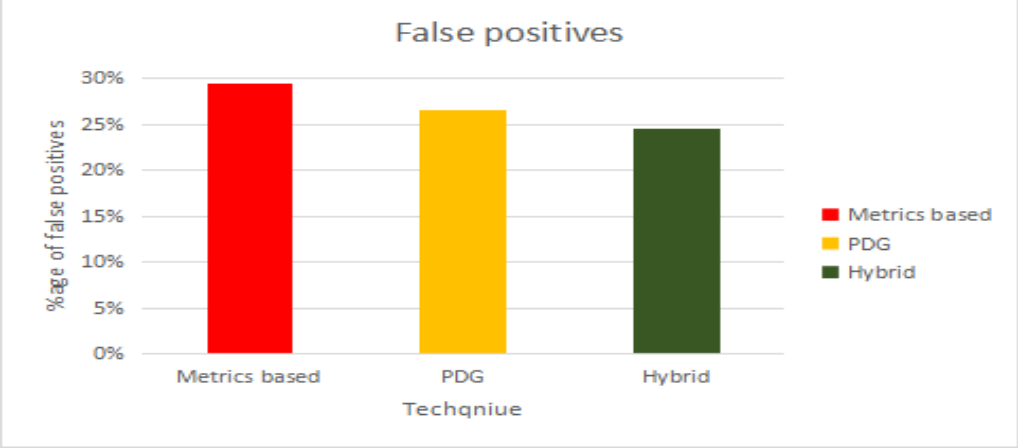

Figure 4.13: False Positives

4.3.3 True Negatives: True negatives are code clones missed out by the algorithm.

True negatives are cloning candidates that algorithm missed in detection. Here they found that there are lesser number of true negatives in hybrid algorithms thus it has higher accuracy as compared to PDG and Metrics based techniques. Using hybrid method helps in reducing true negative by adding PDG comparison to metrics based comparison of code. Metrics method added with PDG helps in tackle false positives. Formula gives true negatives TN where ECC is set of code clones found by simian but not found by algorithm. n is total number of clones that were not detected.

Formula:

$$\left[\sum_{i=0}^{n} ECC \Big/ n\right]$$ where ECC ∈ CodeClones from simian. n is total number of clones from Simian

Table 4.3: True negative

TECHNIQUES	%AGE OF CLONE FOUND
METRIC BASED	20
PDG	20
HYBRID	8

A graph is generated on the basis of above mentioned table 4.3. Which represents percentage of true negative clones found in Hybrid approach and is compared with the existing approaches.

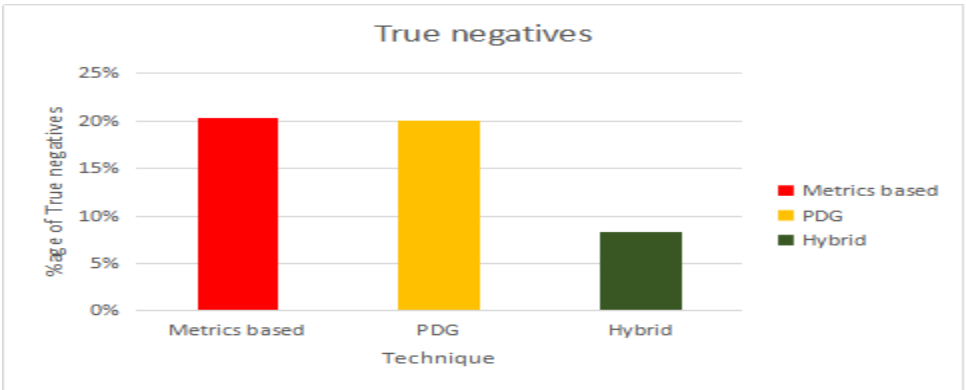

Figure 4.14: True Negatives

4.3.4 Performance Time: Time taken by algorithm to process.

Performance is time taken to process a set of code for clone detection. PDG and metrics based techniques take more time as compared to hybrid method raises performance of the code clone detection by avoiding over calculations of text based comparisons. One formula is used for calculating this as written below.

Formula: Performance time = current time before execution/ current time after execution (millisecond)

Table 4.4: performance time

TECHNIQUES	TIME TAKEN IN (MILLI SEC)
METRIC BASED	**3377**
PDG	**3775**
HYBRID	**3136**

A graph is generated on the basis of above mentioned table 4.4. Which represents performance time to clones found in Hybrid approach and is compared with the existing approaches.

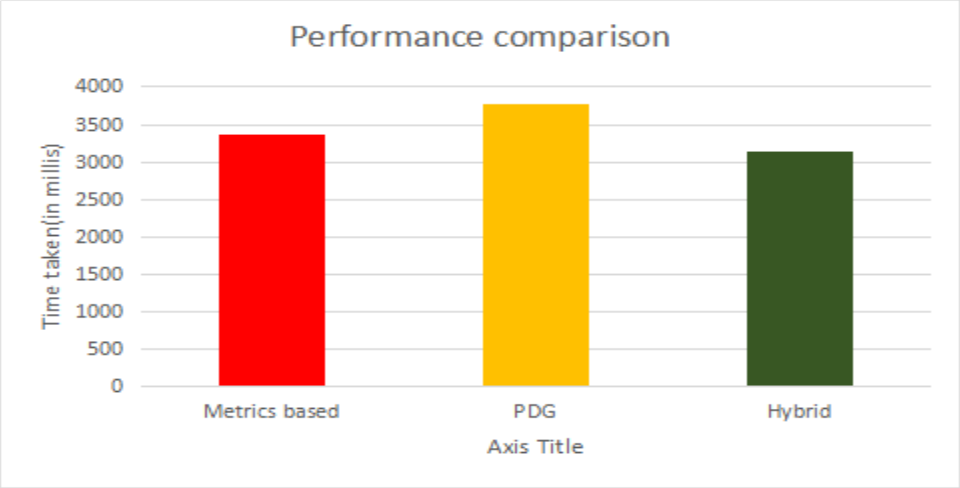

Figure 4.15: Performance Time

4.3.5. Metrics combination: Below table shows coverage with different number of metrics used. Code cloned detection was done on basis of number of metrics utilized.

Table 4.5: metric combination

# Metrics	Metrics based	PDG	Hybrid
1	15.540%	24.42%	22.78%
2	19.658%	24.42%	23.01%
3	21.654%	24.42%	24.60%
4	23.547%	24.42%	27.96%
5	24.589%	24.42%	29.66%
6	26.420%	24.42%	30.04%

A graph is generated on the basis of above mentioned table 4.5. Which represents percentage of metric combination of clones found in Hybrid approach and is compared with the existing approaches.

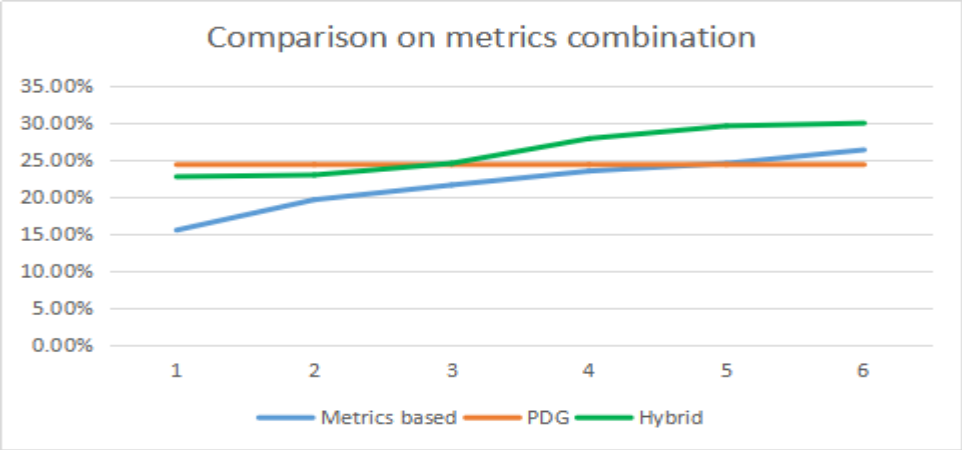

Figure 4.16: Metrics combination

4.3.6 Comparison on different releases on the basis of code clone detection in percentage: Below graph shows comparison of clone detection techniques on different software codes. We used 4 different java code repositories to scan and compare results from

Metrics, PDG and Hybrid techniques. Below table shows accuracy comparison with 4 different datasets.

Table 4.6: Comparison on different releases in percentage

DATASETS	METRIC BASED	PDG	HYBRID BASED
ANTRL4	24%	26%	30%
JEDIT	24%	26%	27%
FREECOL	24%	26%	27%
CAMELLIA	24%	26%	57%

A graph is generated on the basis of above mentioned table 4.6. Which represents percentage of comparison done on different releases in Hybrid approach and is compared with the existing approaches.

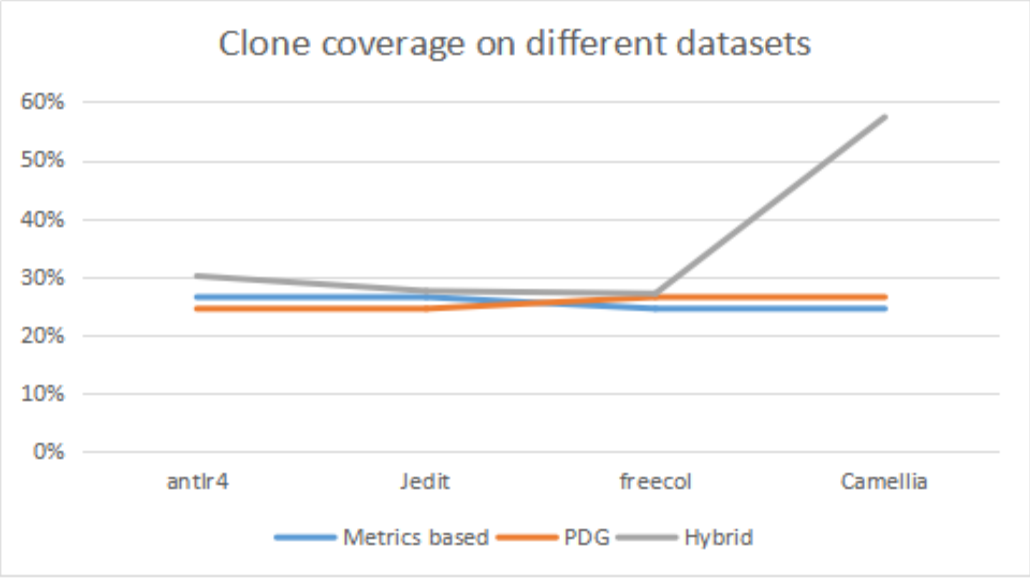

Figure 4.17: Comparison on different releases

CHAPTER 5

CONCLUSION AND DISCUSSIONS

5.1 CONCLUSION: Developer might have copied wrong code in different places or may have simply skipped updating all code clones whenever there is a change. In this research, the focus has been given on increasing efficiency of clone detectors in order to detect more code clone count and thus increased customer satisfaction. In order to resolve the issues discussed above, algorithm has been designed (called hybrid technique) by merging two approaches i.e. MTB, PDG. Hybrid approach has been used to achieve clone detection. This approach calculates code metrics first. Initially, set of metrics vector prepared for code fragments and code units gives an idea of code clones. Candidate clone metrics were compared using program dependency graph which detects code clones even if variables have been renamed and a few changes have been made in logic. Combined results from metrics based detection and PDG provides ample data for code clones detected with higher rate of accuracy. As shown in experimental results section, proposed technique proves to be of higher quality in terms of accuracy and performance. The comparison of metric based, PDG and hybrid techniques has been done and Hybrid technique performs best among existing techniques. Use of metrics in addition with PDG to find code clones has increased accuracy.

5.2 FUTURE SCOPE: Code clones detection is a way to keep code maintainable and healthy in terms of coding standards. This method has been proven better than existing techniques for detection. Code clone detection can further be enhanced by combining semantics based techniques. There are many languages used in industry for development. Studies can further be extended to include different languages.

REFRENCES

[1] Robin sharma, "Hybrid Approach for Efficient Software Clone Detection," vol. 3, no. 2, pp. 406–410, 2013.

[2] K. Kaur and R. Maini, "A Comprehensive Review of Code Clone Detection Techniques," vol. IV, no. Xii, pp. 43–47, 2015.

[3] G. Bansal and R. Tekchandani, "Selecting a set of appropriate metrics for detecting code clones," *2014 7th Int. Conf. Contemp. Comput. IC3 2014*, pp. 484–488, 2014.

[4] B. Priyambadha and S. Rochimah, "Case study on semantic clone detection based on code behavior," *Proc. 2014 Int. Conf. Data Softw. Eng. ICODSE 2014*, 2014.

[5] J. F. Islam and C. K. Roy, "Bug Replication in Code Clones : An Empirical Study," 2016.

[6] R. Sivakumar and K. . Kodhai.E, "Code Clones Detection in Websites using Hybrid Approach," *Int. J. Comput. Appl.*, vol. 48, no. 13, pp. 23–27, 2012.

[7] M. Bharti, R. Goyal, and M. Goyal, "Software Cloning and Its Detection Methods 1 1," vol. 8491, no. 2012, pp. 2012–2015, 2014.

[8] R. V. Patil, S. D. Joshi, S. V. Shinde, D. A. Ajagekar, and S. D. Bankar, "Code clone detection using decentralized architecture and code reduction," *2015 Int. Conf. Pervasive Comput. Adv. Commun. Technol. Appl. Soc. ICPC 2015*, vol. 0, no. c, 2015.

[9] P. Prem, "A Review on Code Clone Analysis and Code Clone Detection," vol. 2, no. 12, pp. 43–46, 2013.

[10] S. Gupta and P. C. Gupta, "Literature Survey of Clone Detection Techniques," vol. 99, no. 3, pp. 41–44, 2014.

[11] F. Van Rysselberghe and S. Demeyer, "Evaluating Clone Detection Techniques," *Evol. Large-Scale Ind. Softw. Appl.*, no. i, pp. 1–12, 2003.

[12] Toshihiro Kamiya, Shinji Kusumoto ,"A token-based code clone detection tool-ccfinder and its empirical evaluation," no. January, 2000.

[13] A. Shahjahan, "Impact of Refactoring on Code Quality by using Graph Theory : An Empirical Evaluation," pp. 595–600, 2015.

[14] N. Kumari and A. Saha, "E FFECT OF R EFACTORING ON S OFTWARE," pp. 37–46, 2014.

[15] Geetika,mr rajkumar, "Code Clone Detection by Evaluating Combinations of Software Metrics," no. June, 2014.

[16] S. U. Rehman, K. Khan, S. Fong, and R. Biuk-Aghai, "An efficient new multi-language clone detection approach from large source code," *Conf. Proc. - IEEE Int. Conf. Syst. Man Cybern.*, pp. 937–940, 2012.

[17] U. Edvhg, H. Ri, C. Sun, S. Khoo, and S. J. Zhang, "*udsk edvhg 'hwhfwlrq ri /leudu\ $3, ,plwdwlrqv," pp. 183–192, 2011.

[18] L. Jiang, G. Misherghi, and Z. Su, "D ECKARD : Scalable and Accurate Tree-based Detection of Code Clones ∗," no. 520320, 2007.

[19] F. Su, J. Bell, G. Kaiser, and S. Sethumadhavan, "Identifying Functionally Similar Code in Complex Codebases," pp. 1–10, 2016.

[20] 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.

[21] P. Batta, "HYBRID TECHNIQUE FOR SOFTWARE CODE CLONE," vol. 2, no. 2, pp. 97–102, 2012.

[22] I. Verebi, "A Model-Based Approach to Software Refactoring," pp. 606–609, 2015.

[23] E. Choi, N. Yoshida, T. Ishio, K. Inoue, and T. Sano, "Extracting Code Clones for Refactoring Using Combinations of Clone Metrics," pp. 7–13, 2011.

[24] G. Chatley, "Software Clone Detection : A review."

[25] M. Gabel, L. Jiang, and Z. Su, "Scalable detection of semantic clones," *Proc. 13th Int. Conf. Softw. Eng. - ICSE '08*, p. 321, 2008.

[26] S. Horwitz, "Using Slicing to Identify Duplication in Source Code."

[27] A. Rani, "Refactoring Methods and Tools," vol. 2, no. 12, pp. 256–260, 2012.

[28] A. Perumal, S. Kanmani, and E. Kodhai, "Extracting the similarity in detected software clones using metrics," *2010 Int. Conf. Comput. Commun. Technol. ICCCT-2010*, pp. 575–579, 2010.

[29] G. Singh and J. Ali, "A Novel Composite Approach for Software Clone Detection," vol. 126, no. 7, pp. 31–35, 2015.

[30] N. H. Pham, H. A. Nguyen, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Complete and Accurate Clone Detection in Graph - based Models," pp. 276–286, 2009.

[31] T. Imai, H. Andou, and T. Fukaya, "A Quantitative Evaluation of Maintainability Enhancement by Refactoring," 2002.

[32] J. Krinke, "Identifying similar code with program dependence graphs," *Proc. Eighth*

Work. Conf. Reverse Eng., pp. 301–309, 2001.

[33] S. K. Abd-El-Hafiz, "A Metrics-Based Data Mining Approach for Software Clone Detection," *Comput. Softw. Appl. Conf. (COMPSAC), 2012 IEEE 36th Annu.*, pp. 35–41, 2012.

[34] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," *Proceedings. Int. Conf. Softw. Maint. (Cat. No. 98CB36272)*, pp. 368–377, 1998.

[35] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello, "A Tree Kernel based approach for clone detection," *2010 IEEE Int. Conf. Softw. Maint.*, pp. 1–5, 2010.

[36] M. Gabel, L. Jiang, and Z. Su, "Scalable detection of semantic clones," *Proc. 13th Int. Conf. Softw. Eng. - ICSE '08*, p. 321, 2008.

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

[38] L. Jiang, G. Misherghi, and Z. Su, "D ECKARD : Scalable and Accurate Tree-based Detection of Code Clones ∗," no. 520320, 2007.

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