A

Project Report

on

CODE CLONE DETECTION USING LIGHT WEIGHT HYBRID APPROACH

Submitted in Partial Fulfillment of
the Requirements for the Degree
of

Bachelor of Engineering

in

Computer Engineering

to

North Maharashtra University, Jalgaon

Submitted by

Sayali S.Patil
Ashwini M. Sonawane
Sonal S. Salunke
Sachin S. Chaudhari
Makarand R. Bhole

Under the Guidance of

Mr.Pravin Patil

DEPARTMENT OF COMPUTER ENGINEERING
SSBT's COLLEGE OF ENGINEERING AND TECHNOLOGY,
BAMBHORI, JALGAON - 425 001 (MS)

2016 - 2017

SSBT's COLLEGE OF ENGINEERING AND TECHNOLOGY, BAMBHORI, JALGAON - 425 001 (MS)

DEPARTMENT OF COMPUTER ENGINEERING

CERTIFICATE

This is to certify that the project entitled *Code Clone Detection Using Light Weight Hybrid Approach*, submitted by

Sayali S.Patil
Ashwini M. Sonawane
Sonal S. Salunke
Sachin S. Chaudhari
Makarand R. Bhole

in partial fulfillment of the degree of *Bachelor of Engineering* in *Computer Engineering* has been satisfactorily carried out under my guidance as per the requirement of North Maharashtra University, Jalgaon.

Date: October 10, 2016
Place: Jalgaon

Mr.Pravin Patil
Guide

Prof. Dr. Girish K. Patnaik
Head

Prof. Dr. K. S. Wani
Principal

Acknowledgement

First of all we would like to extend our deep gratitude to almighty God, who has enlightened us with power of knowledge. We would like to express our sincere gratitude towards Principal Prof. Dr. K. S. Wani (SSBT, College of Engineering and Technology, Jalgaon) for his encouragement during the work. We wish to express our sincere and deep gratitude to Prof. Dr. Girish K. Patnaik (Head of Computer Department) for giving us such a great opportunity to develop this project. Inspiration and Guidance are invaluable in all aspects of life especially on the fields of gratitude and obligation and sympathetic attitude which we received from our respected project guide, Mr. Pravin Patil whose guidance and encouragement contributed greatly to the completion of this project. We would like to thanks to all faculty members of Computer Engineering Department and all friends for their co-operation and supports in making this project successful. We would also like to thanks our parents for supporting us and helping us. We acknowledge our sincere gratitude to all who have directly or indirectly helped us in completing this project successfully.

Sayali S.Patil
Ashwini M. Sonawane
Sonal S. Salunke
Sachin S. Chaudhari
Makarand R. Bhole

Abbreviations

AST : Abstract Syntax Tree

CC : Clone Cluster

CF : Code Fragment

CP : Clone Pair

DFD : Data Flow Diagram

LWH : Light Weight Hybrid

PDG : Program Dependency Graph

UML : Unified Modelling Language

Contents

Acknowledgement ii

Abbreviations iii

Abstract 2

1 Introduction 3

1.1 Background . . . 4

1.1.1 Textual Similarity . . . 4

1.1.2 Functional Similarity . . . 5

1.2 Motivation . . . 6

1.3 Problem Statement . . . 7

1.4 Problem Definition . . . 7

1.5 Objectives . . . 7

1.6 Summary . . . 7

2 System Analysis 9

2.1 Literature Survey . . . 9

2.2 Existing Techniques . . . 10

2.2.1 Classification of Code Clone Techniques . . . 12

2.2.2 Comparison of the Clone Detection Techniques . . . 12

2.3 Proposed System . . . 13

2.4 Feasibility Study . . . 14

2.4.1 Economic Feasibility . . . 14

2.4.2 Operational Feasibility . . . 14

2.4.3 Technical Feasibility . . . 15

2.5 Risk Analysis . . . 15

2.5.1 Software Risk . . . 15

2.5.2 Project Risk . . . 15

2.6 Project Scheduling . . . 16

2.7 Effort Allocation . . . 16

2.8 Summary 16

3 System Requirements **17**

3.1 Hardware Requirements 17

3.2 Software Requirements 17

3.3 Functional Requirements 18

3.4 Non-Functional Requirements 18

3.5 Summary 18

4 System Design **19**

4.1 Proposed System Flow 19

4.2 Data Flow Diagram 20

4.2.1 Level 0 21

4.2.2 Level 1 21

4.2.3 Level 2 22

4.3 UML Diagram 23

4.3.1 Use Case Diagram 23

4.3.2 Sequence Diagram 26

4.3.3 Class Diagram 29

4.3.4 Statechart Diagram 30

4.3.5 Activity Diagram 31

4.3.6 Component Diagram 32

4.3.7 Deployement Diagram 33

4.4 Applications 34

4.5 Advantages 35

4.6 Disadvantages 36

4.7 Summary 36

5 Conclusion **37**

Bibliography **38**

Index **39**

List of Figures

2.1 Project Scheduling 16

4.1 Clone Detector 20
4.2 DFD Level 0 21
4.3 DFD Level 1 22
4.4 DFD Level 2 23
4.5 Use Case Diagram for Clone Detector 24
4.6 Use Case Diagram for Metrics Computation 24
4.7 Use Case Diagram for Metrics Computation 25
4.8 Use Case Diagram for Template Conversion 26
4.9 Sequence Diagram for Clone Detector Module 27
4.10 Sequence Diagram for Method Identification 28
4.11 Sequence Diagram for Template Conversion 28
4.12 Sequence Diagram for Matrics Computation 29
4.13 Class Diagram for Code Clone Detection 30
4.14 State Diagram for Clone Detector Module 31
4.15 Activity Diagram 32
4.16 Component Diagram 33
4.17 Deployment Diagram 34

List of Tables

1.1 Type-1 Clone 4
1.2 Type-2 Clone 5
1.3 Type-3 Clone 5
1.4 Type-4 Clone 6

2.1 Classification of code clone techniques 12
2.2 Comparison of the clone detection techniques with respect to different properties 13
2.3 Effort Allocation 16

Abstract

Many researchers have investigated different techniques to automatically detect duplicate code in programs exceeding thousand lines of code.These techniques have limitation finding either the structural or functional clones.Code clones are the duplicated code which degrade the software quality and hence increase maintanance cost. Detection of code clone in software system is very neccessary to improve design,structure and quality of software product.The proposed Light weight hybrid approach combine texual analysis and matrices for detection of method level syntactic and semantic clones in C projects.

Keywords: Clone detection, Function clones, Source code metrics, String-matching.

Chapter 1

Introduction

Copying code fragments and then reusing them through the paste option with or without minor modification or adaptation is called Code Cloning and the pasted code fragment is called a clone. Most of the software systems comprise a substantial quantity of code clones; typically 10-15 % of the source code in large software systems are part of single or more code clones.

Code clone is a current area of research. To copy the code and reused the code by doing some modifications or without doing some modification in the exiting code are common activities in software development. The pasted code are called clone of the original and the process is called software cloning [4]. In the software system copied code fragments and code clones are considered as bad smell of the software. It is observed that code clone has bad effect on the maintenance of the software system. To remove the clones from the software systems is quite beneficial. These clones are syntactically or semantically similar [3]. It is very difficult to identify which code is copied code or which code is original. Several studies show that it is difficult to maintain software system which contains the code clones as compared to others which does not contain the clone. Cloning may increases the bug probability if some bug is found in the source code and that code is reused by copying and pasting then that bug is also found in that pasted code fragment. for fixing the bug all these code fragment should be detected. Code clones are basically of four types , where the first three Type I, Type II, Type III are textual and last one Type IV is functional [2].

- Reasons of Code Duplication:-

  There are various reasons for code duplication. Reuse of code, logic and design is the main reason of code duplication. Sometimes there is a need to merge two similar system having similar fuctionalities to develop a new one which result duplication of code even both the system are developed by different teams. There is frequent update of the software Developers are asked to reuse the existing code because of high risk in developing the new code. One of the major cause of code duplication is the time

limit assigned to developers. To complete a project some time limit are assigned to developers. Developers find the easy solutions of the problem due to time limit. They find the similar code related to their project .they just copy and paste the existing code [1].

In this chapter, in Section 1.1 background of the code clone detection is presented. Code clone detection motivation presented in Section 1.2. In Section 1.3 problem statement of the project is presented. Problem definition is presented in Section 1.4. In Section 1.5 project objectives is presented.

1.1 Background

On the basis of functionalities and program text, two code fragments are said to be similar. The first type of clone are mainly the result of copy and paste activities. In the following type of clones Type I , Type II and Type III clones are based on the textual similarity and Type IV clones are based on the functional similarity [4].

1.1.1 Textual Similarity

- ***Type-1*** is an exact copy without modifications (except for whitespace and comments). In Type I clone, two code fragments are identical to each other. However, there might be some variations in white space, comments and layouts.

  For example:-

  The clone pair (a, b) is of type-1 which have exactly the same code except the alignment, space and comment.

Table 1.1: Type-1 Clone

Source Code(a)	Type-1 Clone(b)

int main()
{
int x=1;
int y=x+5;
return y;
}
	int main()
	{
	int x=1;
	int y=x+5;
	return y;
	}

- ***Type-2*** is a syntactically identical copy; except some changes in variable name, data type, identifier name, etc.

  The clone pair (a, c) is of type-2 which have minor differences in function names and parameters.

Table 1.2: Type-2 Clone

Source Code(a)	Type-2 Clone(c)

int main()
{
int x=1;
int y=x+5;
return y;
}
	int fun2()
	{
	int p=1;
	int q=p+5;
	return q;
	}

- ***Type-3*** is a copied fragment with further modifications. Statements can be changed, added or removed in addition to variations in identifiers, literals, types, layout and comments.

  The clone pair (a, d) is of type-3 with additional statements in code, as they need not be functionally similar.

Table 1.3: Type-3 Clone

Source Code(a)	Type-3 Clone(d)

int main()
{
int x=1;
int y=x+5;
 	
return y;
}
	int fun2()
	{
	int s=1;
	int t=s+5;
	t=s++;
	return t;
	}

1.1.2 Functional Similarity

- ***Type-4*** Two or more code fragments that perform the same computation , but implemented through different syntactic variants.

  The clone pair (a, e) is of type-4 clones with no similarity in code, but the output of the functions are same.

Table 1.4: Type-4 Clone

Source Code(a)	Type-4 Clone(e)

int main()
{
int x=1;
int y=x+5;
return y;
}
	int fun4()
	{
	int n=5;
	return ++n;
	
	}

The results of the code clone detection are presented as clone pairs and clone clusters.

- ***Clone Pair (CP)or Code Fragment (CF) :***

  Clone Pair or Code Fragment (CF) is pair of code portions/fragments that are identical or similar to each other.

- ***Clone Cluster or Clone Class or Clone Set (CS):*** Clone Cluster is the union of all clone pairs that have code portions in common.

The quality of clone detection is assessed by two key parameters precision and recall.

1. Precision

   Precision is the ratio of the number of correctly detected clones to the total number of detecting clones by the proposed tool.

2. Recall

   Recall is the ratio of the number of correctly detected clones by the proposed tool to the total number of actual clones in the project by reference values.

1.2 Motivation

A number of clone detection techniques have been proposed. Among them, Text-based techniques are lightweight and are able to detect accurate clones with higher recall values, where recall refers to the overall percentage of clones exist in the source code that have been detected by the clone detector. However, it failed to detect suitable syntactic units. Token-based techniques are fast with high recall, but failed in precision. Precision refers to the quality of clones returned by the clone detector. Parser-based techniques are worthy in detecting syntactic clones. However, they give low recall values. Metric-based techniques are able to detect syntactic as well as semantic clones with high precision values. They are also very fast in detecting both syntactic and semantic clones. However, they fail to detect

some of the actual clones. PDG (Program Dependency Graph) based techniques are able to find more semantic clones, where PDG is a directed graph which represents the dependencies among program elements in a program. However, sub-graph comparisons are very costly. These limitations in existing methods provide a path to investigate hybrid or combinational techniques in order to overcome them.

1.3 Problem Statement

Code clones have bad impact on the maintainability, reusability and quality of the software. If there is any code segment present in the software which having a bug and the code segment is copied and pasted anywhere in the system then the bug is remains in all the pasted code segment which is difficult to maintain. When duplicated code used in the system it may lead to bad design which increase the cost of the system. If in the software system there is duplicated code, to understand the system additional time needed. It becomes difficult to upgrade the system or even to change the existing one.

1.4 Problem Definition

Develop a system that detect clone from copied source code. User can give two C files as input. The Clone Detector module take input from user. Removal of unnecessary part of code and standardization is performed on the input. The method identification module identify and seperates methods. Metrics Computation and Template Conversion is performed on the complete source code of two files to detect clone based on similarity between these two source files. On the basis of resultant clone pair clone type is detected.

1.5 Objectives

The main aim of this project is to detect syntactic and semantic clones which will cover all four type of clones. A LWH (Light Weight Hybrid) approach has been proposed with a combination of textual comparison and metrics computation. As there is no need for external parsing, this approach is of light weight. Moreover, a model has been arrived to detect syntactic and semantic clones which will cover all four types of clones.

1.6 Summary

In this chapter, information about background, Motivation, problem statement, problem definition and objective of the project are presented. In the next chapter, literature review

of related work in clone detection, existing system, proposed system and feasibility study of the project are presented.

Chapter 2

System Analysis

System analysis is the process of gathering and interpreting facts, diagnosing problems and using the facts to improve the system. System analysis chapter will shows overall system analysis of the concept, description of the system, meaning of the system. System analysis is the study of sets of interacting entities, including computer systems analysis. The development of a computer based information system includes a systems analysis phase which produces or enhances the data model which itself is to creating or enhancing a database.

There are a number of different approaches to system analysis. When a computer based information system is developed, systems analysis would constitute the development of a feasibility study, involving determining whether a project is economically, socially, technologically and organizationally feasible.

In this chapter, In the Section 2.1 literature survey is presented. Existing techniques of code clone detection is presented in Section 2.2. In Section 2.3 feasibility study of proposed system is presented. Risk analysis is presented in Section 2.4. In Section 2.5 project scheduling. Effort allocation table is presented in Section 2.6.

2.1 Literature Survey

There has been more than a decade of research in the field of software clone.Clone detection research has proved that software systems have 9 %-17% of duplicated code [3]. Thummalapenta indicated that,in most of the cases, clones are changed consistently and for the remaining inconsistently changed cases, clones undergo independent evolution. Effective code clone detection will support perfective maintenance.Comparison and evaluation of code clone detection techniques have been carried out by Bellon,Koschke and Roy and Cordy.

A clone detection process is usually done by converting the source code into another form that is handled by an algorithm to detect the clones. A rough classification is then carried out depending on the level of matches found. Token-based technique use a similar sequence matching algorithm. However,its accuracy is not that adequate as the normalization, and

also token conversion process may bring false positive clones in result set. Many of the clone detection approaches have used Abstract Syntax Tree (AST) and suffix tree representation of a program to find clones [4]. Some of the clone detection techniques use an AST that is generated by a pre-existing parser. Baker describes one of the earliest applications of suffix trees for the clone detection process. An algorithm based on feature-vector computation over AST was applied by Lee to detect similar clones. However, all of them use parsing, which results in heavy-weighted approach.

Lighter weight techniques without the use of parsing namely text-based techniques and metrics-based techniques. Text-based techniques are investigated by comparing two code fragments with each other to find longest common subsequences of same text/strings to detect clones. Though these techniques detect clones they are not low in precision values. Metric-based techniques identify a set of suitable metrics to detect a particular type of clone. By a quantitative assessment of the metric values in the source code, the clone detection is done.

Marco Funaro proposed Hybrid technique. A proposed a hybrid technique using Abstract Syntax Tree to identify clone candidates and textual methods to discard false positives [6]. Leitao also proposed a hybrid approach with the combination AST and PDG. Both approaches use parsing which results in heavy-weight. As text-based techniques preserve higher recall, metrics-based techniques preserve higher precision and both of them are light-weight, a hybrid technique with the combination of textual analysis and metrics, is experimenting for the detection of all four types of clones.

2.2 Existing Techniques

In the literature several types of clone detection tools techniques are presented . For the research purposes most of he techniques are used, while a few of them are also used for commercial purpose. Following are some existing techniques for code clone detection:-

1. Text-based Techniques:

   In the text based technique the source code fragment are assumed as sequence of line. After removing the various comments, whitespace by applying the various transformations the code fragment are compared with each other. Once the two code fragment are found to similar to each other to some extent they are known as clone pair or clone pairs form the clone class [3]. Sometimes in the clone detection process the source code is directly used. Text based technique is efficient technique but it can detect only Type I clones. Text based approach can not detect the structural type of clone having the

same logic but different coding. In the text based approach following transformations are applied on source code.

(a) Comments Removal: In the code fragment ignore all the comments.

(b) White space Removal: In the code fragment removes all the tabs and blank spaces.

(c) Normalization: On the source code some normalization are applied.

Though text based approach can detect only type 1 clone . This technique cannot detect the structural type of clones having same logic but different coding.

2. Token-based Techniques:

   In the token-based technique, first sequence of tokens is generated from the source code. For converting the source code into tokens it requires a lexer. Lexer convert the source code into tokens then the various transformation are performed by adding, changing or deleting some tokens. For finding the duplicated code or duplicated subsequence of token the sequence is scanned. and the code portions representing the duplicated code returned as clones. Token based technique can detect Type I, Type II clone [7].

3. Tree-based Techniques:

   In the tree-based approach from the source code a parse tree or an abstract syntax tree is obtained. This technique creates sub trees rather than creating tokens from each statements [5]. The code then said to be code clone if the sub trees match. With the help of parser of a language similar sub trees are searched in the tree using tree matching algorithm or structural metrics then the code of similar sub trees are returned as clone pairs. Abstract syntax tree have the complete information about the code. The result obtained from this technique is quite efficient but to create a abstract syntax tree is difficult for a large software and the scalability is also not good [1].

4. PDG-based Techniques:

   Program Dependency Graph(PDG) technique is more efficient then tree based technique. Program dependency graph show data flow and control flow information. First the program dependency graph is obtained from the source code then to find the similar sub graphs or clones several type of sub graph matching algorithm are applied and returned as clones [3]. This technique can detect both semantic and syntactic clones but in case of large software to obtain the program dependency graph is very difficult.

5. Metrics-based Techniques:

   In Metrics based Technique first different types of metrics of the code like number of lines and number of functions are calculated and compare these metrics to find the clones. Metrics based technique does not compare code directly. To find the code clones several type of software metrics are used by clone detection techniques. Most of the time, for calculating the various type of metrics the source code is converted into abstract syntax tree or program data graph. Metrics are calculated from the name, layout, control flow and expression of the functions [7].

2.2.1 Classification of Code Clone Techniques

Table 2.1: Classification of code clone techniques

Category	Text Based Texual Approach	Token Based Lexical Approach	Tree Based Syntactic Approach	PDG Based Semantic Approach
Clone Type	Type-1	Type-1,2	Type-1,2,3	Type-1,2,3
Meaning of n	Lines of code	Number of token	Nodes of tree	Nodes of PDG

2.2.2 Comparison of the Clone Detection Techniques

A clone detection process is usually done by converting the source code into another form that is handled by an algorithm to detect the clones. A rough classification is then carried out depending on the level of matches found. Token-based techniques use a similar sequence matching algorithm. However, its accuracy is not that adequate as the normalization, and also token conversion process may bring false positive clones in result set. Many of the clone detection approaches have used Abstract Syntax Tree (AST) and suffix tree representation of a program to find clones. Some of the clone detection techniques use an AST that is generated by a pre-existing parser. An algorithm based on feature-vector computation over AST was applied by Lee et al to detect similar clones. However, all of them use parsing, which results in heavy-weighted approach.

In Text based approach only type 1 clone is detected. It cannot detect the clones which having structural similarity but having different coding or have same logic. Token based approach is more efficient then text based approach it can detect type 1 clone as well as type II clone [4]. In tree based approach a parsed tree is generated from the source program. The result which is obtained from the tree based approach is efficient but to create a syntax tree is very difficult and scalability of this is also not good. Program dependency approach contains the data flow and control flow information of a program. The semantic information

of a program also contain in the program dependency graph. It detect type 1, type II and type III clones [6].

Table 2.2: Comparison of the clone detection techniques with respect to different properties

Properties	Tree Based	Token	Tree	PDG Based
Transformation	Removes whitespace and comments	based Tokens is generated from the source code	Based AST is generated from source code	PDG is generated from source code
Representation	normalized source code	In the form of tokens	Represent in the form of abstract syntax tree	Set of program dependency graph
Comparison Based	tokens of line	Token	Node of tree	Node of program dependency graph
Computational complexity	Depends on algorithm	Linear	Quadratic	Quadratic
Refactoring opportunities	Good for exact matches	Some Post processing needed	It is good for refactoring because Find syntactic clones	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

2.3 Proposed System

All the advantages and disadvantages of various approaches discussed in section 2.2 clearly show that although many techniques but still none is able to find the clones correctly. So we are proposing a hybrid technique which is able to find more number of true positives. This approach will find all the clones in the system irrespective of their place and will show the same to the programmer so that after or during the development of the code the programmer itself can identify the chunks which contain the clone and can decide whether to remove the clone or it is a good smell. In the proposed work one or two fragments can be compared [2]. It will give the result in form of chucks so it will be called as clone-chuck extraction algorithm. Firstly, the statements of the code will be examined serially if the statements are found to be reordered then the semantic-preserving transformation will be applied to the code so that reordering of the code does not affect the procedure of identifying the clone. This approach will be a bit faster removing the sluggish behavior of many approaches discussed above and

also will handle all types of clones.

A LWH (Light Weight Hybrid) approach has been proposed with a combination of textual comparison and metrics computation. As there is no need for external parsing, this approach is of light weight. Moreover, a model has been arrived to detect syntactic and semantic clones which will cover all four types of clones.The proposed LWH approach is able to detect method/function level clones for C projects. This project has been developed in Java. The proposed LWH approach accepts a C source project as the input. LWH approach identify and separates the functions/methods present in input. Having identified the methods, different source code metrics is computed for each method and stored . With the help of these metric values the near equal methods are extracted and are subjected to textual comparison to detect potential clone pairs.The overall process is carried out in three major stages: Pre-processing, detection and post-processing.

2.4 Feasibility Study

LWH (Light Weight Hybrid) approach is a combination of textual comparison and metrics computation. As there is no need for external parsing, this approach is of light weight. The proposed system is built in order to detect all four types of clone. The existing techniques has many difficulties in detecting all four types of clone.The idea behind the proposed system is to solve the drawbacks of the existing techniques.

2.4.1 Economic Feasibility

The project involves the utilization of software and Jdk1.6.0. The proposed system is economically feasible which will help to detect code clone. LWH approach is lightweight.On detecting code duplication, the quality of code is improved. This approach detects a significant amount of code duplication. Identification and subsequent unification of simple clones is beneficial in software maintenance. As a structural clone comprises many simple clones, structural clones are bigger in size and smaller in number than simple clones.

2.4.2 Operational Feasibility

The proposed approach will take source code of C projects as input. It detect clone in source code of medium sized C project of 11,000 lines to a large sized C project with 6,265,000 lines.The combination of texual comparison and metric computation provides to assess quality. We consider this as an indication that metric based and texual based approach together can be used to assess and assure quality of software.

2.4.3 Technical Feasibility

The propose a LWH (Light Weight Hybrid) approach combining textual analysis and metrics for the detection of method-level syntactic and semantic clones in C projects. This approach has been experimenting for the detection of all four types of clones by a specific set of metrics assessment and textual comparison. A LWH approach has been developed in Java to detect the clones in C file.LWH approach is lightweight.On detecting code duplication, the quality of code is improved.So user can easily detect clones in C projects.

2.5 Risk Analysis

Risk analysis and management are a series of steps that help a software team to understand and manage uncertainty. Many problems can plague a software project. A risk is a potential problem might happen, it might not. But, regardless of the outcome, it is really a good idea to identify it, assess its probability of occurrence, estimate its impact, and establish a contingency plan should the problem actually occur.

2.5.1 Software Risk

Although there has been considerable debate about the proper definition for software risk, there is general agreement that risk always involves two characteristics.

1. Uncertainty-the risk may or may not happen; that is, there are no 100.

2. Loss-if the risk becomes a reality, unwanted consequences or losses will occur.

When risks are analyzed, it is important to quantify the level of uncertainty and the degree of loss associated with each risk. To accomplish this, different categories of risks are considered.

2.5.2 Project Risk

Threaten the project plan. That is, if project risks become real, it is likely that project schedule will slip and that costs will increase. Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, customer, and requirements problems and their impact on a software project. In our project, project risk occurs if our requirement of technical member means technical team is unavailable according to our project plan and estimation and if our project is not completed within time in this situation project risk can occurs.

2.6 Project Scheduling

Following figure shows the schedule of the project. It gives the details of how much time is required to complete each and every phase of software engineering for this project.

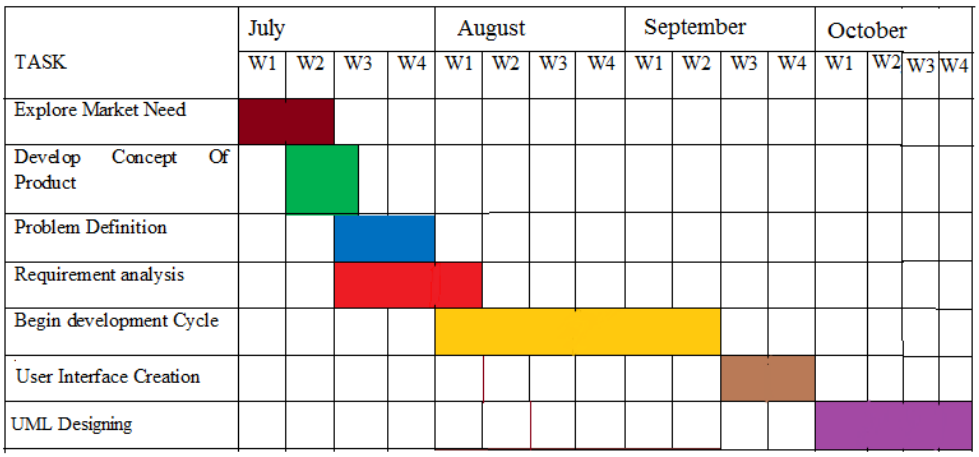

Figure 2.1: Project Scheduling

2.7 Effort Allocation

Following figure shows the efforts and involvement of every member in this project

Table 2.3: Effort Allocation

	Sayali Patil	Ashwini Sonawane	Sonal Saluke	Sachin Chaudhari	Makarand Bhole
Project Planning	20%	20%	20%	20%	20%
Requirement Gathering	25%	10%	20%	25%	20%
Design	25%	20%	15%	25%	15%

2.8 Summary

In this chapter, literature review of related work in clone detection , existing system, proposed system and feasibility study of the project are presented. In next chapter hardware and software requirement of the system are presented.

Chapter 3

System Requirements

System requirement specifications chapter will provides various requirements of the project such as functional, nonfunctional, software and hardware requirements. Understanding user requirements is an integral part of information systems design and is critical to the success of interactive systems.

It is now widely understood that successful systems and products begin with an understanding of the needs and requirements of the users. User-centered design begins with a thorough understanding of the needs and requirements of the users. The benefits can include increased productivity, enhanced quality of work, reductions in support and training costs, and improved user satisfaction.

In Section 3.1 the hardware requirements of the system are presented. Software requirements are presented in Section 3.2. In Section 3.3 functional requirements of system are presented. Non functional requirements are presented in Section 3.4.

3.1 Hardware Requirements

The hardware requirement includes a system with following configurations:

1. Processor: Intel(R)Core(TM)2 Duo and Intel Core(TM) i3
2. Hard-disk: 40GB
3. RAM: 2GB (2048MB)
4. Monitor : 15 VGA color
5. Mouse: Logitech.

3.2 Software Requirements

The various software requirements of the system are summarized here:

1. Operating system : Windows 7/8
2. Front end : Java
3. Java version : Jdk1.6.0

3.3 Functional Requirements

In software engineering, a functional requirement defines a function of a software system or its component. A function is described as a set of inputs, the behavior, and outputs. Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. The system provides the following functions :-

1. Clone Detection
2. Type of clone.
3. Clone pair

3.4 Non-Functional Requirements

A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. This should be contrasted with functional requirements that define specific behavior or functions. The plan for implementing functional requirements is detailed in the system design. The plan for implementing non functional requirements is detailed in the system architecture. The non functional requirements are:-

1. LWH approach flowchart
2. Clone Detection Algorithm

3.5 Summary

In this chapter, hardware and software requirement, functinal and non functional requirements of system are presented. In next chapter designing part of the code clone detection system and all UML, data flow diagrams, and architecture of the proposed approach are presented.

Chapter 4

System Design

System Design chapter will provides graphical structure of the project by using various UML diagrams. System design provides the understanding and procedural details necessary for implementing the system recommended in the system study.

Design is a meaningful engineering representation of something that is to be built. It can be traced to a customers requirements and at the same time assessed for quality against a set of predefined criteria for good design. In the software engineering context, design focuses on four major areas of concern are data, architecture, interfaces and components.

This chapter describe the design of the system. In Section 3.1 block diagram of proposed approach is presented. Data flow diagram of project is presented in Section 3.2. In Section 3.3 all the UML diagrams including use case diagram, sequence diagram, class diagram component diagram, statechart diagram, activity diagram, deployment diagram of the project are presented.

4.1 Proposed System Flow

The proposed LWH approach for automatic detection of function clones in C source code. A LWH approach has been developed in Java. This project accepts a C source project as the input.Then identify and separates the functions/methods present in it. Having identified the methods, different source code metrics is computed for each method and stored in a database. With the help of these metric values the near equal methods are extracted and are subjected to textual comparison to detect potential clone pairs [4].

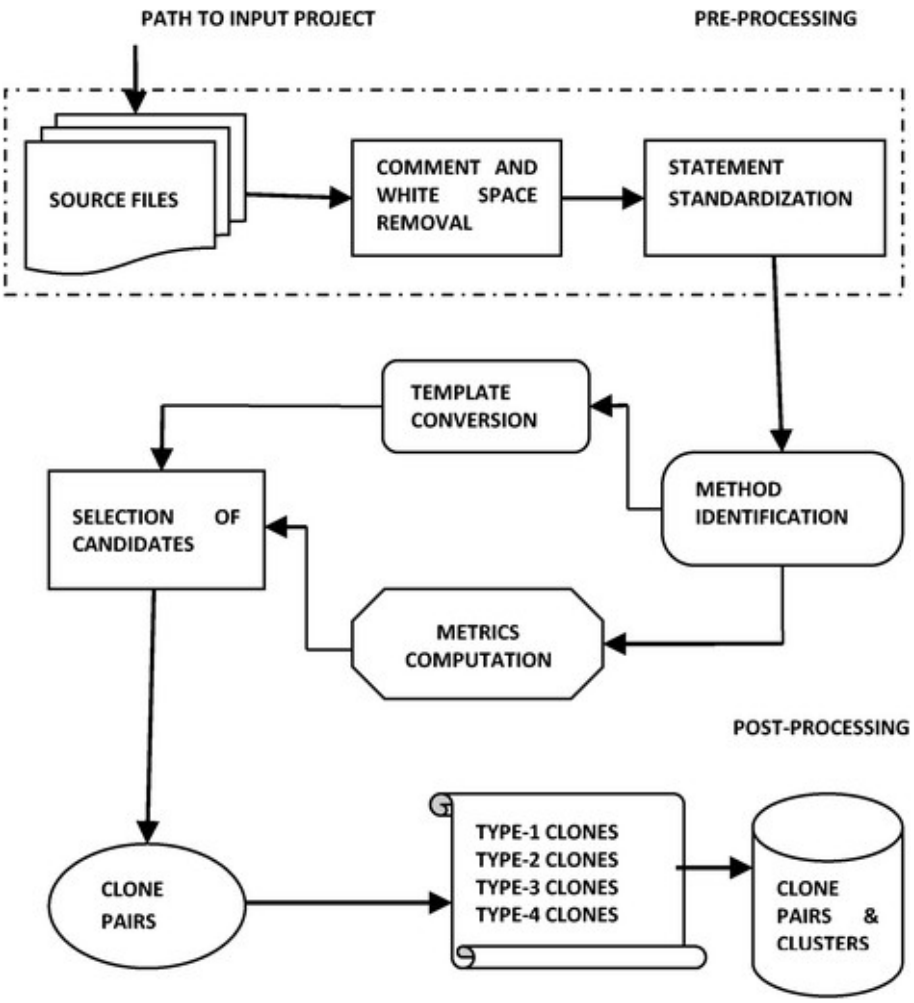

Figure 4.1: Clone Detector

4.2 Data Flow Diagram

A DFD is a graphical technique that depicts the information ow and the transformation that we have applied as the data moves from input to output. The data flow diagram also known as data flow graph or bubble chart. A data flow diagram may be used to represent a system or software at any level of abstraction. The data flow diagram can be completed using only four simple notations i.e. special symbols or icons and the annotation that with a special system. A data flow diagram (DFD) is a graphical technique that describes information about flow and that are applied as data moves from input to output. The DFD is also called as data flow graph or bubble chart. Named circles show the processes in DFD or named arrows entering or leaving the bubbles represent bubbles and data flow. A rectangle represents a source or sink and is not originate or consumer of data. Data flow diagrams are the basic building blocks that define the flow of data in a system to the particular destination and difference in the ow when any transformation happens. It makes whole procedure like a good document and makes simpler and easy to understand for both programmers and non-programmers by

dividing into the sub process. The data ow diagram serves two purposes:

- To provide an indication of how data are transform as the moves through the system.
- To depict the function that transforms the data flow.

4.2.1 Level 0

Context level DFD, also known as level 0 DFD, sees the whole system as a single process and emphasis the interaction between the system and external entities.

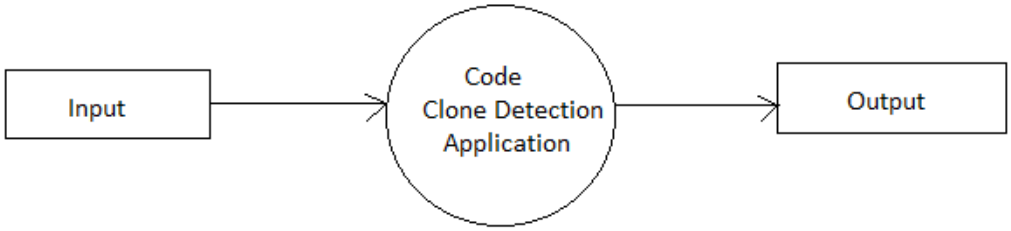

Figure 4.2: DFD Level 0

4.2.2 Level 1

Level 1 DFD's aim to give an overview of the full system. DFD look at the system in more detail. Major processes are broken down into sub-processes. Level 1 DFD's also identifies data stores that are used by the major processes. Level 1 DFD is shown in Figure 4.3.

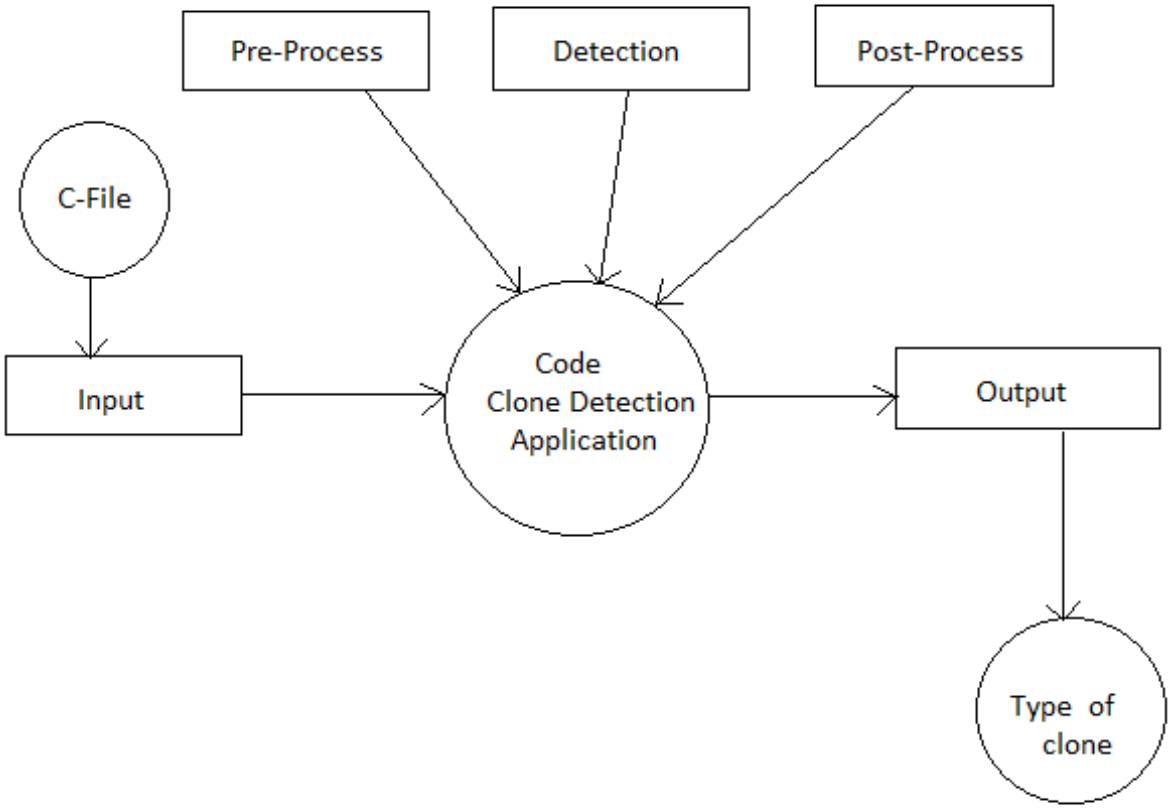

Figure 4.3: DFD Level 1

4.2.3 Level 2

Level 2 DFDs aim to give an detailing of the full system. They look at the system in more detail. Major processes are broken down into sub-processes. Level 1 DFD is shown in Figure 4.4.

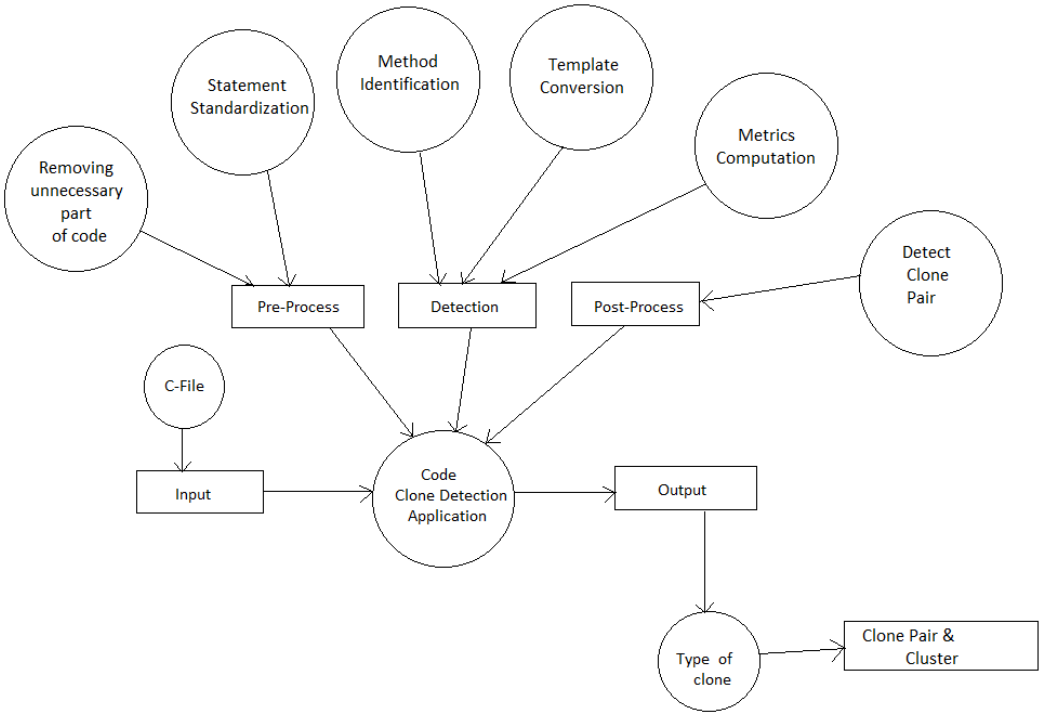

Figure 4.4: DFD Level 2

4.3 UML Diagram

4.3.1 Use Case Diagram

A Use case diagram shows a set of use cases and actors and their relationships. Use case diagrams address the static use case view of a system. These diagrams are especially important in organizing and modeling the behaviors of a system. Clone Detector module of project helps in finding clone. It is main module of project. Use case diagram for clone detector is shown in Figure 4.5.

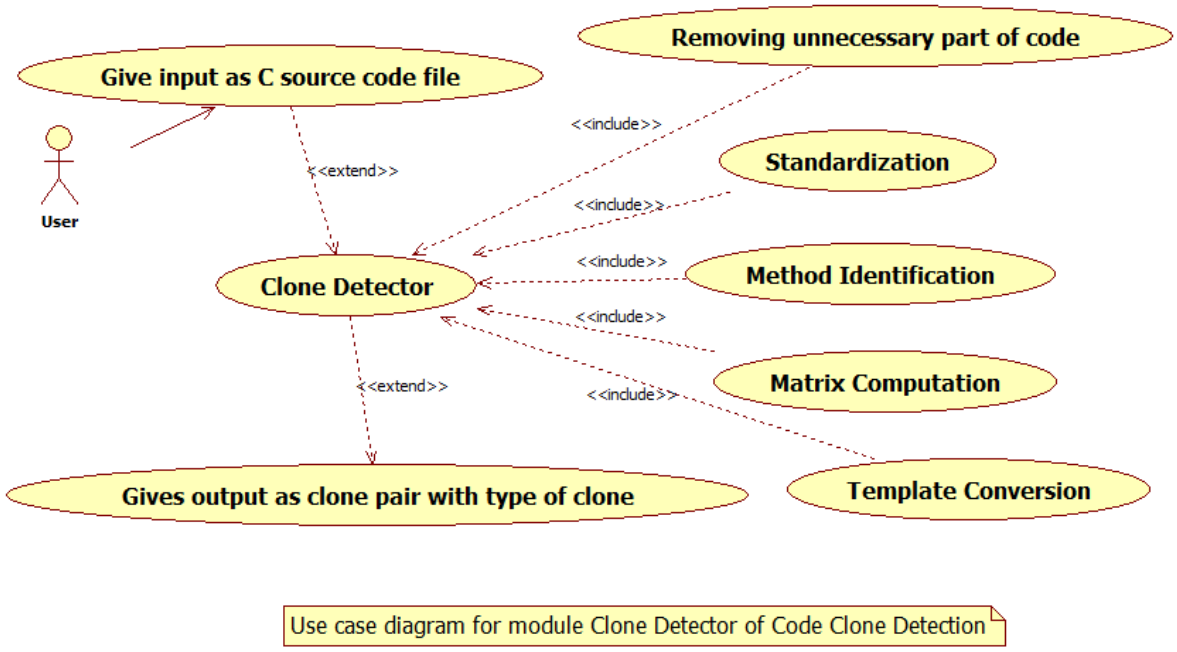

Figure 4.5: Use Case Diagram for Clone Detector

- *Use Case Diagram for Method Identification*

Method Identification module identify and seperate methods from each source file. This will be give as input to next step.

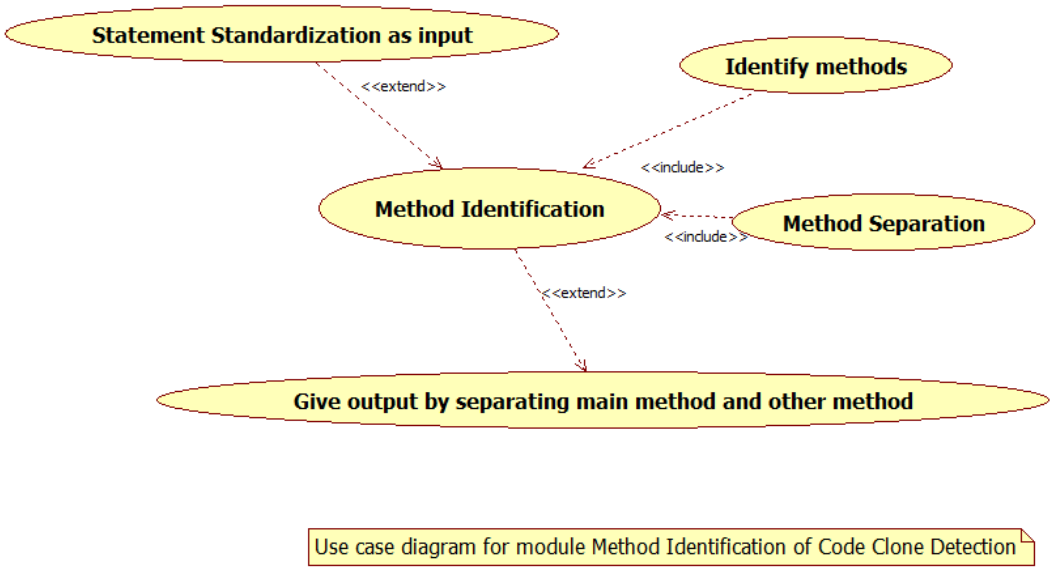

Figure 4.6: Use Case Diagram for Metrics Computation

- *Use Case Diagram for Metrics Computation*

Metrics Computation module compute the matric value for each method which will help in finding type of clone. The type of clone return by the Matric Computation module is based on similarity between various matrics that are computed for each method.

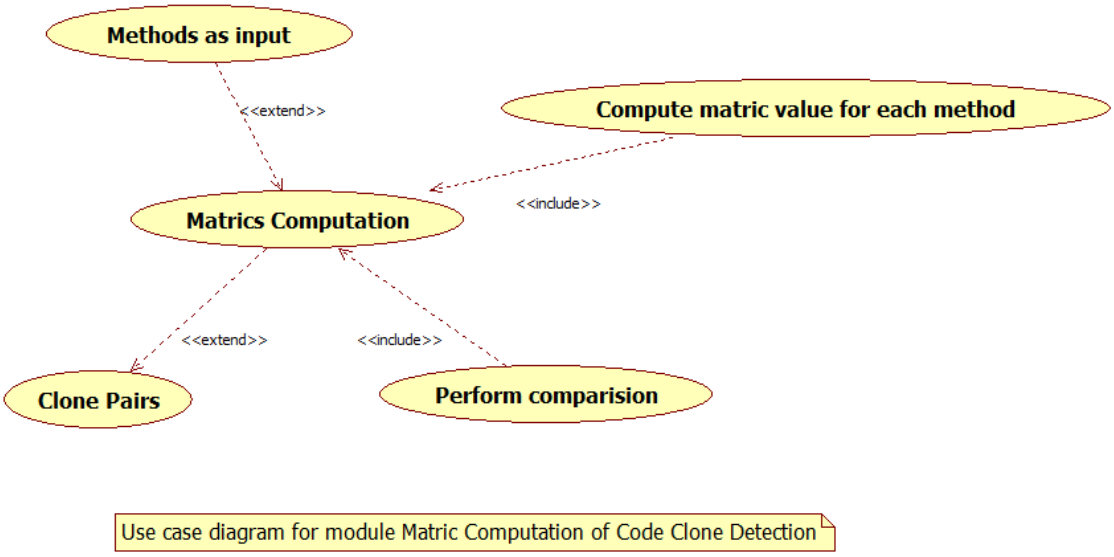

Figure 4.7: Use Case Diagram for Metrics Computation

- *Use Case Diagram for Template Conversion*

Template Conversion module convert original source code into new form. This module perform template conversion for all four type of clone.

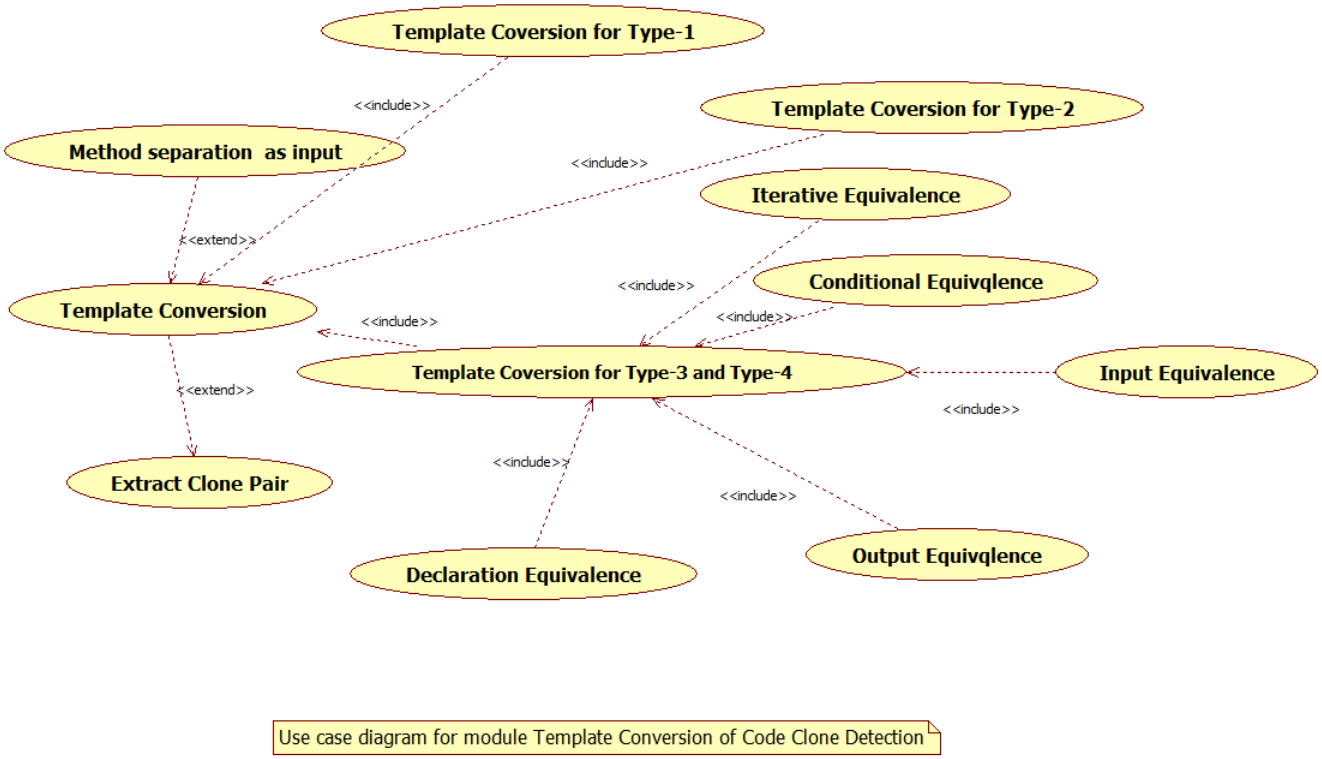

Figure 4.8: Use Case Diagram for Template Conversion

4.3.2 Sequence Diagram

Sequence diagram is kind of interaction diagram. An interaction diagram shows an interaction, consisting of a set of objects and their relationships. They address the dynamic view of a system.

- A sequence diagram is an interaction diagram that emphasizes the time-ordering of messages.
- A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive massages.

- *Sequence Diagram for Clone Detector Module*

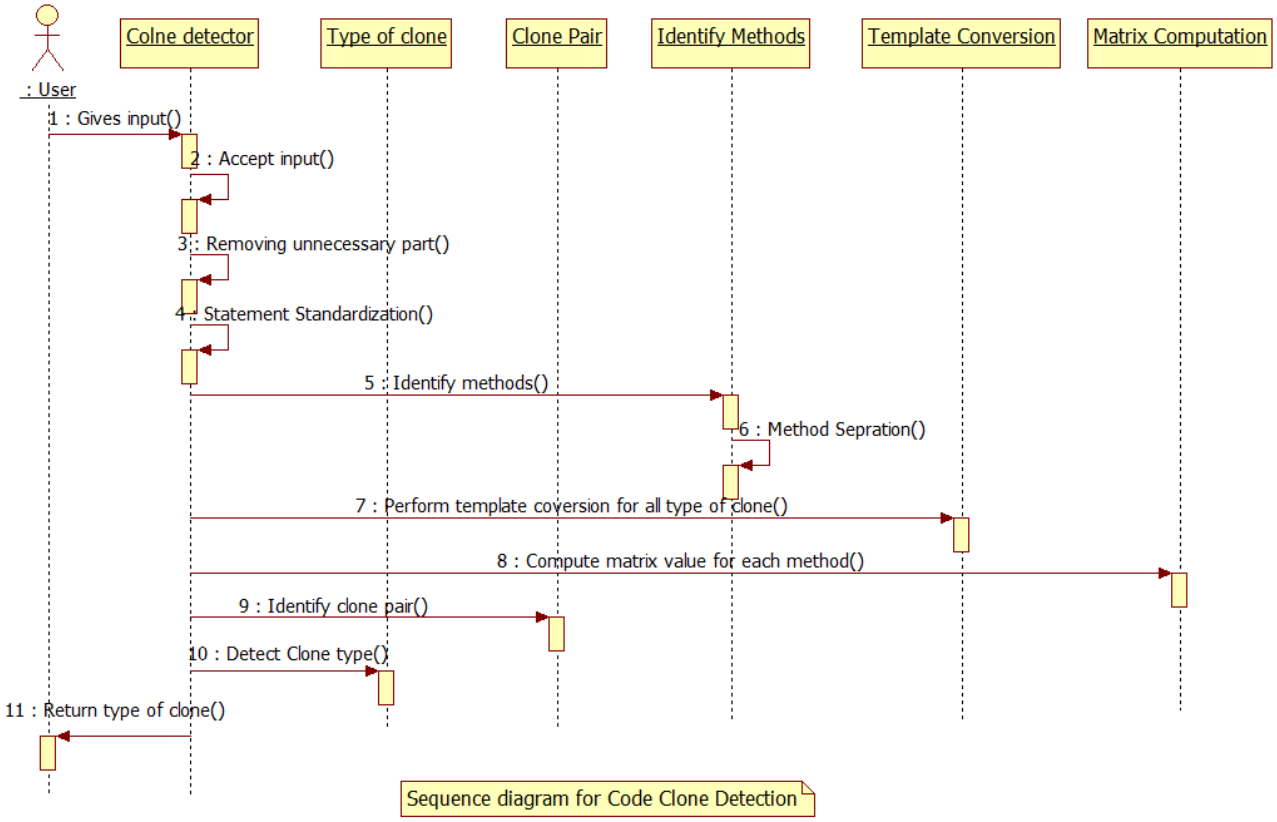

Figure 4.9: Sequence Diagram for Clone Detector Module

- *Sequence Diagram for Method Identification*

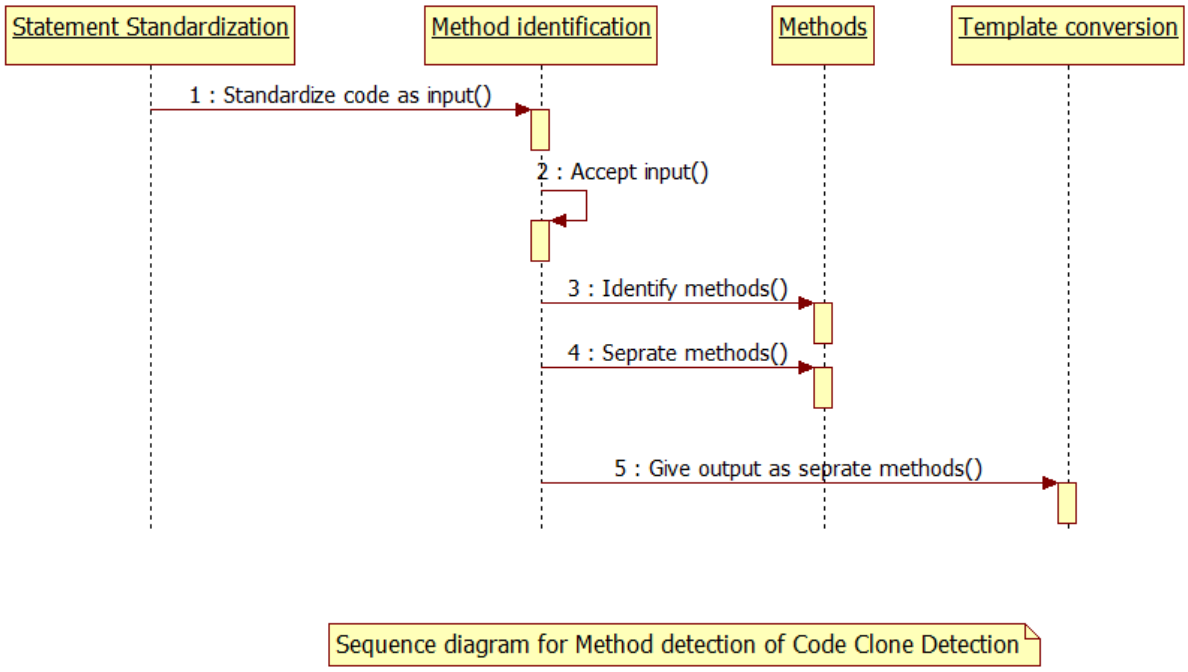

Figure 4.10: Sequence Diagram for Method Identification

- *Sequence Diagram for Template Conversion*

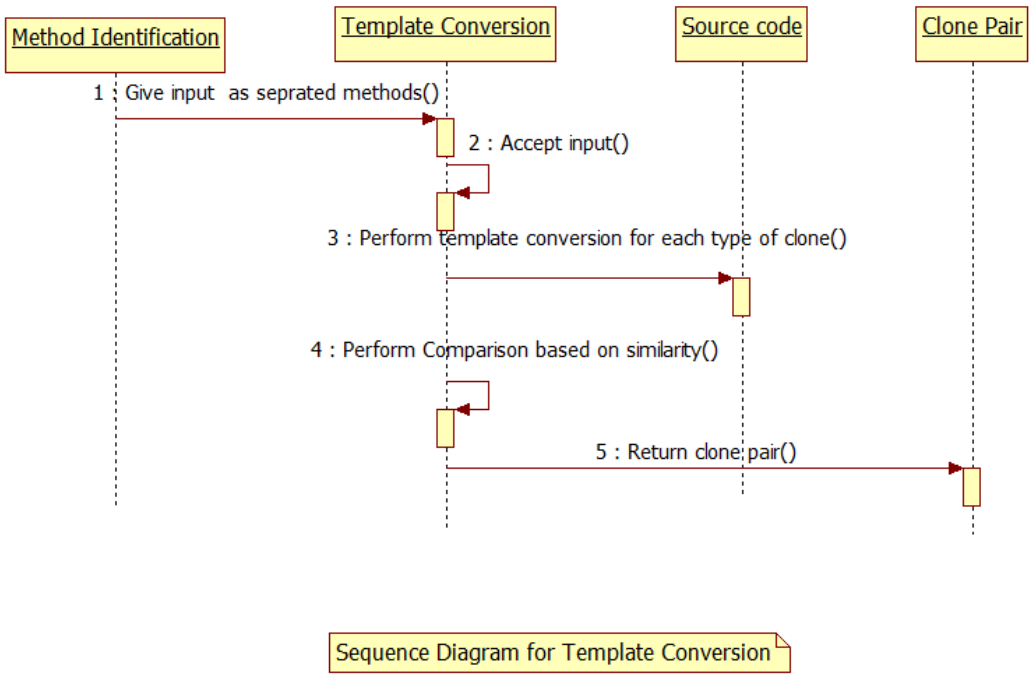

Figure 4.11: Sequence Diagram for Template Conversion

- *Sequence Diagram for Matrics Computation*

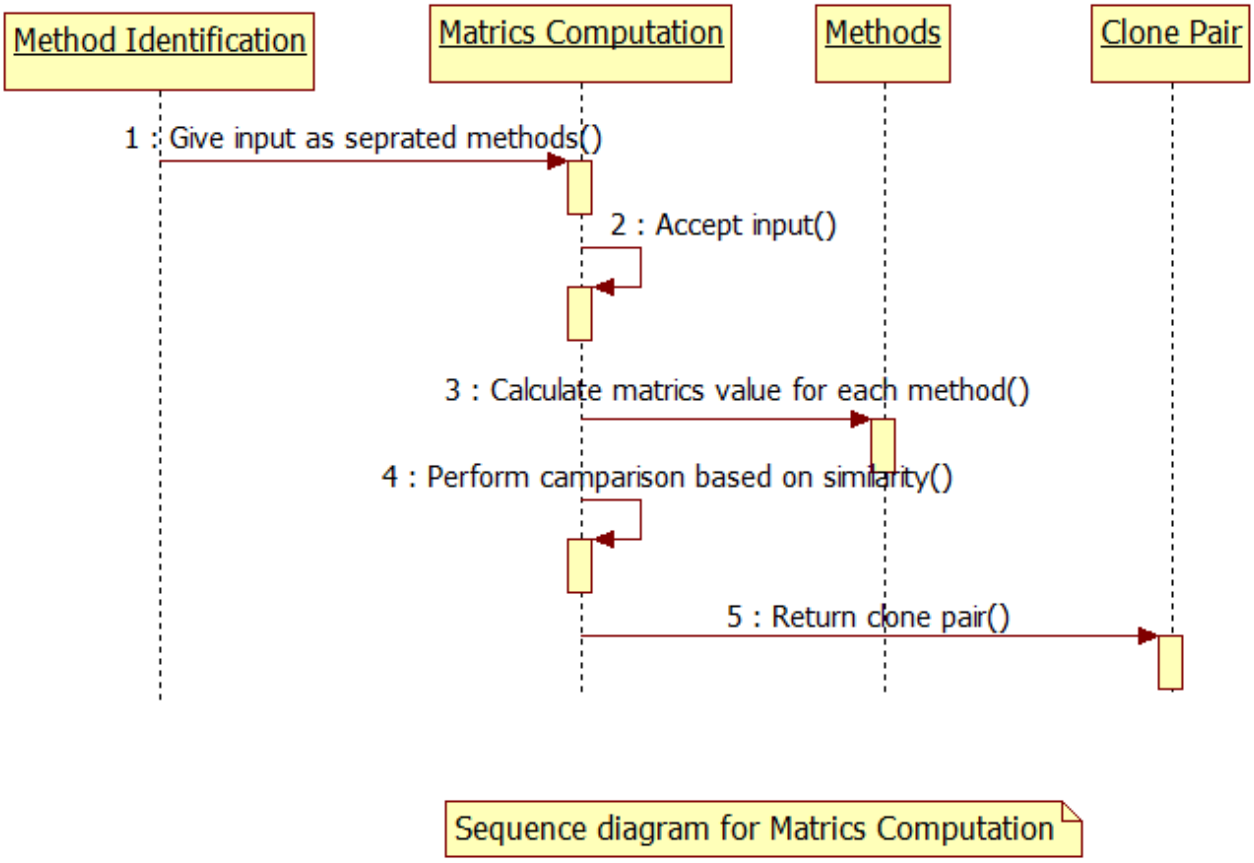

Figure 4.12: Sequence Diagram for Matrics Computation

4.3.3 Class Diagram

A Class diagram shows a set of classes, interfaces and collaborations and their relationships. These diagrams are the most common diagram found in modeling object-oriented systems. Class diagram address the static design view of a system. Following figure shows the class diagram of the proposed system.

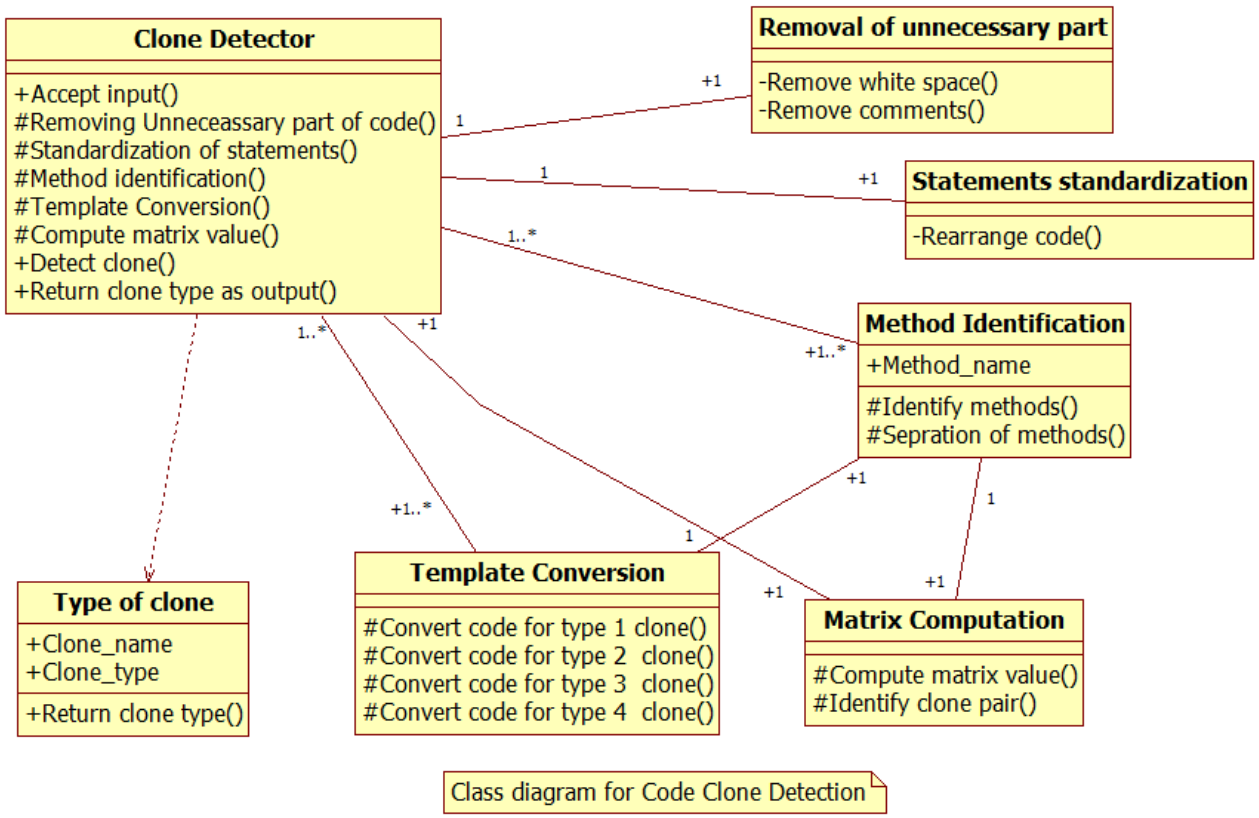

Figure 4.13: Class Diagram for Code Clone Detection

4.3.4 Statechart Diagram

Statechart diagram is use to show behaviour of system in case of some external events. In statechart diagram, a state is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event. An initial is a kind of pseudostate that represents the starting point in a region of a state machine. It has a single outgoing transition to the default state of the enclosing region, and has no incoming transitions. There can be one (and only one) initial state in any given region of a state machine. It is not itself a state but acts as a marker. A final state represents the last or "final" state of the enclosing composite state. There may be more than one final state at any level signifying that the composite state can end in different ways or conditions. When a final state is reached and there are no other enclosing states it means that the entire state machine has completed its transitions and no more transitions can occur.

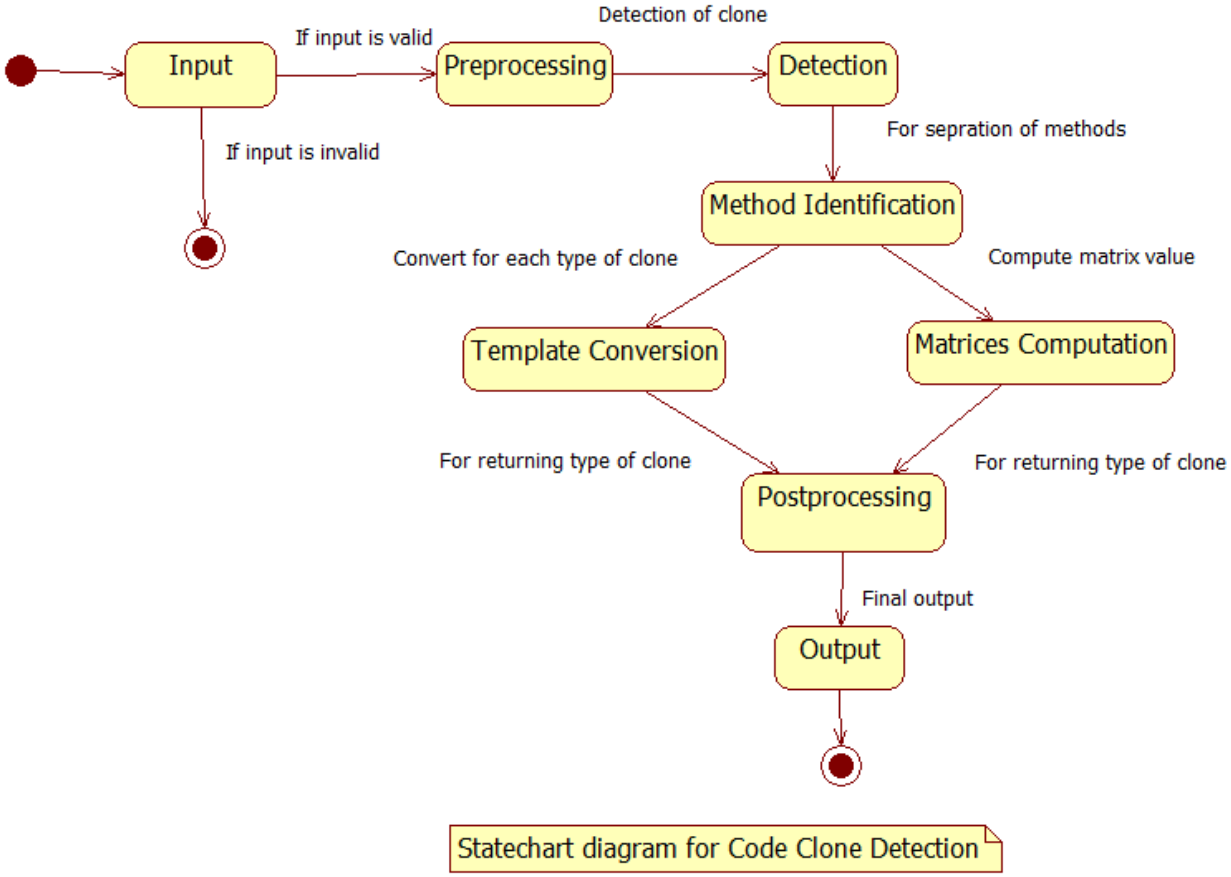

Figure 4.14: State Diagram for Clone Detector Module

4.3.5 Activity Diagram

Activity diagram shows the basic activities between two immediate states of statechart diagram. In activity diagram, an action state represents the execution of an atomic action, typically the invocation of an operation. An action state is a simple state with an entry action whose only exit transition is triggered by the implicit event of completing the execution of the entry action. The state therefore corresponds to the execution of the entry action itself and the outgoing transition is activated as soon as the action has completed its execution.

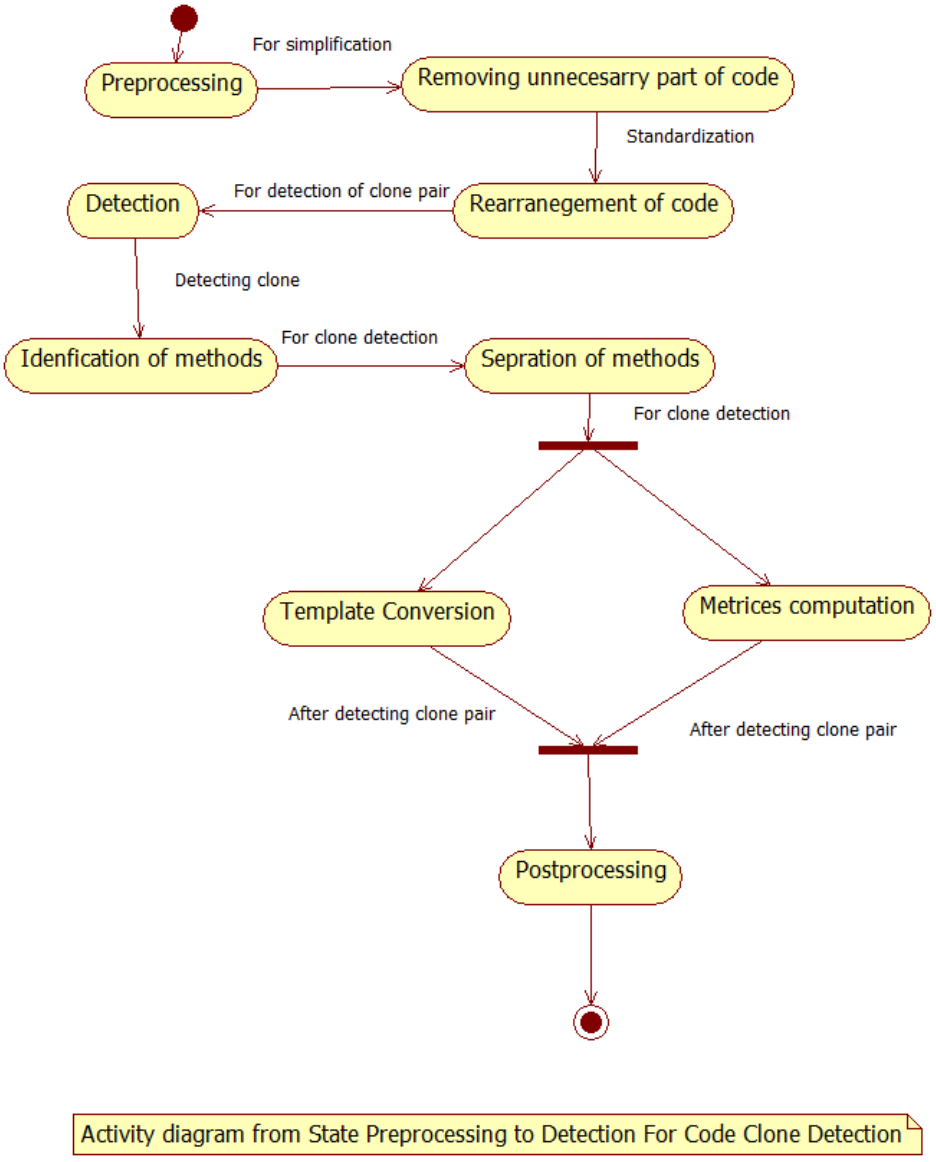

Figure 4.15: Activity Diagram

4.3.6 Component Diagram

A component diagram shows the organization and dependencies among a set of components. Component diagrams address the static implementation view of a system. In component diagram, a component represents a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.The component diagram for the proposed system is shown in following figure:-

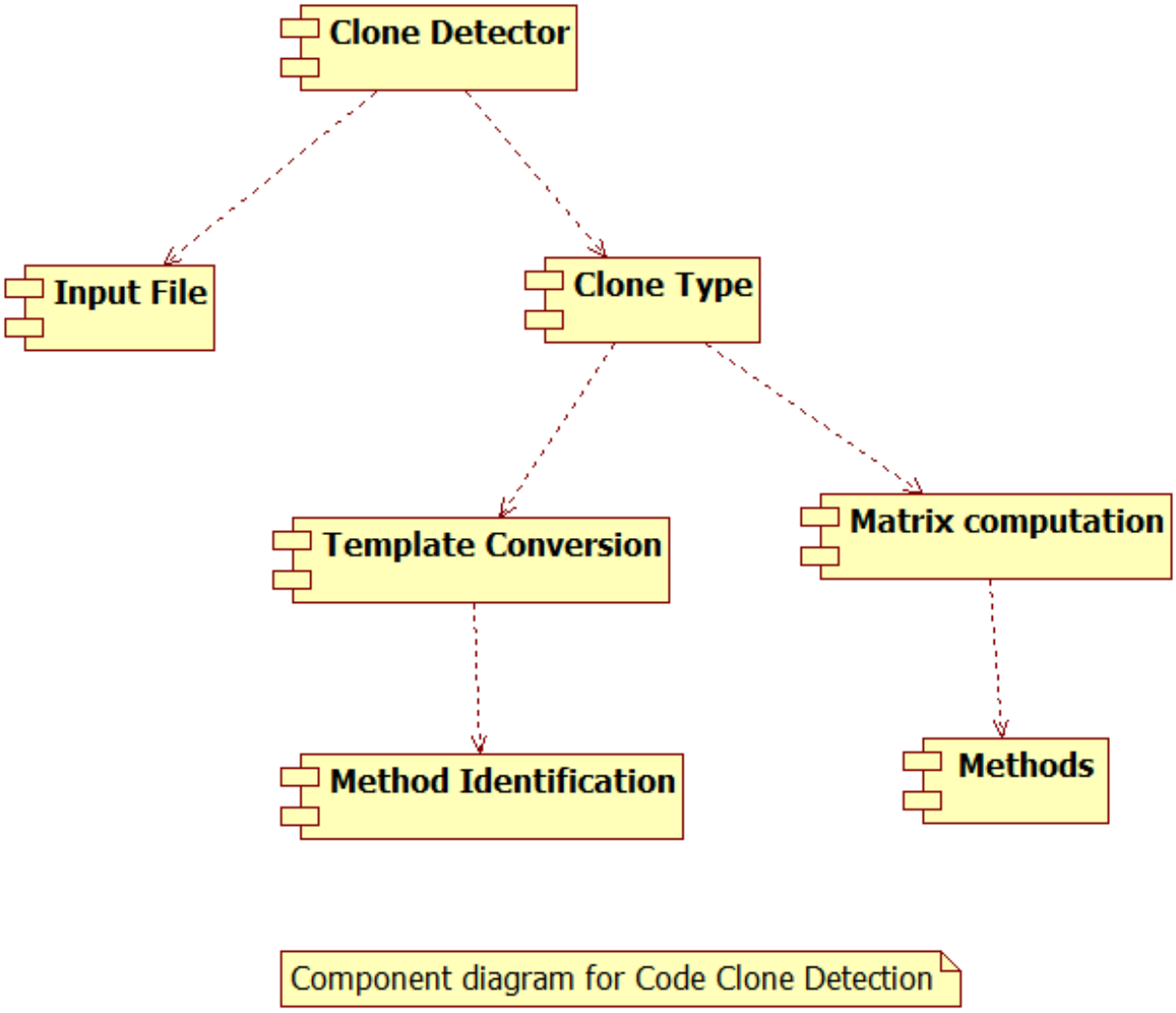

Figure 4.16: Component Diagram

4.3.7 Deployement Diagram

A deployment diagram shows the configuration of runtime processing nodes and the components that live on them. Deployment diagram address the static deployment view of an architecture. Deployment diagram for the proposed system is shown in figure:-

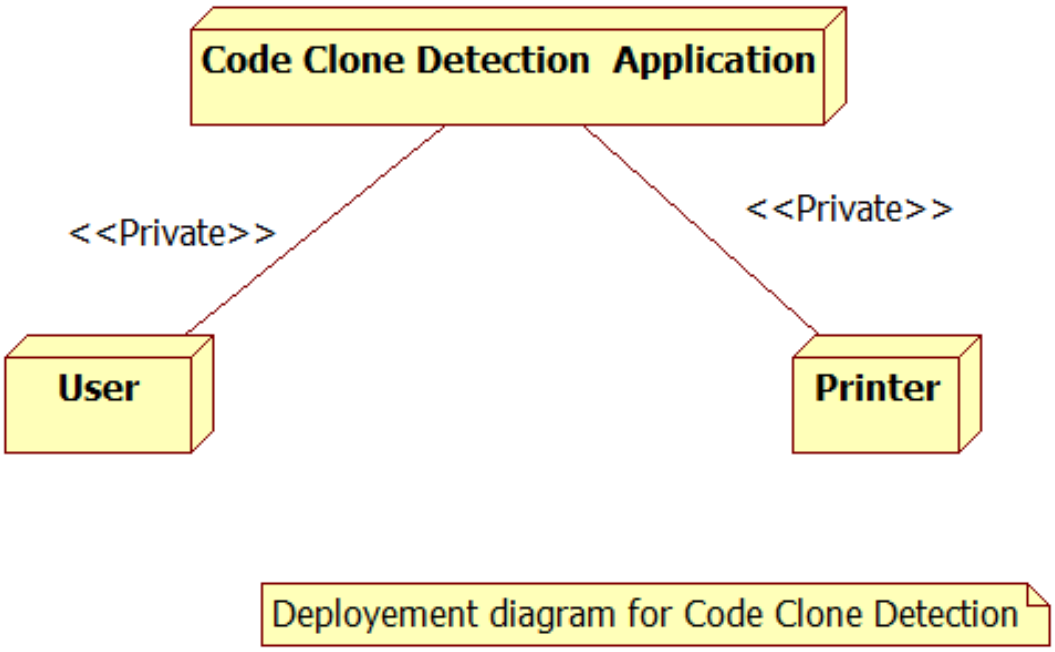

Figure 4.17: Deployment Diagram

4.4 Applications

In addition to the direct benefit of knowing how to improve the quality of the source code, there are several applications of detecting clones.

- Code fragment proves its usability by coping and reusing multiple times in the system that can be incorporated in a library and announce its reuse potential officially [6].
- In aspect mining research:-

  Detecting code clone is also necessary in aspect mining to detect cross-cutting concerns. The code of cross-cutting concerns is typically duplicated over the entire application that could be identified with clone detection tools [3].
- In software companies for:

  1. Detecting malicious software.
  2. Detecting plagiarism copyright content.
  3. Software evolution.

4.5 Advantages

Code clone duplication has many advantages in the development of software project. Some of them are discussed as-

1. **Detects library candidates:**

   Code fragment proves its usability by coping and reusing multiple times in the system that can be incorporated in a library and announce its reuse potential officially.

2. **Understanding program:**

   It is possible to get an overall idea of other files containing similar copies of the fragment, if the functionality of a cloned fragment is understood [6]. For example, when we have a piece of code managing memory we know that all files that contain a copy must implement a data structure with dynamically allocated space.

3. **Helps aspect mining research:**

   Detecting code clone is also necessary in aspect mining to detect cross-cutting concerns. The code of cross-cutting concerns is typically duplicated over the entire application that could be identified with clone detection tools [3].

4. **Finds usage patterns:**

   The functional usage patterns of the cloned fragment can be discovered if all the cloned fragments of the same source fragments are detected.

5. **Detects malicious software:**

   To detect malicious software clone detection techniques can play a vital role [2]. By comparing one malicious software to another, it is possible to find the evidence where parts of the one software system match parts of another.

6. **Helps Detecting plagiarism copyright content:**

   Finding similar code may also useful in detecting plagiarism and copyright infringement.

7. **Software evolution:**

   Clone detection techniques are successfully used in software evolution analysis by looking at the dynamic nature of different clones in different versions of a system [5].

8. **Code compacting:**

   Clone detection techniques can be used for compact device by reducing the source code size.

4.6 Disadvantages

Apart from benefits of code clones, it has severe impact on the quality, reusability and maintainability of a software system. The following are the list of some drawbacks of having cloned code in a system.

1. **Increased probability of bug propagation:**

   If a code segment contains a bug and that segment is reused by coping and pasting without or with minor adaptations, the bug of the original segment may remain in all the pasted segments in the system and therefore, the probability of bug propagation may increase significantly in the system [1].

2. **Increased probability of introducing a new bug:**

   In many cases, only the structure of the duplicated fragment is reused with the developer's responsibility of adapting the code to the current need. This process can be error prone and may introduce new bugs in the system.

3. **Increased resource requirements:**

   Code duplication introduces higher growth rate of the system size. While system size may not be a big problem for some domains, others (e.g., telecommunication switch or compact devices) may require costly hardware upgrade with a software upgrade [3]. Compilation times will increase if more code has to be translated which has a detrimental effect on the edit-compile-test cycle.

4. **Increased difficulty in system upgradation:**

   Because of duplicated code in the system, one needs additional time and attention to understand the existing cloned implementation and concerns to be adapted, and therefore, it becomes difficult to add new functionalities in the system, or even to change existing ones [3].

4.7 Summary

In this chapter, the designing of the proposed system including data flow diagrams, UML diagrams,and also applications and advantage, disadvantage of the proposed system are presented. In next chapter conclusion of proposed approach is presented.

Chapter 5

Conclusion

Code clone is a big problem. A copy and paste activity which is done by programmer is the main reason of code cloning. It looks like a simple and effective method, these copy and paste activities are not documented. Which create a bad effect on the software quality and duplication also increase the bug probability and maintenance problem. The proposed approach is able to detect all four types of clones accurately with the precision and recall values ranging from 88% to 100%. Also the texual comparison with matrics computation approach is very effective approach as it discovered the clones and also helps in identifying the clones of each types.

Bibliography

[1] B. Baker. "a program for identifying duplicated code, in: Proceedings of computing science and statistics: 24th symposium on the interface". *International Journal of Computer Applications*, 24:49–57, 1992.

[2] Dr. M. Punithavalli D.Gayathri Devi. comparison and evaluation on metrics based approach for detecting code clone. *IJCSE*, 2(5):750, 2011.

[3] Prajila Prem. a review on code clone analysis and code clone detection.. *International Journal of Engineering and Innovative Technology*, 2(12), June 2013.

[4] Adamov R. Literature review on software metrics. *International Journal of Advanced Research in Computer Science and Software Engineering*, 1987.

[5] Chanchal Kumar Roy and James R. Cordy. a survey on software clone detection research. *School of Computing Queen's University at Kingston Ontario, Canada*, September 2007.

[6] Koschke R. Roy CK, Cordy JR. comparison and evaluation of clone detection techniques and tools: A qualitative approach. *International Journal of Computer Applications*, pages 470–495, 2009.

[7] Kodhai. E Rubala Sivakumar. code clones detection in websites using hybrid approach. *IJCA*, 48(13), June 2012.

Index

Approach, 14

Clones, 3
Cloning, 3
Code, 3
Comparison, 12
Computation, 5

Detection, 6

Fragment, 5
Functions, 5

Methods, 19
Metric, 10

Pairs, 10
Precision, 6
Program, 13

Quality, 14

Reacll, 6

Software, 3

Techniques, 10
Token, 11
Transformations, 13
Types, 14