HYBRID TECHNIQUE FOR OBJECT ORIENTED SOFTWARE CLONE DETECTION

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
In
Software Engineering

Submitted By
Yogita Sharma
(800931026)

Under the supervision of
Dr. Rajesh Bhatia
Professor
&
Mr. Raj Kumar Tekchandani
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTEMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2011

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Hybrid Technique for Object Oriented Software Clone Detection"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Dr. Rajesh Bhatia** and **Mr. Raj Kumar Tekchandani** and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Yogita

(Yogita Sharma)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Dr. Rajesh Bhatia)
Professor
Department of Computer
Science and Engineering
DCRU, MURTHAL
(SONEPAT)

(Mr. Raj Kumar Tekchandani)
Assistant Professor
Department of Computer
Science and Engineering
Thapar University, Patiala

Countersigned by

(Dr. Maninder Singh)
Head 29/1/12
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

First of all I would like to thank the Almighty, who has always guided me to work on the right path of the life.

This work would not have been possible without the encouragement and able guidance of my supervisors **Dr. Rajesh Bhatia** and **Mr. Raj Kumar Tekchandani**. I thank my supervisors for their time, patience, discussions and valuable comments. Their enthusiasm and optimism made this experience both rewarding and enjoyable.

I am equally grateful to **Dr. Maninder Singh**, Associate Professor & Head, Computer Science & Engineering Department, for motivation and inspiration that triggered me for the thesis work.

I will be failing in my duty if I don't express my gratitude to **Dr. Abhijit Mukherjee**, Director of the University, for making provisions of infrastructure such as library facilities, computer labs equipped with net facilities, immensely useful for the learners to equip themselves with the latest in the field.

I am also thankful to the entire faculty and staff members of Computer Science and Engineering Department for their direct-indirect help, cooperation, love and affection, which made my stay at Thapar University memorable.

Last but not least, I would like to thank my parents for their wonderful love and encouragement, without their blessings none of this would have been possible. I would also like to thank my brother and sister, since they insisted that I should do so. I would also like to thank my close friends for their constant support.

Abstract

Code cloning — duplication of source code is a major problem for large, industrial systems. The major consequence of cloning is that it risks the maintenance process. Cloning is the most basic means of software reuse. Code cloning has been very extensively used within the software development design community. Unofficial surveys carried out within large, long term software development projects suggest that 25-30% of the modules in this kind of system may have been cloned.

A software clone is a code fragment identical or similar to another in the source code. Clones are considered harmful for software maintenance and evolution, because it increases complexity of the system. Detection of software clones decreases software maintenance costs and increases understandability of the system. Many code clone detection techniques have been proposed to detect clones.

This thesis proposes a practical method, for detecting clones for large software system. The clones are detected by using hybrid approach. The novel aspect of the approach takes the advantage of metric based and text based techniques for clone detection.

The proposed work is divided into two stages. In the first stage potential clones are selected on the basis of the metric match and in the second stage potential clones are further processed by text-based technique to determine whether two potential clones really are clone of each other.

Table of Contents

List of Figures

List of Table

Chapter 1

Introduction

A huge number of applications created today, rely on object-oriented programming (OOP) languages. Object-oriented programming (OOP) is today the dominant paradigm in mainstream software development. As their requirement is increasing day by day they are becoming larger and complex. Large-scale software systems are expensive to build and, are even more expensive to maintain. Sometimes, developers take easier way of implementation by copying some fragments of the existing programs and use that code in their work. This kind of work is called code cloning.

1.1 Code Cloning

Definition: A code clone is a code portion in source files that is identical or similar to another [35].

Duplication of code occurs frequently during the development of large software systems. Code cloning is a form of software reuse, and exists in almost every software project. This ad-hoc form of reuse consists in copying, and eventually modifying, a block of existing code that implement a piece of required functionality. Duplicated blocks are called clones and the act of copying, including slight modifications, is said cloning.

The results of several studies [5] indicate that a considerable fraction (5-10%) of the source code in large software systems is duplicate code. Software clone is usually generated by programmer's copy and paste activities. Programmers often copy and paste an existing similar code and further modify it according to their need.

Code cloning or the act of copying code fragments and making minor, non – functional alterations, is a well-known problem for evolving software systems leading to duplicated

code fragments or code clones. The normal functioning of the system is not affected, but further development may become prohibitively expensive.

1.2 Why Code Duplication

There are a number of reasons why developers clone source code. Cloning mainly occurs because programmers find that it is cheaper and quicker to use the copy and paste feature than writing the code from scratch. Some times programmers intent on implementing new functionality find some working code that performs a computation nearly identical to the one desired, copy it entirely and then modify in place [7].

While this is actually good reuse practice, it complicates the maintenance process. Code cloning, is considered a serious problem in industrial software [10]. Duplicated code proves easy and cheap during the software development phase, but it makes software maintenance much harder. Software clone has a number of negative effects on the quality of the software. Besides increasing the amount of the code, which needs to be maintained, it also increases the bug probability.

So there is a need to detect the clones to figure out the problems and to help better software understandability and maintenance. Concerning the detection of duplicated code, numerous techniques have been successfully applied on industrial systems. These techniques can be roughly classified into following categories [7].

- String-based, i.e. the program is divided into a number of strings (typically lines) and these strings are compared against each other to find sequences of duplicated strings.
- Token-based, i.e. a laxer tool divides the program into a stream of tokens and then searches for series of similar tokens.
- Parse–tree based, i.e., after building a complete parse-tree one performs pattern matching on the tree to search for similar sub–trees.
- PDG based, i.e. after obtaining program dependency graph similar graphs are search.

- Metric based i.e. metrics are calculated form program and these are used to find duplicate code.
- Hybrid, i.e. detection techniques that use a combination of the other clone detection techniques.

1.3 Motivation & Objective

Maintenance cost of software is generally more than implementation cost. According to Brooks [6] more than 90% of software cost belongs to software maintenance activities. According to Roy et al. [8] between 7% up to 23% of software systems contain clone code.

Cloning is problematic for software maintenance for several reasons [7]:

- Cloning unnecessarily increases program size. Since many maintenance efforts correlate with program size, this increases the maintenance effort.
- Changes to one clone, such as bug fixing, typically need to be made to the other clones as well, again increasing maintenance effort.
- If changes to duplicated source code fragments are performed inconsistently, this can introduce bugs.

Code clone detection could be useful in many ways such as for the purpose of plagiarism detection, aspect mining as well as well as decreasing the cost of software maintenance activities. Detection of duplicate code fragments increases understandability of software systems and may help system maintainers to increase code quality of the existing system. Detection of duplicate code fragments leads to efficiency on the software maintenance process and decreases maintenance cost [7].

The aim of this thesis is to design and implement a tool for detecting clones. The novel aspect of the work is using metric and text based technique in detection process. In order to achieve this aim, the following objectives must be fulfilled.

- To study and compare the existing software clone detection techniques.
- To propose a noval clone detection technique for object oriented system.
- To validate the proposed technique.

1.4 Outline of the Thesis

The remaining of the report will be organized into the following division:

Chapter 2- Gives a review of code clone detection and various existing techniques for code clone detection.

Chapter 3- Specify problem statement and what is new in proposed work.

Chapter 4- Explain Proposed model and its implementation.

Chapter 5- Explains the experiments performed and evaluates the results achieved.

Chapter 6– Concludes the report and provides some suggestion for future work.

Chapter 2

Background Information and Literature Review

2.1 Basic Concept of Clone Detection

A code clone is two or more segments of code that are similar structurally, and have a similar intended use or semantics [2].

Definition 1: Code Fragment. A code fragment (CF) is any sequence of code lines (with or without comments). It can be of any granularity, e.g., function definition, begin end block, or sequence of statements. A CF is identified by its file name and begin-end line numbers in the original code base and is denoted as a triple (CF.FileName, CF.BeginLine, CF.EndLine) [8].

Definition 2: Code Clone. A code fragment CF2 is a clone of another code fragment CF1 if they are similar by some given definition of similarity, that is, f (CF1) = f (CF2) where f is the similarity function [8].

2.2 Reason of Code Cloning

Clones do not occur in the software themselves. There are various reasons that tend the developer to do cloning [7]. Figure 2.1 displays various factors for which clones can be introduced in the source code and a short description for some of the factors are discussed below:

a) Time Limit

One of the major causes of code cloning is that a certain time limit is assigned to developer to finish a project. To do this developers just copy and paste the existing one and adapt to their current need.

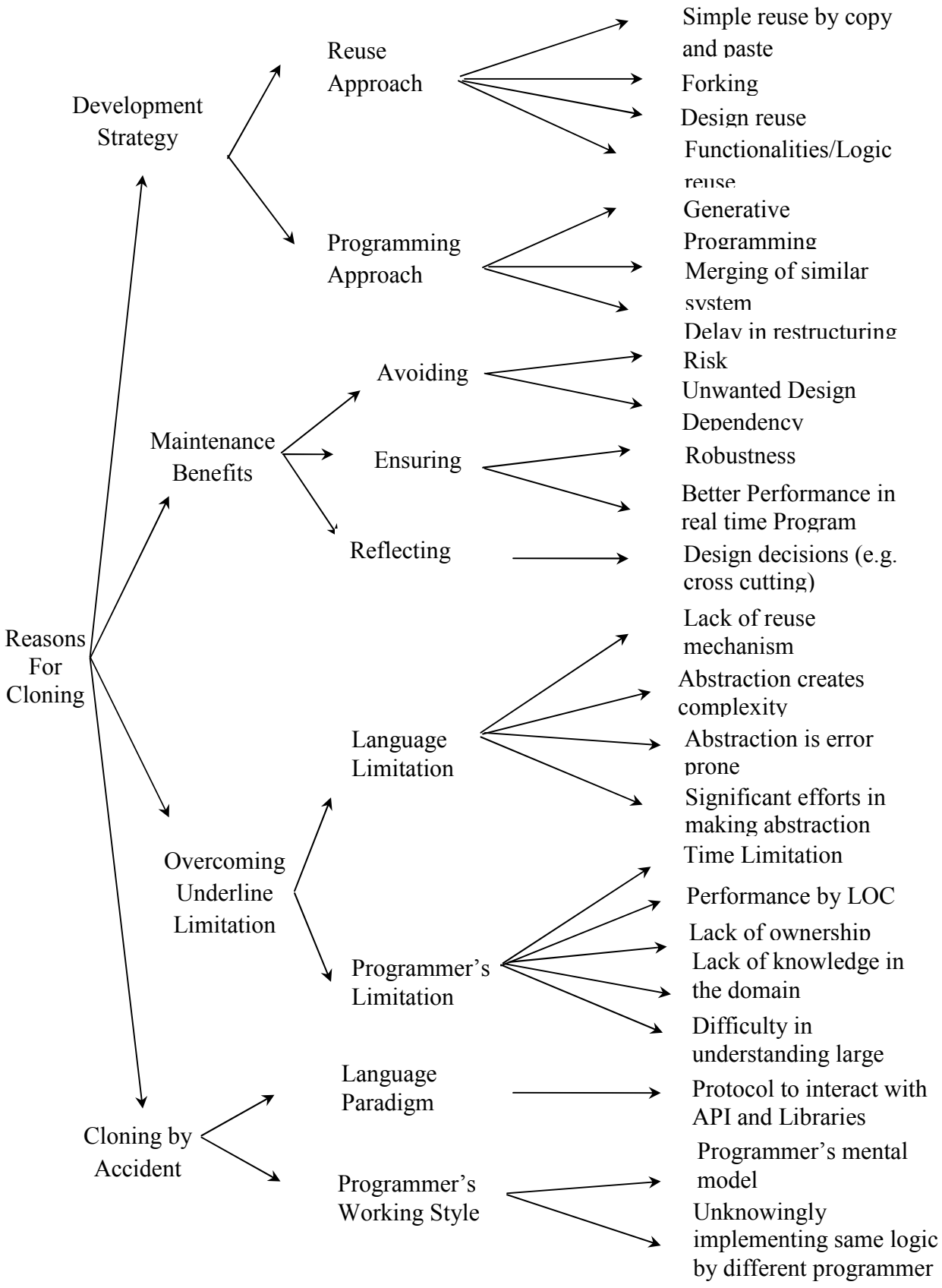


Figure 2.1 Reason of Cloning [7]

b) Language Limitation

Clones can be introduced due to the limitations of the language, especially when the language in question does not have sufficient abstraction mechanisms. Sometimes, the developers are forced to copy because of limitations of their knowledge in that particular programming language.

c) Difficulty in Understanding Large System

It is generally difficult to understand a large software system. These force the developers to use the example-oriented programming by adapting previously developed existing code.

d) Reuse

The prime reason of code duplication is reusing code, logic, design or an entire system. Reusing existing code by copying and pasting is the most common form of reuse mechanism in the development process which results code cloning.

e) By Accident

Code cloning may be accidentally. There may be a case that two software developers may come with same solution. Technically these are not clones since they were not intentionally copied from each other, but the clone detection tools identify them as clones since they look similar. Programmers may unintentionally repeat a common solution for similar kind of problems using the common solution pattern of his memory to such similar problems. Therefore, several clones may unknowingly be created to the software systems.

f) Developer's Performance

Sometimes the productivity of a developer is measured by the number of lines he produces per hour. In such circumstances, the developer's focus is to increase the number of lines of the system and hence tries to reuse the same code again and again by copying and pasting with adaptations instead of following a proper development strategy.

g) Risk in New Code

There is high risk of software error in new code fragments and because existing code is already tested and there is less risk of error , so the developer is often asked to reuse the existing code by copying and modify it according to the new product's requirements.

h) Merging of Two Similar Systems

Sometimes two software systems of similar functionalities are merged to produce a new one. Although these systems may have been developed by different teams, clones may produce in the merged system because of the implementations of similar functionalities in both systems.

2.3 Consequence of Code Cloning

While it is beneficial to practice cloning, software clone has a number of negative effects on the quality of the software. Code clones can have severe impacts on the quality, reusability and maintainability of a software system. Besides increasing the amount of the code which needs to be maintained, duplication also increases the defect probability and resource requirements [9]. The following list gives an overview of these problems.

a) Increased Maintenance Work and Cost

Because of duplicated code in the system, one needs additional time and attention to understand the existing code. When programmers maintain a piece of clone code, the changes should also perform on every other clone pairs. Since programmers who usually have no records of this duplicate code, the maintaining work should perform on the entire system. If a cloned code segment is found to be contained a bug, all of its similar counterparts should be investigated for correcting the bug in question, as there is no guarantee that this bug has been already eliminated from other similar parts at the time of reusing or during maintenance.

b) Increased Defect Probability

By simply copying a piece of code into a new context, which will cause the conflict between each other, e.g. conflict and clash between variables from the copied code and variables in the new context. Dependencies of copied code may also not be fully understood by the new context is another potential defect cause. Duplication of the source code also increases the probability of bug propagation in the system.

c) Increased Resource Requirement

The clone code will consume more compilation times, because more codes have to be compiled. It may also lead the upgrading of hardware resources, especially when the system is running in a tight hardware environment, e.g. telecommunication switch, which the software system upgrading will lead an upgrading in hardware as well.

d) Increased Probability of Bad Design

Code cloning effect the design of the system. Cloning may also introduce bad design, lack of good inheritance structure or abstraction. Consequently, it becomes difficult to reuse part of the implementation in future projects. It also has bad impact on the maintainability of the software.

2.4 Clone Detection Process

A clone detector must try to find pieces of code of high similarity in a system's source text. The main problem is that it is not known beforehand which code fragments can be found multiple times. The detector thus essentially has to compare every possible fragment with every other possible fragment. Such comparison is very expensive from a computational point of view and thus, several measures are taken to reduce the domain of comparison before performing the actual comparison. Once potential cloned fragments are identified further analysis is carried out to detect actual clones [8]. Figure 2.2 displays the phases of clone detection process.

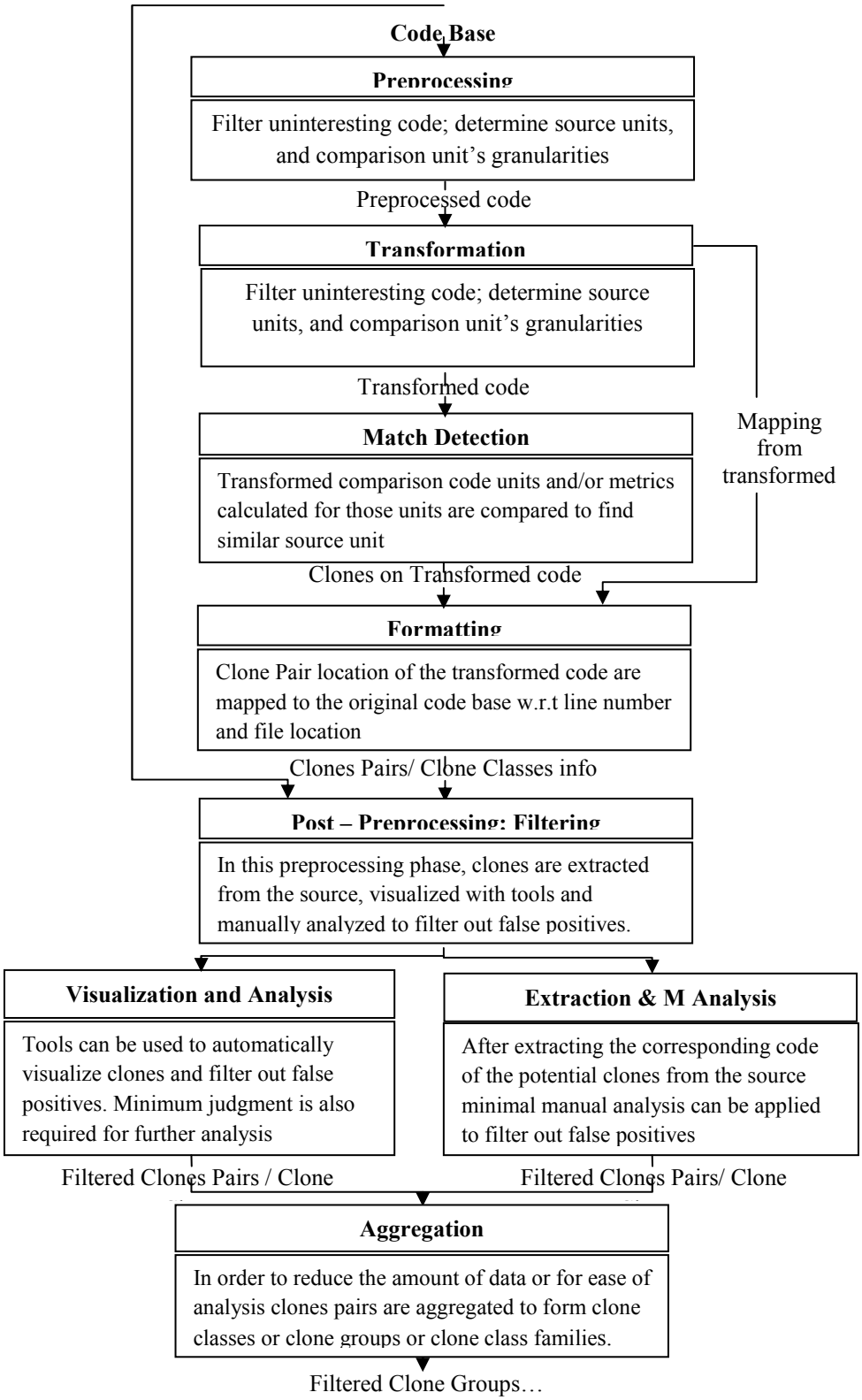


Figure 2.2 Clone Detection Process [8]

a) Preprocessing

Initially, in the clone detection process the target source is portioned and comparison domain is determined. The main objectives to be considered in this phase are removing uninteresting parts, determining the source units and determining the comparison unit. All the source code uninteresting to the comparison phase is filtered in this phase. After removing the uninteresting code, the remaining source code is partitioned into a set of disjoint fragments called source units. These units are the largest source fragments that are involved in direct clone relations to each other. Source units may need to be further partitioned into smaller units depending on the comparison function of a method.

b) Transformation

In the next stage, the source code's comparison unit is transformed to another intermediate internal representation of ease of comparison or for extracting the comparable properties. The transformation could be very simple by just removing the white space and comments [3] or could be very complex by generating PDG representation [31] or extensive source code transformation. Few transformation techniques are pretty printing of source code, removal of comments, removal of whitespace, tokenization, parsing, generating PDG, normalizing identifiers, transformation of program elements and calculating metric values etc. However, source units may themselves be used as comparison units.

c) Match Detection

The transformed code is given as input to a suitable comparison unit, where it is compared with each other in order to find the match. The order of comparison units are used, to sum up the adjacent similar units to form larger units. The output of the comparison unit is a list of matches with respect to the transformed code. These matches can be either the clone pair candidate or they have to be aggregated to form clone pair candidates. Then every clone pair is generally represented with the location information of the matched fragments in the transformed code.

d) Formatting

In this stage, the clone pair list obtained with respect to the transformed code is then converted to a clone pair list obtained with respect to the original code base. Normally, after finding the clone pair location from the previous phase, it is then converted into line numbers on the original source files.

e) Preprocessing

This stage helps out in filtering the false positives in two ways such as manual analysis and visualization tool

f) Manual Analysis

Once the original source code has been extracted, the raw code of the clones of the clone pairs subjected to the manual analysis, in order to filter out the false positive.

g) Visualization

To speed up the manual analysis in filtering out the false positives, visualization tool are used to visualize the clone pair.

h) Aggregation

Finally, to perform certain analysis, there is a need to reduce the amount of data, so the clone pairs should be aggregated to clusters, classes, cliques of clones or clone groups etc.

2.5 Clone Relation Terminology

Clone detection tools report clones in the form of Clone Pairs or Clone Classes [8]. Figure 2.3 displays the clone pair that can be grouped together by their common property into a clone class.

Clone Pair: If a clone relation exit between a pair of code fragment is called a Clone Pair.

A pair of code portions/fragments, which are identical or similar to each other, is a Clone Pair [8].

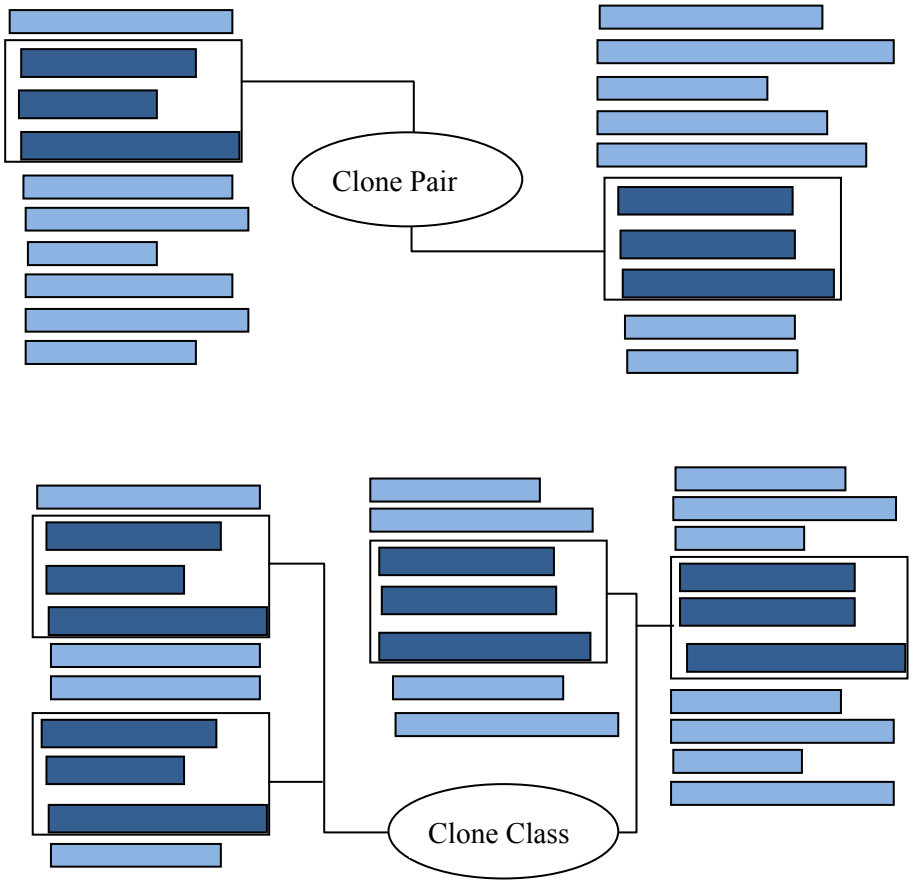


Figure 2.3 Clone Pair and Clone Class [38]

Clone Class: The maximal set of code portions/fragments in which any two of the code portions/fragments hold a clone-relation (form a clone Pair) is a Clone Class [8].

2.6 Types of Clone

There is no solid definition of what constitutes a clone [2]. The general answer of the question "what is clone?" is that two code fragments if they are identical or similar [34]. According to the different similarities, clone can be classified into two categories: One

type of similarity considers textual similarity and other second considers the semantic level, which the clone code must have the same behaviors, means the functional similarity.

Textual Similarity: Two code fragments can be similar based on the similarity of their program text. The following types of clones are discussed in order to find textual similarity [7].

Type I: In Type I clone, a copied code fragment is the same as the original. However, there might be some variations in whitespace (blanks, new line(s), tabs etc.), comments and/or layouts. Type I is widely known as exact clones

Type II: A Type II clone is a code fragment that is the same as the original except for some possible variations about the corresponding names of user-defined identifiers (name of variables, constants, class, methods and so on), types, layout and comments. The reserved words and the sentence structures are essentially the same as the original one.

Type III: Type III is copy with further modifications, e.g. a new statement can be added, or some statements can be removed. The structure of code fragment may be changed and they may even look or behave slight differently. This kind of clone is hard to be detected, because the fully context understanding is needed.

Functional Similarity: Two code fragments can be similar based on the similarity of their functionalities without being textually similar. If the functionalities of the two code fragments are identical or similar i.e., they have similar pre and post conditions referred as Type IV clones [7].

Type IV: Type IV clones are the results of semantic similarity between two or more code fragments. In this type of clones, the cloned fragment is not necessarily copied from the original. Two code fragments may be developed by two different programmers to

implement the same kind of logic making the code fragments similar in their functionality.

2.7 Advantages of Clone Detection

In addition to improve the quality of the source code by refactoring the cloned code, there are several other benefits of detecting clones [7]. The list of some of those is as follows:

i) Detect Library Candidates

It has noticed that a code fragment that has been copied and reused multiple times in the system apparently proves its usability. As a result, this fragment can be incorporated in a library, to announce its reuse potential officially.

j) Bug Detection

There is also a close relation between clone detection and software bug detection. Copy-pasted software bugs, especially, can be successfully detected by clone detection tools.

k) Program Understanding

Clone detection techniques may assist in understanding a software system. As clones hold important domain knowledge, one may achieve an overall understanding of the entire system by understanding the clones of a system. For example, Johnson [20] visualizes the redundant substrings to ease the task of comprehending large legacy systems. Program comprehension techniques, such as search-based techniques or concept analysis may greatly help clone detection research.

l) Code Compaction

Clone detection techniques can assist with fitting code into compact devices (e.g., mobile devices) by reducing source code size [7].

m) Find Usage Patterns

If all the cloned fragments of a same source fragment can be detected, the functional usage patterns of that fragment can be discovered.

This is the reason why the software clone detection gains considerable attention.

2.8 Clone Detection and its Application

In addition to the immediate applications of clone detection techniques to clone refactoring, avoidance and management, there are several other domains in which clone detection techniques seem helpful. There are also some other areas related to clone detection from which clone detection techniques themselves can get benefited [7]. Clone detection is useful in finding malicious software. By comparing one malicious software with another, it is possible to find the matched parts of one software system with another. Some applications of clone detection are as follows:

a) Plagiarism Detection in Projects

Plagiarism Detection is one of the closely related areas of clone detection [13]. Clone detection techniques can be used in the domain of plagiarism detection. A clone detection tool such as token-based CCFinder [34] has been applied in detecting plagiarism.

b) Copyright Infringement

The problem of detecting source code copyright infringement is viewed as a code similarity measuring problem between software systems. Clone detection tools can, therefore, be applied or can easily be adapted in detecting copyright infringement [5].

c) Clone Detection in Models

Clone detection is also used in models [11]. Phenomenon is not restricted to code, but occurs in models in a very similar way. So it is likely that model clones are as

detrimental to model quality as they are to code quality. Clone detection is also used in data flow model, General Model, UML Domain Model [12, 14].

2.9 Clone Detection Techniques

Code cloning detection had been an active research for almost two decades. Considering the importance of detecting clones, many clone detection techniques have been proposed in the literature and can be classified into following:

1) **Text – Based Technique**
2) **Token – Based Technique**
3) **Abstract Syntax Tree - Based Technique**
4) **Program Dependence Graph – Based Technique**
5) **Metric – Based Technique**
6) **Hybrid Technique**

2.9.1 Text – Based Technique

Text based technique is the oldest and simplest way to detect clone, which takes each line of source code as code representation. In this approach, the target source program is assumed to be sequence of liner or strings. And then, two code fragments are compared with each other to find the matched sequences of text or strings. When a match is found i.e. two or more code fragments are fount to be similar, then they are returned as clone pair or clone class by the detection technique. As the name says, it is purely text based so the detected clones do not correspond to structural elements of the language. Mostly, the raw source code is directly used in clone detection process, without any transformation.

However, few of the latest text based clone detection technique use some to the transformation such as comments removal, whitespace removal. Because text based technique does not perform any syntactical or semantically analysis on source code, it is

one of the fastest clone detection approach. It can easily deal with type 1 clone, and with additional data transformation, the type 2 can also be taken care.

Ducasse et al. [33] proposed one of the newer text-based clone detection techniques that are based on dot plots. A dot plot is a two-dimensional chart where both axes list source entities. In this approach the lines of a program are comparison entities. if x and y are equal there is a dot at coordinate (x, y). Two lines are considered equal if they have the same hash value. Dot plots can be used to visualize clone information; diagonals in dot plots are identified as clone. The detection of clones in dot plots can be automated, and string-based dynamic pattern matching is used to compare whole lines. Diagonals that have gaps indicate type 3 clones.

Johnson et al. [18] proposed text-based clone detection technique. This approach uses "fingerprints" on substrings of the source code. First, code fragments of a fixed number of lines are hashed. A sliding window technique in combination with an incremental hash function is used to identify sequences of lines having the same hash value as clones.

2.9.2 Token – Based Technique

Token based technique is similar to text based technique; however, instead of taking a line of code as representation directly, a lexical analyzer converts each line of code into a sequence of token [34]. After data values and identifier are substituted by some special tokens, the token sequences of lines are compared efficiently through a suffix tree algorithm. The result is also presented in dot plot graph. This technique is slightly slower than text based method, because of the tokenization step. However, applying suffix tree matching algorithm, the time complexity is similar as text based technique. By breaking line into tokens, it can easily detect both type 1 and type 2 clone, with token filter applied, the result of clone can be controlled very precisely, for example, skip any uninterested information.

Baker et al [4] proposed Parameterized matching using suffix trees with Dup as implementation example. Parameterized Matching with Suffix Trees consists of three consecutive steps manipulating a suffix tree as internal representation. In the first step, a lexical analyzer passes over the source text transforming identifiers and literals in parameter symbols, while the typographical structure of each line is encoded in a non-parameter symbol. One symbol always refers to the same identifier, literal or structure. The result of this first step is a parameterized string or p-string. Once the p-string is constructed, a criterion to decide whether two sequences in this p-string are a parameterized match or not is necessary. Two strings are a parameterized match if one can be transformed into the other by applying a one-to-one mapping renaming the parameter symbols. After the lexical analysis, a data structure called a parameterized suffix tree (p-suffix tree) is built for the p-string. The use of a suffix tree allows a more efficient detection of maximal, parameterized matches. All that is left for the last step, is to find maximal paths in the p-suffix tree that are longer than a predefined character length.

Kamiya et al. [34, 35] has proposed a clone detection algorithm and implemented a tool named CCFinder (Code clone finder). The tool makes a token sequence form the input code through a lexical analyzer and applies the rule-based transformation to the sequence and used a suffix-tree matching algorithm to compute matching, in which the clone location information is represented as a tree with sharing nodes for leading identical subsequences and the clone detection is performed by searching the leading nodes on the tree.

2.9.3 Abstract Syntax Tree (AST) – Based Technique

In AST based technique the program is parsed into a parser tree or an abstract syntax tree (AST) with a parser of language of interest. Then, using a tree matching technique, similar sub trees are searched in the tree. When a match is found corresponding source code of the similar sub trees are returned as clone pairs or clone classes. The complete information is available in the parse tree of AST. The variable names and literal values of

the source code are discarded during the tree representation; still it is possible to employ more sophisticated clone detection tools. By using AST as code representation gives this technique a better understanding of the system structure. However parsing source file is still a very expensive process on both time and memory.

Baxter et al. [16] pioneers AST-based clone techniques and implemented as a tool CloneDR. A compiler generator is used to generate an annotated parse tree (AST) and compares its subtrees by characterization metrics based on a hash function through tree matching [4]. Source code of similar subtrees is then returned as clone. The hash function enables one to do parameterized matching, to detect gapped clones and to identify clones of code portions in which some statements are reordered

Yang [39] has proposed approach for finding the syntactic differences between two versions of the same programs by generating a variant of parse tree for both the versions and then applying dynamic programming approach in searching similar subtrees.

Wahler et al. [36] find exact and parameterized clones in a more abstract level than AST where the AST of a program is converted to an XML represented and then a data mining frequent item set technique is applied in the XML representation of the AST for finding clones.

2.9.4 Program Dependence Graph (PDG) – Based Technique

Control and data dependencies of a program can be represented as a program dependency graph. As it records the relationship between the data and structure, it can be used to trace the modification after programmer's copy and paste activities [30]. The PDG based technique takes one step further than AST based method, that to obtain the PDG of the system. Semantic information is carried in PDG, because it contains both control flow and data flow information of a program. When a set of PDGs are obtained from a program, isomorphic sub graphs matching algorithm is employed, in order to find similar sub graphs, which are returned as clones. The clone pair can be extracted from the isomorphic sub graph. With control and data dependency information, PDG based

method is the only one that can detect type 3 clones precisely. However, the process is very inefficient, generating PDG, as same as AST parsing process, is a very expensive process on both time and memory.

Komondoor et al [30], proposed an approach that use program dependence graphs (PDGs) and program slicing to find non-contiguous clones, intertwined clones, and clones that involve variable renaming and statement reordering. The tool finds duplicated code fragments in C programs and displays them to the programmer.

Krinke et al. [23] present an approach to identify similar code in programs based on finding similar subgraphs in attributed directed graphs. This approach is used on program dependence graphs and therefore considers not only the syntactic structure of programs but also the data flow within. This approach uses an iterative approach (k-length patch matching) for detecting maximally similar subgraphs in the PDG.

Liu et al. [9] have proposed a plagiarism detection algorithm and implement a PDG-based plagiarism detection tool, called GPlag which detects program plagiarism based on the analysis of program dependence.

2.9.5 Metric – Based Technique

In Metric based technique, instead of comparing the code directly, different metric of code are gathered and these metrics were compared to detect clones. Many clone detection techniques today use metrics for detecting similar codes. Initially, fingerprinting functions which are nothing but a set of software metrics are calculated for one or more syntactic units such as a function or a class, a method or even a statement and then these metrics values are compared to find clones over these syntactic units, generally such metrics are calculated by parsing the source code into AST/PDG representation. Then the metric were calculated form names, layout, expression and simple control flow of function. A clone is detected only when pair of whole function bodies that have similar metrics values are identified.

Mayrand et al. [20] use various metrics to detect clone. Functions with similar metrics values are identify as code clones. Metrics are calculated from names, layout, expressions, and control flow of functions. A function clone is identified as a pair of whole function bodies with similar metrics values.

Patenaude et al. [21] use very similar method-level metrics to extend the Bell Canada Datrix tool to find Java clones. Kontogiannis et al. [24] have proposed two different ways of detecting clones. One approach uses direct comparison of metrics values as a surrogate for similarity at the granularity of begin − end blocks. Five well known metrics that capture data and control flow properties are used. The second approach uses dynamic programming (DP) techniques to compare begin − end blocks on a statement-by-statement basis using minimum edit distance. The hypothesis is that pairs with a small edit distance are likely to be clones caused by cut-and-paste activities. Metric- based approaches have also been applied to find clone in web documents [13].

2.9.6 Hybrid Technique

There are several other detection approaches that use a hybrid technique in detecting clones. Hybrid technique is a combination of the other detection techniques.

Leitao [1] provides a hybrid approach that combines syntactic techniques based on AST metrics and semantic techniques (using call graphs) in combination with specialized comparison functions. Greenan et al [25] proposed an approach for finding method level clones on transformed AST using sequence matching algorithm.

Jiang et al [26] proposed an approach for detecting similar trees and has been implemented as a tool Deckard. In this approach, certain characteristic vectors are computed to approximate the structure of ASTs in a Euclidean space. Locality sensitive hashing (LSH) is then used to cluster similar vectors using the Euclidean distance metric and thus finds corresponding clones.

Koschke et al. [31], proposed an approach in which, in stead of comparing the AST nodes, the approach compares the tokens of the AST-nodes using a suffix tree-based algorithm and therefore, this approach can find clones in linear time and space.Balazinska et al. [27] use a hybrid approach of metric-based characterization and dynamic pattern matching and implemented as a tool SMC (similar methods classifier),measurement is limited to a pre-processing stage for reducing the search space, and then a pattern matching algorithm is applied to compare code fragments for object-oriented programs written in Java.

Cordy et al. [22] propose a hybrid method that combines language-sensitive parsing with language-independent similarity analysis to yield structurally meaningful near-miss clones and implemented as Nicad. It is a new clone detection method that has been shown to yield both high precision and high recall in detecting near-miss intentional clones. The NiCad method involves three main stages, parsing, normalization, and comparison. Keivanloo et al. [15] propose a hybrid approach and implemented as a tool SeClone and involve four stages (1) preprocessing, (2) indexing, (3) searching and (4) postprocessing.
.

Tairas et al. [32] proposed a method to finds function-level clones in a program using abstract syntax trees (ASTs) and suffix trees. An AST provides the structural representation of the code after the lexical analysis process. The AST nodes are used to generate a suffix tree, which allows analysis on the nodes to be performed rapidly. This approach the same methods that have been successfully applied to find duplicate sections in biological sequences to search for matches on the suffix tree that is generated, which in turn reveal matches in the code. Funaro et al. [28] proposed a novel hybrid (syntactic, textual) approach using the abstract syntax tree to identify clone candidates and textual methods to discard false positives and implemented in a prototype called Syn Tex. This hybrid approach, based on two code representations, the abstract syntax tree (AST), to retrieve structural similarities, and, as a string, to refine the results through direct comparison.

Chapter 3

Problem Statement

There are various code clone detection techniques each have its own advantages and disadvantages.

Some techniques find clones by comparing program text [18, 19] with little or no code normalization and other techniques, use a laxer to make a token sequence [34] for the whole program and find clones by finding common subsequences on the token sequence(s) and Some make use of parsers to build a parse tree or an Abstract Syntax Tree (AST) and then find clones by comparing trees/sub-trees [16].

Some of the other techniques [20] calculate some metrics and find clones by comparing the metric and others first build Program Dependency Graphs (PDGs) and find clones by comparing the PDGs [23].

Discussions in the literature review indicate that most of the existing hybrid techniques used tree- based detection. Abstract Syntax tree (AST) based techniques, is a heavyweight and require subtree comparison and full parser. Token-based technique are very fast w.r.t. the number of tokens, but also give many false positives.

The metric based detection technique is more scalable and accurate for large software system. Metric comparison is straightforward as compare with AST/PDG based clone detection technique and Text-based techniques can find clones with high accuracy and confidence.

This thesis, presents a hybrid approach, for detecting clones for object oriented program. This approach use metric based and text based technique to detect clones and divided into two stages. In the first stage metric based technique is used for the selection on potential clone. Potential clones are selected on the basis of metric match and after this potential clones are further processed with text based technique. The potential clones are compared line by line to determine whether two potential clones really are clones of each other.

Chapter 4

Proposed Model

Cloned software is a major problem in large software systems that have evolved over a long period of time. The problem is not amenable to a simple solution – to completely reengineer the system is a possible alternative, but is very cumbersome and involves much cost. A feasible alternative can be to track down and log potential clones. Manually tracing code clones is very difficult. In the proposed model a automated tool for the detecting clone is developed.

Current work presents a practical method, for detecting exact clones for object oriented program source code. In this model a hybrid approach using combination of metrics based and text based technique is used for detection of clone. This code clone detector detects function and class as clone on the basis of the metric match and textual match. The tool will find out which of the functions and class in the given source code files can be possible clones of each other.

4.1 Proposed Model

In proposed work an automated tool for detecting exact clone for object oriented program is developed. The Proposed Model detect clone based on two stages.

- Selection of Potential clone
- Comparison of potential clone

4.1.1 Selection of Potential Clone

The selection of potential clones is automatically performed by a tool. In the first stage

metrics are calculated for two object oriented programs by using Columbus tool [39]. This tool analyzes object-oriented programs and calculates various class level and function level metrics [17].

This tool result in comma separated value (csv) files. After this two csv files are compared with each other and class and function that contain same metric value is selected as potential clone.

4.1.2 Comparing of Potential Clone

The purpose of this stage is to determine whether potential clones are actually clones to each other for this purpose text based technique is used. In this stage two programs are compared line by line.

4.2 Apply Metrics

The metrics computed by the tool include some of the well-known traditional and Object Oriented metrics at class level and function level are discussed in table 4.1 and table 4.2.

4.2.1 Class Level Metrics

Abbr	Description
LOC	Line of Code
WMC	Weighted methods for class. The weight is the McCabe cyclomatic complexity.
SIZE2	Number of attributes plus number of local methods
NUMPAR	The sum of the number of parameters of the methods implemented in a class.
DIT	The DIT of a class is the length of the longest path from the class to the root in the inheritance hierarchy.
NOC	The number of classes that directly inherit from a given class

NOP	The number of classes that a given class directly inherits from
NOD	The number of classes that directly or indirectly inherit from a class
NOA	The number of classes that a given class directly or indirectly inherits from.
NMO	The number of methods in a class that override a method inherited from an ancestor class.
NMA	The number of new methods in a class, not inherited, not overriding.
RFC	The response set of a class consists of the set *M* of methods of the class, and the set of methods directly or indirectly invoked by methods in *M*.
PriA	Number of private attributes
PriM	Number of private methods
ProA.	Number of protected attributes
ProM	Number of protected methods
PubA	Number of public attributes
PubM.	Number of public methods.
NA	Number of attributes of a class (local and inherited)
NAI.	Number of attributes inherited of a class.
NAL	Number of attributes locally defined of a class
NAM	Number of attributes and methods of a class
NAML	Number of attributes and methods locally defined of a class
NM	Number of methods of a class (local and inherited)
NMI	Number of methods inherited of a class
NML.	Number of methods local of a class

Table 4.1 Class Level Metrics

4.2.2 Function Level Metrics

Abbr	Description
LOC	Line of code
NUMPAR	Number of parameters
NI	Number of invocations of other methods in method body
NMAA	Number of accesses on attributes in method body
NOS	Number of statements in method body.
NTIG	Number of times invoked by methods non-local to its class, i.e. from methods implemented in other classes
NTIL	Number of times invoked by methods local to its class, i.e. from methods implemented in the same class
McCC	McCabe's measure is defined as the number of decisions within the specified function plus 1, where each *if*, *for*, *while*, *do...while* and *?:* (conditional operator) counts once and each *N*-way *switch* counts *N*+1 (*else* does not increment the number of decisions).

Table 4.2 Function Level Metrics

4.3 Design

This Code Clone detection tool takes two source code files as an input and gives the output as detecting the function and class, which appear to be clones of each other based on the metric and textual match.

4.3.1 Flow Chart of the Proposed Model

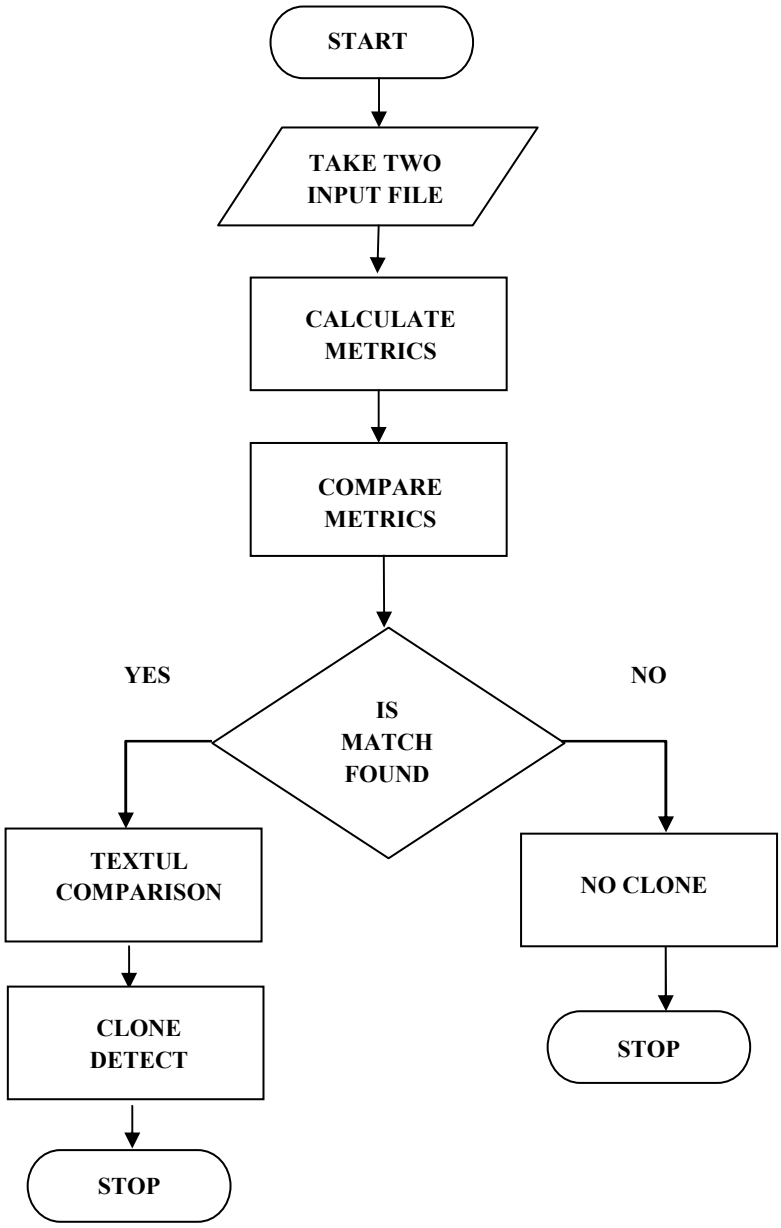


Figure 4.1 Flow Chart of the Proposed Model

4.3.2 Architecture of the Proposed Model

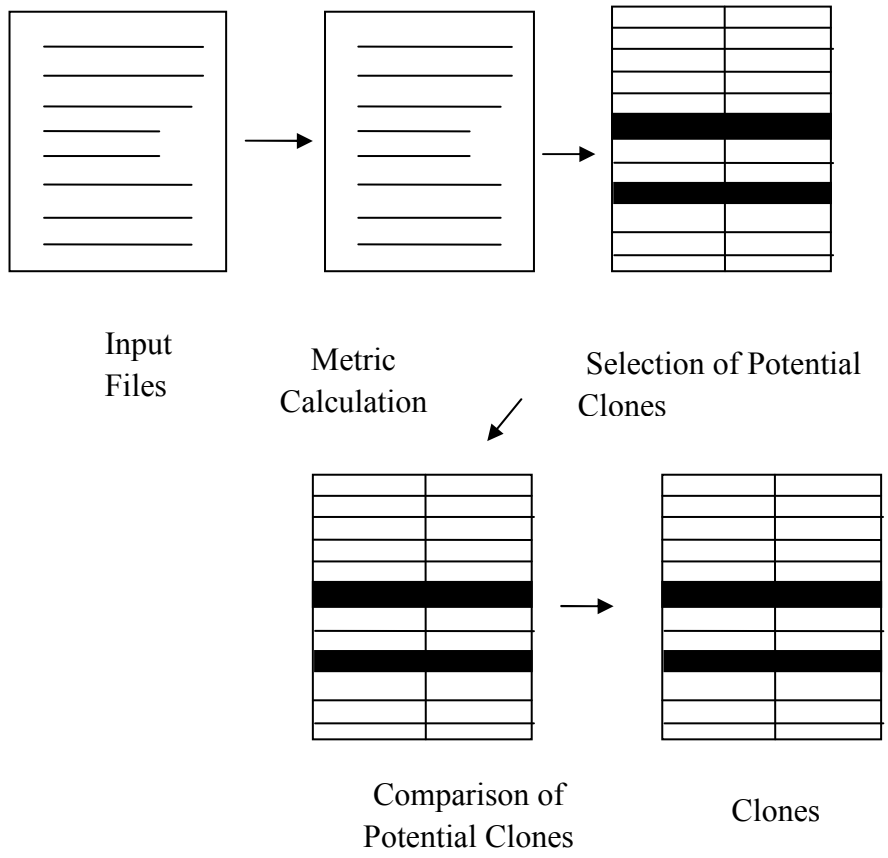


Figure 4.2 Architecture of Proposed Model

4.4 Implementation

Proposed clone detection method has been implemented in vb.net. It extracts clones from C++ source files. The tool receives the paths of source files as input, and display the result by highlighting the lines that are same.

4.5 Steps of Proposed Model

Step I:

In the first Step two C++ source files are added as shown in figure 4.3.

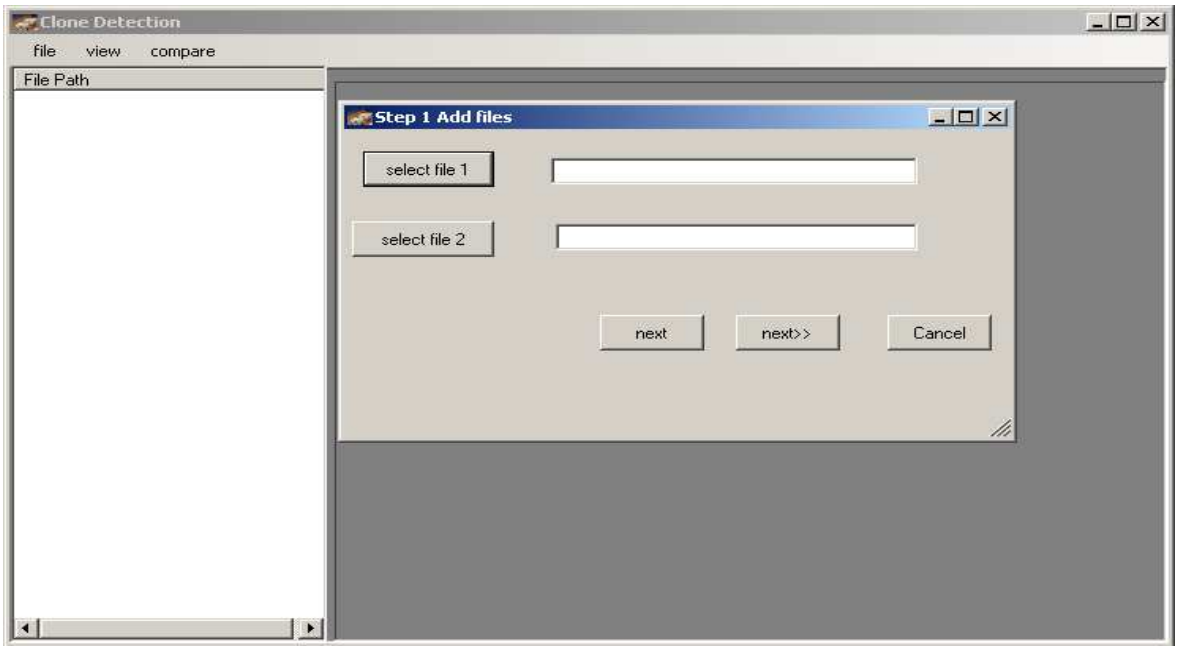


Figure 4.3 Add two Source Files

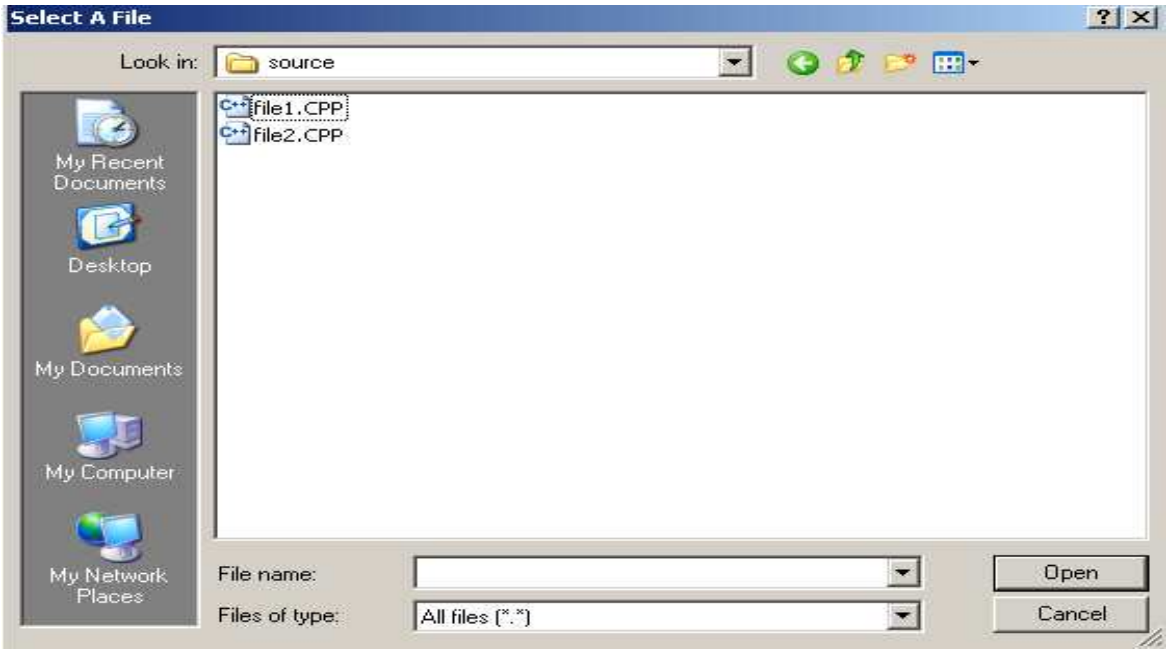


Figure 4.4 Open File Dialog

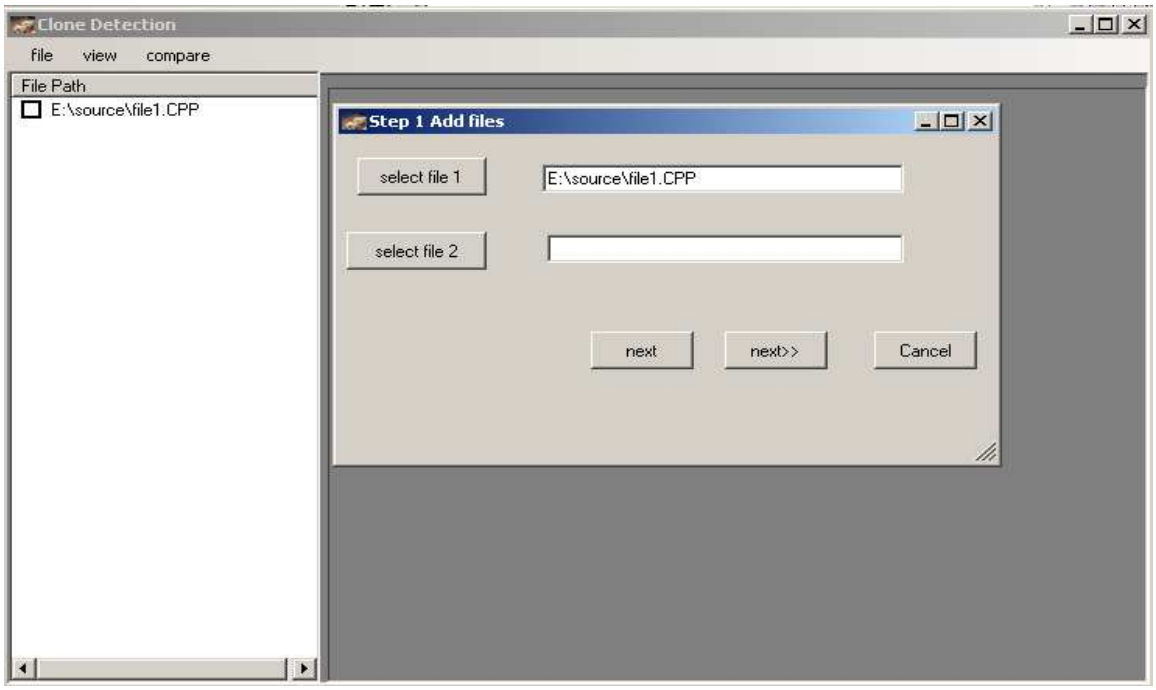


Figure 4.5 Select Source Files

Step II: Metric Calculation

In the Second Step metrics are calculated for two source files as shown in figure 4.6 and result is displayed in to comma separated values (csv) file as shown in figure 4.7 it can be read by Microsoft excel. The metrics are calculated by using Columbus tool [40].

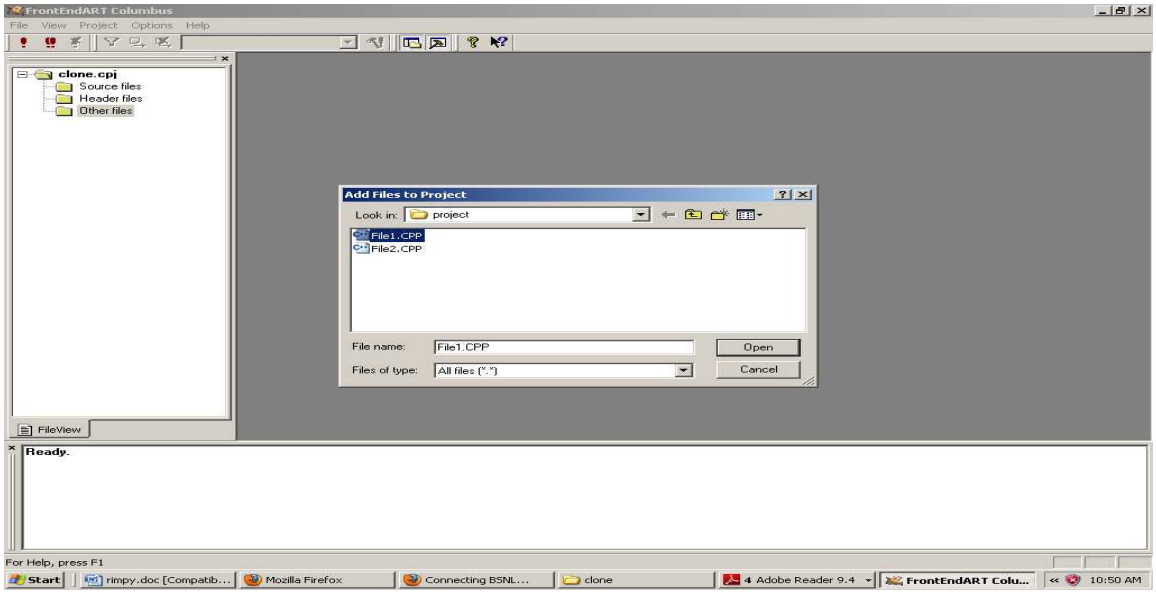


Figure 4.6 Add files for Metric Calculation

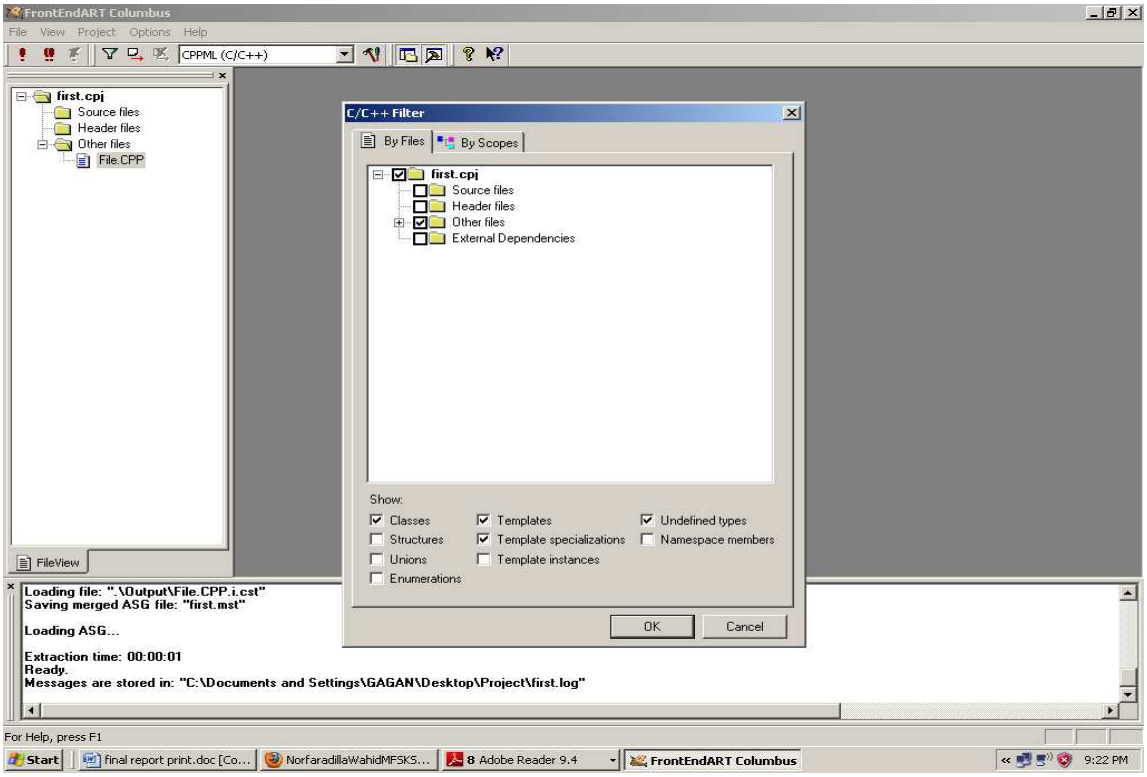
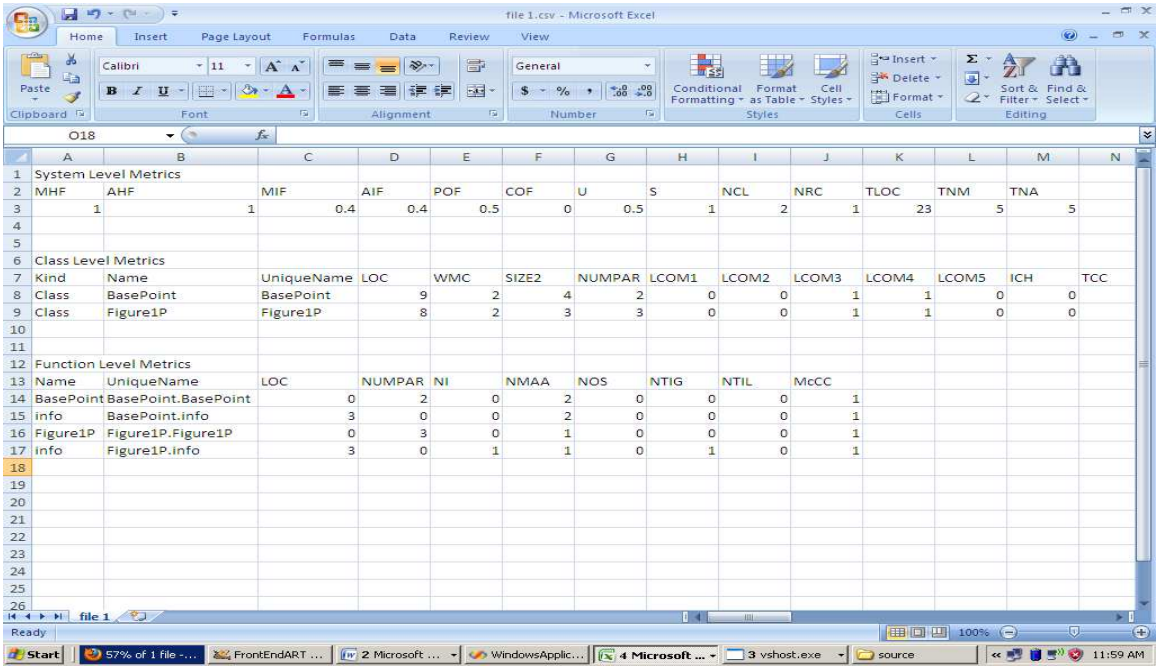


Figure 4.7 Metric Extraction by Tool



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	System Level Metrics													
2	MHF	AHF	MIF	AIF	POF	COF	U	S	NCL	NRC	TLOC	TNM	TNA	
3	1	1	0.4	0.4	0.5	0	0.5	1	2	1	23	5	5	
4														
5														
6	Class Level Metrics													
7	Kind	Name	UniqueName	LOC	WMC	SIZE2	NUMPAR	LCOM1	LCOM2	LCOM3	LCOM4	LCOM5	ICH	TCC
8	Class	BasePoint	BasePoint	9	2	4	2	0	0	1	1	0	0	
9	Class	Figure1P	Figure1P	8	2	3	3	0	0	1	1	0	0	
10														
11														
12	Function Level Metrics													
13	Name	UniqueName	LOC	NUMPAR	NI	NMAA	NOS	NTIG	NTIL	McCC				
14	BasePoint	BasePoint.BasePoint	0	2	0	2	0	0	0	1				
15	info	BasePoint.info	3	0	0	2	0	0	0	1				
16	Figure1P	Figure1P.Figure1P	0	3	0	1	0	0	0	1				
17	info	Figure1P.info	3	0	1	1	0	1	0	1				
18														
19														
20														
21														
22														
23														
24														
25														
26														

Figure 4.8 csv File of Metric Values of a Program

Step III: Compare Excel Files

In this step two comma separated value (csv) files that contain the metric value of C++ program are compared with each other and if the metric value matches in both file is highlighted in csv files as shown in figure 4.10.

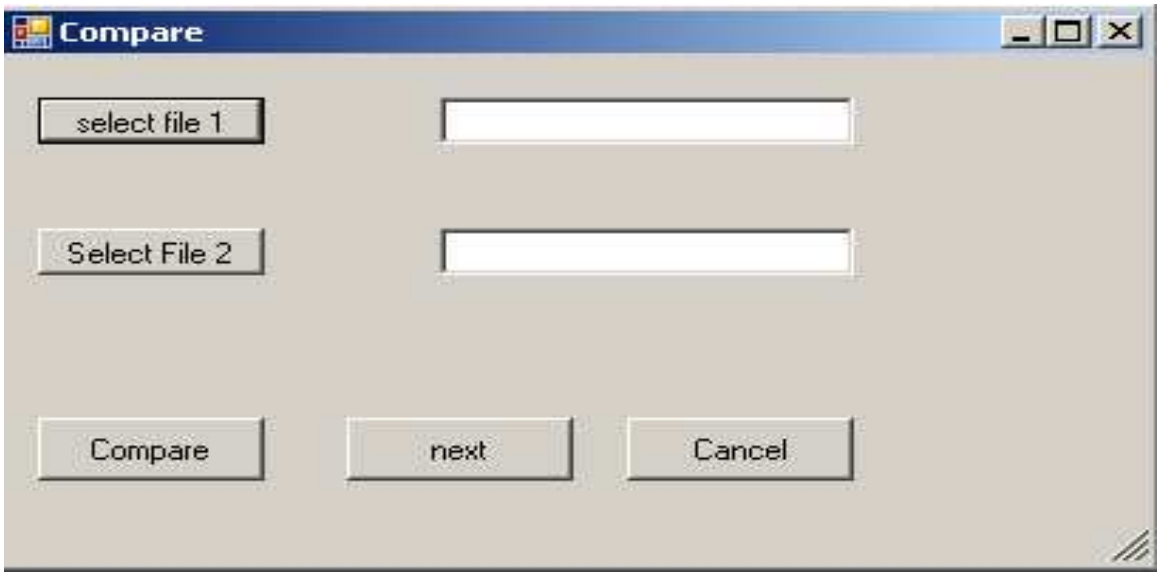


Figure 4.9 Add Two csv File

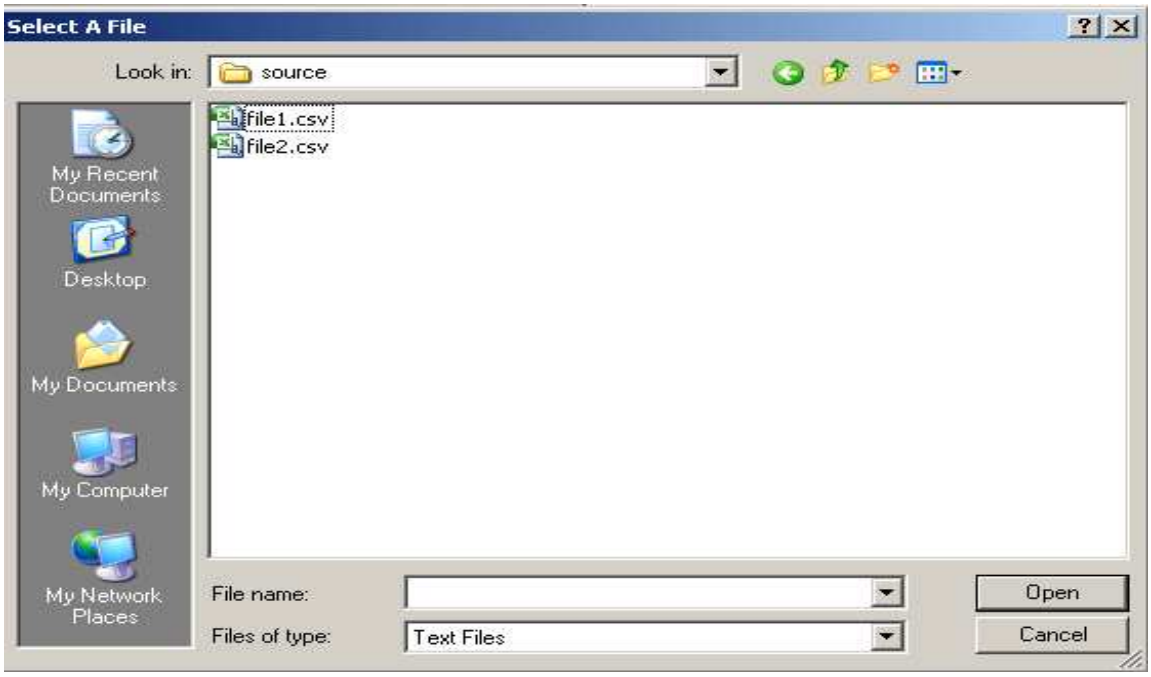


Figure 4.10 Select csv File

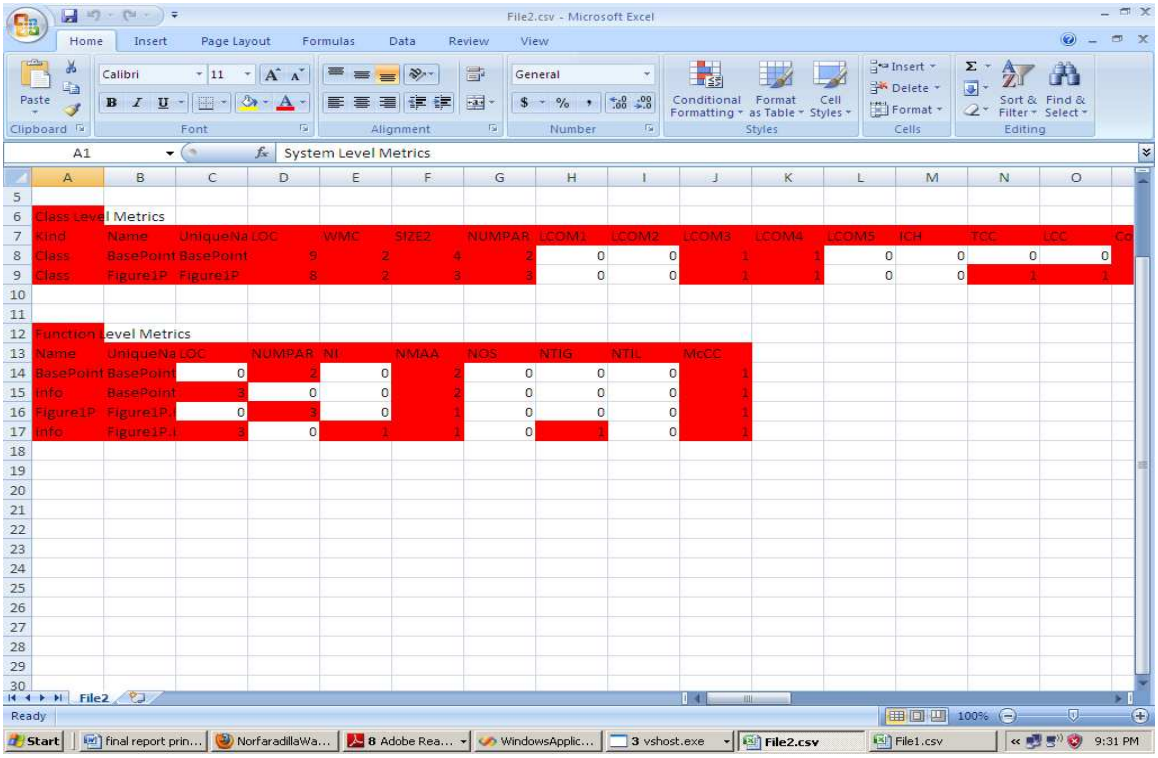


Figure 4.11 Result of the Metric Comparison

Step IV: Textual Comparison

In the last step if the metric value matches text based technique is applied to the C++ source file. In this step two source files are compared with each other and the lines that are same in both files are highlighted in both files as shown in figure 4.11.

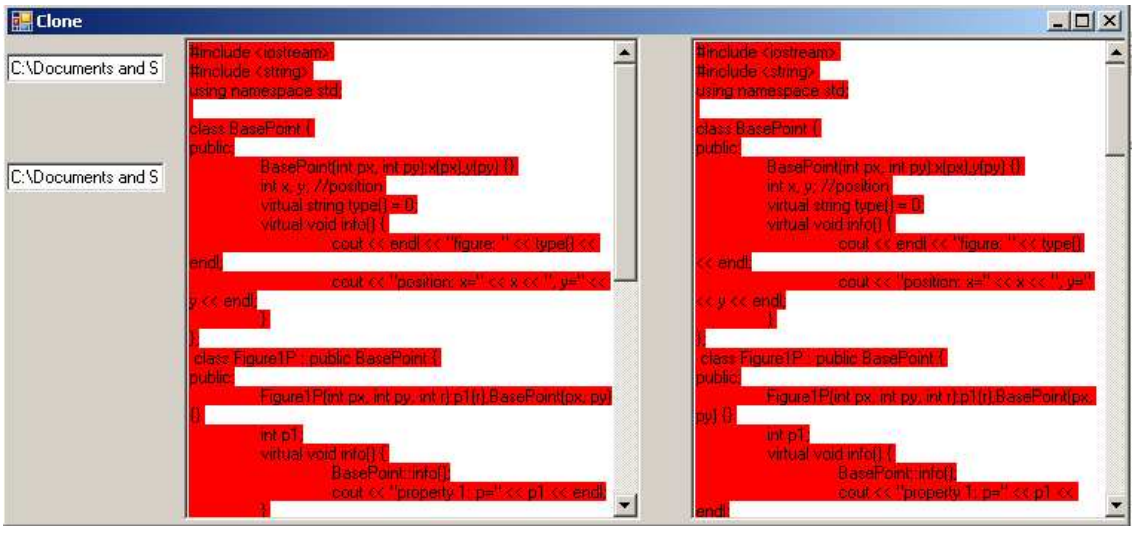


Figure 4.12 Result of the Clone Detection Process

Chapter 5

Experimental Result

5.1 Case Study

Two files written in C++ programming language are considered here for testing of proposed method. Having implemented the above process a popular medium sized open source C++ project Mozilla [40] file is considered for experimental result.

5.1.1 Input Files

The files of code are as follows

File 1:

```
#include "nsIAbBooleanExpression.h"
#include "nsCOMPtr.h"
#include "nsString.h"

class nsAbBooleanConditionString : public nsIAbBooleanConditionString
{
public:
   NS_DECL_ISUPPORTS
   NS_DECL_NSIABBOOLEANCONDITIONSTRING

   nsAbBooleanConditionString();
   virtual ~nsAbBooleanConditionString();
protected:
   nsAbBooleanConditionType mCondition;
   nsCString mName;
   nsString mValue;
};

class nsAbBooleanExpression: public nsIAbBooleanExpression
{
public:
   NS_DECL_ISUPPORTS
```

```
  NS_DECL_NSIABBOOLEANEXPRESSION

  nsAbBooleanExpression();
  virtual ~nsAbBooleanExpression();

protected:
  nsAbBooleanOperationType mOperation;
  nsCOMPtr<nsISupportsArray> mExpressions;
};

#endif
```

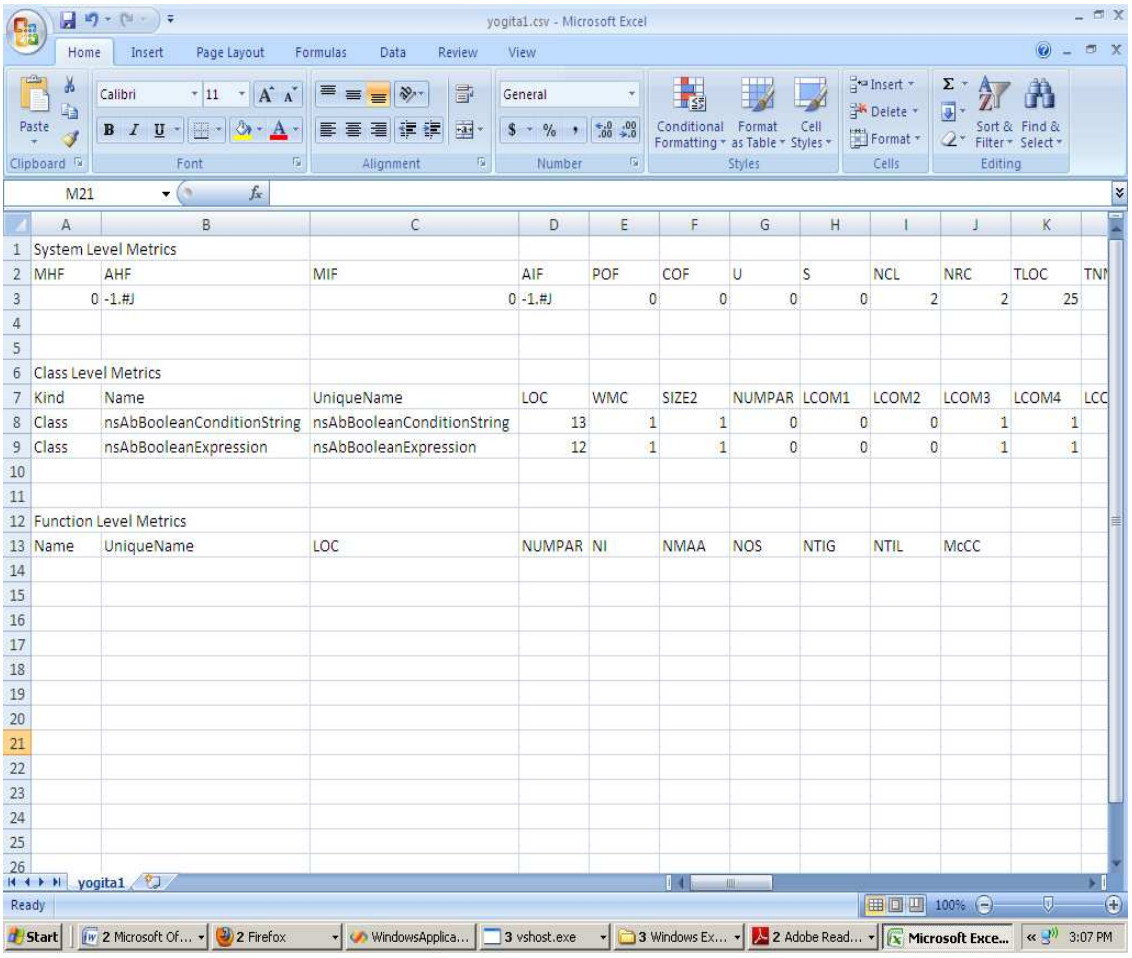
File 2:

```
#include "nsIAbBooleanExpression.h"
#include "nsCOMPtr.h"
#include "nsString.h"

class nsAbBooleanConditionString : public nsIAbBooleanConditionString
{
public:
  NS_DECL_ISUPPORTS
  NS_DECL_NSIABBOOLEANCONDITIONSTRING

  nsAbBooleanConditionString();
  virtual ~nsAbBooleanConditionString();
protected:
  nsAbBooleanConditionType mCondition;
  nsCString mName;
  nsString mValue;
};
class nsAbBooleanExpression: public nsIAbBooleanExpression
{
public:
  NS_DECL_ISUPPORTS
  NS_DECL_NSIABBOOLEANEXPRESSION
  nsAbBooleanExpression();
  virtual ~nsAbBooleanExpression();

protected:
  nsAbBooleanOperationType mOperation;
  nsCOMPtr<nsISupportsArray> mExpressions;
};
#endif
```

5.1.2 Output Files

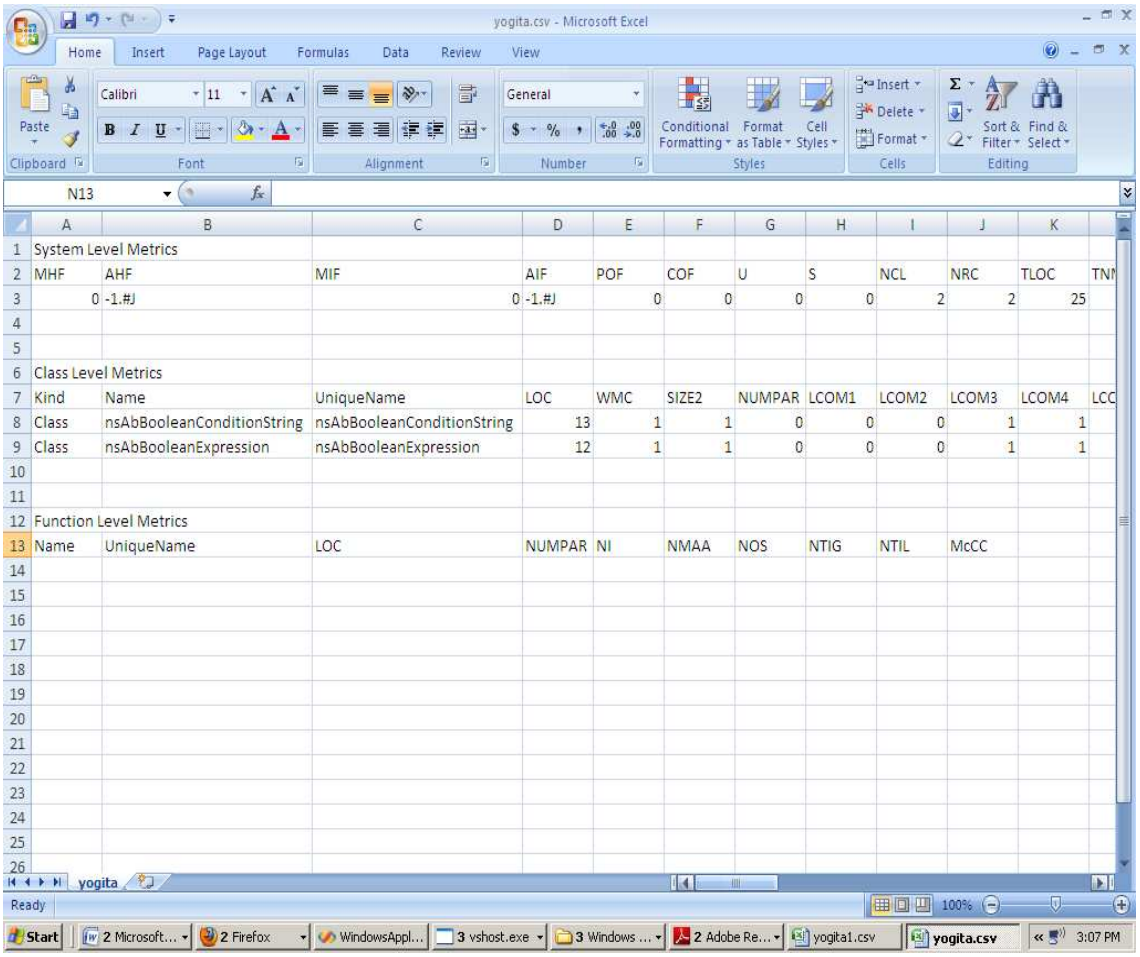
The output of the tool is organized in the following format:

1 The two coma separated valued (csv) files that contain metric values of two input Program files are shown in Figure 5.1 and Figure 5.2.



	A	B	C	D	E	F	G	H	I	J	K
1	System Level Metrics										
2	MHF	AHF	MIF	AIF	POF	COF	U	S	NCL	NRC	TLOC
3	0	-1.#J	0	-1.#J	0	0	0	0	2	2	25
4											
5											
6	Class Level Metrics										
7	Kind	Name	UniqueName	LOC	WMC	SIZE2	NUMPAR	LCOM1	LCOM2	LCOM3	LCOM4
8	Class	nsAbBooleanConditionString	nsAbBooleanConditionString	13	1	1	0	0	0	1	1
9	Class	nsAbBooleanExpression	nsAbBooleanExpression	12	1	1	0	0	0	1	1
10											
11											
12	Function Level Metrics										
13	Name	UniqueName	LOC	NUMPAR	NI	NMAA	NOS	NTIG	NTIL	McCC	

Figure 5.1 csv File of Metric Value for First Program



	A	B	C	D	E	F	G	H	I	J	K
1	System Level Metrics										
2	MHF	AHF	MIF	AIF	POF	COF	U	S	NCL	NRC	TLOC
3	0	-1.#J	0	-1.#J	0	0	0	0	2	2	25
4											
5											
6	Class Level Metrics										
7	Kind	Name	UniqueName	LOC	WMC	SIZE2	NUMPAR	LCOM1	LCOM2	LCOM3	LCOM4
8	Class	nsAbBooleanConditionString	nsAbBooleanConditionString	13	1	1	0	0	0	1	1
9	Class	nsAbBooleanExpression	nsAbBooleanExpression	12	1	1	0	0	0	1	1
10											
11											
12	Function Level Metrics										
13	Name	UniqueName	LOC	NUMPAR	NI	NMAA	NOS	NTIG	NTIL	McCC	

Figure 5.2 .csv File of Metric Value for Second Program

2 After the extraction of metric values of two source files, two csv files are compared with each other. The metric value that macth in both files are shown in figure 5.3.

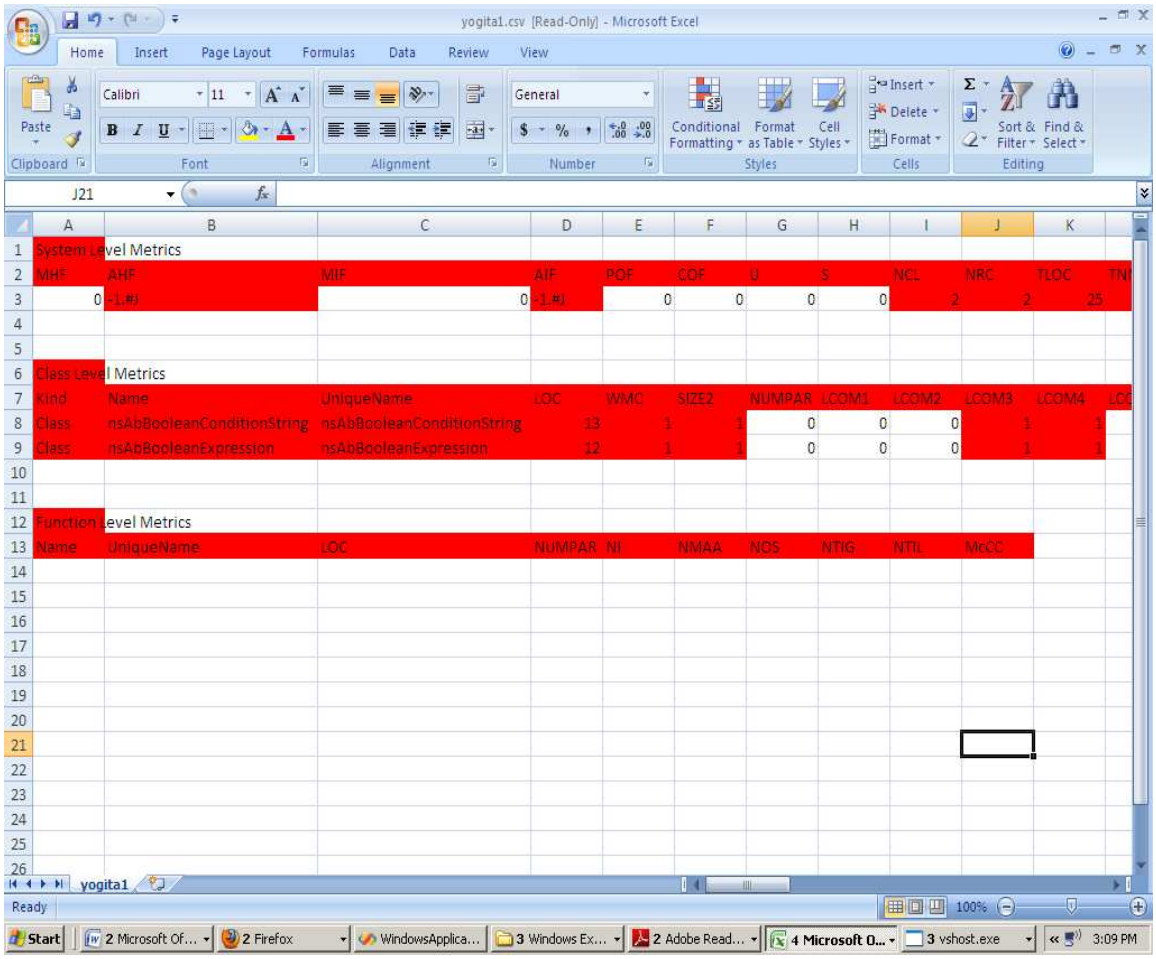


Figure 5.3 Metric Values Comparison of Both File

3 After the comparison of .csv files if metric value match textual caparisson is applied and lines that are similar in both files are highlighted in both files.

Chapter 6

Conclusion and Future Work

Software clone is a phenomenon in large software system. It is usually caused by programmers' copy and paste activities. The reason for the existence of clones in the source code is that making a copy of a code fragment is simpler and faster than writing it from scratch. Sometimes maintenance programmers do not fully understand the program and therefore they re-implement some already existing functionality. Another reason is time limit that is assigned to the developer to finish the project in that case, programmers frequently copy-and-paste the code and update it according to new product's requirements.

Although it seems to be a simple and effective method, these duplication activities are usually not documented, that cause a number of negative effects on the quality of the software, increasing the amount of the code which needs to be maintained, and duplication also increases the defect probability and resource requirements.

Such duplications increase code size and also maintenance and comprehension becomes more difficult. Because as much as 80% of the total life cycle cost is spent on maintenance, clone elimination could translate into substantial savings.

In this thesis a hybrid approach using metric based technique with the combination of text based technique for detection and reporting of clones is proposed. The Proposed work is divided into two stages selection of potential clones and comparing of potential clones. The proposed technique detects exact clones on the basis of metric match and then by text match.

6.1 Conclusion

- The Proposed technique detects clones on the basis of metric match and textual match.
- The Proposed technique looks for clones at the class level and function level.
- Proposed technique detects potential clones on the basis of metric match. Potential clones are compared line-by-line to determine whether two potential clones really are clones of each other.

6.2 Future Work

- This technique further can be extended to detect type II and Type III clone.
- The Current tool works only for the C++ source code files. This work can be extended to any other languages.
- The proposed technique can identify clones in the given two files. It does not support any feature to remove these clones.

References

1. A.M. Leitao, "Detection of redundant code using *R2D2*," *in Software Quality Journal*, vol. 12, no. 4, pp. 361-382, 2004.
2. Arun Lakhotia, Junwei Li, Andrew Walenstein, and Yun Yang," Towards a clone detection benchmark suite and results archive," in *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, pp. 285, Washington, DC, USA, 2003.
3. B. Baker, "A Program for Identifying Duplicated Code," in *Proceedings of Computing Science and Statistics: 24th Symposium on the Interface*, Vol. 24, pp. 49–57, March 1992.
4. B. Baker, "Finding Clones with Dup: Analysis of an Experiment," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 608–621, 2007.
5. B. Baker, "On Finding Duplication and Near-Duplication in Large Software Systems," in *Proceedings of the Second Working Conference on Reverse Engineering (WCRE'95)*, pp. 86–95, Toronto, Ontario, Canada, July 1995.
6. Brooks, F., 1975. *The mythical man-month*: essays on software engineering, Reading Mass: Addison-Wesley Pub. Co.
7. C.K. Roy and J.R. Cordy, "A Survey on Software Clone Detection Research,*"*Queen's School of Computing Technical Report No. 2007-541, vol. 115, September 2007.
8. C.K. Roy, J.R. Cordy and R. Koschke, "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach," *Science of Computer Programming*, vol. 74, no. 7, pp. 470-495, May 2009.
9. Chao Liu, Chen Chen, Jiawei Han and Philip S. Yu, "GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis", in *the Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* , pp. 872-881, Philadelphia, USA, August 2006.

10. Cory Kapser, Michael W. Godfrey , "Cloning Considered Harmful" Considered Harmful," in *proceedings of the 13th Working Conference on Reverse Engineering,* pp. 19-28, Washington, DC, USA, 2006.
11. Ettore Merlo1, "Detection of Plagiarism in University Projects Using Metrics-based Spectral Similarity," in *the Dagstuhl Seminar: Duplication, Redundancy, and Similarity in Software*, 2007.
13. F. Calefato, F. Lanubile and T. Mallardo, "Function Clone Detection in Web Applications: A Semiautomated Approach," in *Journal of Web Engineering*, vol. 3, no. 1, pp 3–21, 2004.
14. H. Liu, Z. Ma, L. Zhang, and W. Shao, "Detecting duplications in sequence diagrams based on suffix trees," in *Proc. of APSEC* , pp. 269-276, 2006.
15. Iman Keivanloo, , Juergen Rilling, Philippe Charland, "SeClone - A Hybrid Approach to Internet-scale Real-time Code Clone Search," in *19th IEEE International Conference on Program Comprehension (ICPC)*, Kingston, Ontario, Canada, 2011.
16. Ira Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna, "Clone Detection Using Abstract Syntax Trees," in *Proceedings of the 14th International Conference on Software Maintenance (ICSM'98)*, pp. 368-377, Bethesda, Maryland, November 1998.
17. Istvan Siket, Rudolf Ferenc, "Calculating Metrics from Large C++ Programs" in *6th International Conference on Applied Informatics Eger, Hungary* ,pp. 27–31, January 2004.
18. J. H. Johnson, "Identifying Redundancy in Source Code Using Fingerprints," in *Proceeding of the 1993 Conference of the Centre for Advanced Studies Conference (CASCON' 93)*, pp. 171–183, Toronto, Canada, October 1993.
19. J. H. Johnson, "Visualizing Textual Redundancy in Legacy Source," in *Proceedings of the 1994 Conference of the Centre for Advanced Studies on Collaborative research (CASCON'94)*, pp. 171–183, Toronto, Canada, 1994.
20. J. Mayrand, C. Leblanc and E. Merlo, "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics," in *Proceedings of the 12th*

International Conference on Software Maintenance (ICSM'96), pp. 244–253, Monterey, CA, USA, November 1996.

21. J.-F. Patenaude, E. Merlo, M. Dagenais, and B. Lague, "Extending software quality assessment techniques to java systems," in *Proceedings of the 7th International Work- shop on Program Comprehension (IWPC)*, pp. 4956, Pittsburgh, PA, USA, May 1999.
22. J.R. Cordy and C.K. Roy, "The NiCad Clone Detector," in *19th International Conference on Program Comprehension*, Kingston, Canada, June 2011.
23. Jens Krinke, "Identifying Similar Code with Program Dependence Graphs," in *Proceedings of the 8th Working Conference of Reverse Engineering,* pp. 301—309, Stuttgart, Germany, October 2001.
24. K. Kontogiannis, R. DeMori, E. Merlo, M. Galler and M. Bernstein," Pattern Matching for Clone and Concept Detection," *Journal of Automated Software Engineering*, Vol. 3, no. 1-2, pp.77–108, June 1996.
25. Kevin Greenan, "Method-Level Code Clone Detection on Transformed Abstract Syntax Trees using Sequence Matching Algorithms," Student Report, University of California - Santa Cruz, Winter 2005.
26. Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu, "DECKARD: Scalable and Accurate Tree-based Detection of Code Clones," in *Proceedings of the 29th International Conference on Software Engineering (ICSE'07)*, pp. 96-105, Minnesota, USA, May 2007.
27. Magdalena Balazinska, Ettore Merlo, Michel Dagenais, Bruno Lague, Kostas Kon togiannis, "Measuring Clone Based Reengineering Opportunities," in *Proceedings of the 6th International Software Metrics Symposium*, pp. 292-303, Boca Raton, Florida, USA, November 1999.
28. Marco Funaro, Daniele Braga,Alessandro Campi, Ghezzi Carlo, **"**A Hybrid Approach (Syntactic and Textual) to Clone Detection," in *international workshop of software clones,* 2001.
29. N. Davey, P. Barson, S. Field and R. Frank, "The Development of a Software Clone Detector," in *International Journal of Applied Software Technology*, vol.1, no. ¾, pp. 219–236, 1995.

30. Raghavan Komondoor and Susan Horwitz, "Using Slicing to Identify Duplication in Source Code," in *Proceedings of the 8th International Symposium on Static Analysis (SAS'01)*, Vol. LNCS 2126, pp. 40-56, Paris, France, July 2001.
31. Rainer Koschke, Raimar Falke and Pierre Frenzel, "Clone Detection Using Abstract Syntax Suffix Trees," in *Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06)*, pp. 253-262, Benevento, Italy, October 2006.
32. Robert Tairas, Je Gray, "Phoenix-Based Clone Detection Using Suffix Trees," in *Proceedings of the 44th annual Southeast regional conference (ACM-SE)*, pp. 679- 684, Melbourne, Florida, USA, March 2006.
33. S. Ducasse, M. Rieger and S. Demeyer, "A Language Independent Approach for Detecting Duplicated Code," in *Proceedings of the 15th International Conference on Software Maintenance (ICSM'99)*, pp. 109–118, September 1999.
34. T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A MultiLinguistic Token-Based Code Clone Detection System for Large Scale Source Code," IEEE Trans. Software Eng., vol. 28, no. 7, pp. 654-670, July 2002.
35. Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue,"A Token-based Code Clone Detection Tool-ccfinder and its empirical evaluation," Technical report, 2000.
36. V. Wahler, D. Seipel, J. Gudenberg and G. Fischer, "Clone Detection in Source Code by Frequent Itemset Techniques" in *Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation (SCAM)*, pp. 128–135, Chicago, IL, USA, September 2004.
37. W. Yang, "Identifying Syntactic Differences Between Two Programs," in *Software Practice and Experience*, vol. 21, no. 7, pp. 739–755, July 1991.
38. Yue JIA, "Clone Detection Using Dependence Analysis and Lexical Analysis,"PhD Thesis, King's College London, 2007.
39. "The Columbus Homepage," http://www.frontendart.com.
40. "The Mozilla Homepage," http://www.mozilla.org.

Publications

Yogita Sharma, Rajesh Bhatia and Raj Kumar Tekchandani "Hybrid Technique For Object Oriented Software Clone Detection" "15th International Conference on Software Engineering and Applications", Dallas, USA, 14 – 16, Dec (Communicated).