Code Clone Detector: A Hybrid Approach on Java Byte Code

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

Master of Engineering
in
Software Engineering

Submitted By
Kanika
(Roll No. 801131015)

Under the supervision of:
RajKumar Tekchandani
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2013

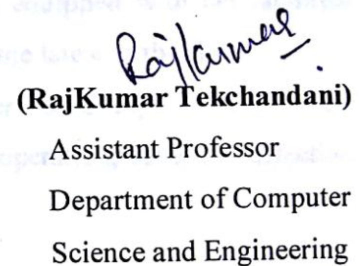
Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Code Clone Detector: A Hybrid Approach on Java Byte Code"**, 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 **RajKumar Tekchandani** and refers other researcher's work which are duly listed in the reference section.

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

(Kanika)

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



(RajKumar Tekchandani)
Assistant Professor
Department of Computer
Science and Engineering

Countersigned by:

(Dr. Maninder Singh)
Head
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 supervisor **RajKumar Tekchandani**.I thank my supervisor 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 and 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. S. K. Mohapatra**, Senior Professor and Dean of Academic Affairs 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 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 copying of source code is a major problem for large, industrial systems. The main effect of cloning is that it risks the maintenance process. Cloning is the basic means of software reuse. Code cloning has been extensively used within the software development design community. Unofficial surveys carried out with large, long term software development projects suggest that 25-30% of the modules in this kind of system may have been cloned.

The objective of this thesis is to understand and analyze the concept of software Cloning and its detection. Software cloning is a perception in which source code is duplicated. Code clones and its detection is one of the emerging and most dominant area of research in the field of software engineering. There exist a number of techniques to detect clone in software. The aim of this study will be given for acquiring and analyzing the concept of hybrid clone detection technique. An algorithm is devised for detecting duplicacy in the software by using hybrid software clone detection technique. This algorithm will first compute the required software metrics that provide sufficient information regarding the software application and then depending on software metrics matches the potential clone will be detected. It uses byte code to calculate the metrics of Java programs, instead of using any transformed representation. The reason of using byte code is that it is platform independent and represents the unified structure of the code. While detecting clones token based approach is applied on potential clones.

Table of Contents

Certificate i

Acknowledgement ii

Abstract iii

Table of Contents iv

Table of Figures vi

List of Tables viii

Abbreviations ix

Chapter1 Introduction 1

1.1 Code Cloning 1

1.2 Reasons of Code Cloning 3

1.3 Problems with clones in code 6

1.4 Consequence of Code Cloning 6

1.5 Motivation and Objective 7

1.6 Outline of thesis 8

Chapter2 Literature Survey 9

2.1 Basic Concepts of Clone Detection 9

2.1.1 Clone Relation Terminology 9

2.1.2 Code Clone types 10

2.2 Advantage of Clone Detection 12

2.3 Clone Detecting Process 13

2.4 Clone Detection Techniques 16

2.4.1 Text Based Technique 16

2.4.2 Token Based Technique 17

2.4.3 Abstract Syntax Tree Based Technique 18

2.4.4 PDG Based Technique 19

2.4.5 Metric Based Technique 20

2.4.6 Hybrid Based Technique 21
2.5 Cloning and Software Maintenance 23
Chapter3 Problem Statement 24
Chapter4 Proposed Work and Implementation 26
4.1 Design of Solution 26
4.1.1 Flow Diagram of Proposed Work 27
4.1.2 Architecture of Proposed Work 28
4.2 MCD Finder 28
4.2.1 Algorithm for Metric Calculation 29
4.2.2 Algorithm for Comparison of Excel Sheets 30
4.3 Implementation of Proposed Tool 31
4.3.1 Adaptation Phase 31
4.3.2 Computation Phase 33
4.3.3 Measurement Phase 37
4.4 Token Based Approach on Potential Clones 39
4.4.1 Clone Detection Process Using Token Based approach 39
4.4.2 Implementation using Token Based approach 40
Chapter5 Experimental Results and Discussion 41
Chapter6 Conclusion and Future Work 48
6.1 Conclusion 49
6.2 Future Work 49
References **50**
Publications **54**
Appendices **55**

List of Figures

Figure 1.1 Code with Clones 2
Figure 1.2 Tree Diagram for Reasons of Cloning 5
Figure 2.1 Clone Pair and Clone Class 10
Figure 2.2 Code Example showing Clone Pair and Clone Class 11
Figure 2.3 Clone Detecting Process 15
Figure 4.1 Flow Diagram of MCD Finder 27
Figure 4.2 Architecture of Proposed Work 28
Figure 4.3 Adaptation Phase: Compilation of Java Program 31
Figure 4.4 Java program to find factorial 32
Figure 4.5 Byte Code Representation of Program 33
Figure 4.6 Start up page 35
Figure 4.7 Choosing .class File 35
Figure 4.8 Metrics at Class level 36
Figure 4.9 Metrics at Function level 36
Figure 4.10 Add two .xls files for Comparison 37
Figure 4.11 Choosing .xls File 37
Figure 4.12 Loading Excel file in database 38
Figure 4.13 Comparison of Class level Metrics 38
Figure 4.14 Clone Detection approach using CCFinder 39
Figure 4.15 Results of Clone Detection using CCFinder 40
Figure 5.1 Class Metric value in database 41
Figure 5.2 Function Metric value in database 42
Figure 5.3 Number of Potential Clones Detected on basis of Metric Comparison 44
Figure 5.4 Class Metric of Adler32 mapped in excel sheets 44
Figure 5.5 Function Metric of Adler32 mapped in excel sheets 45

Figure 5.6 Class Metric of Deflater32 mapped in excel sheets 45

Figure 5.7 Function Metric of Deflater32 mapped in excel sheets 45

Figure 5.8 Comparison of Class level Metrics of Adler32 and Deflater32 46

Figure 5.9 Comparison of Function level Metrics of Adler32 and Deflater32 46

Figure 5.10 Token Based Comparison using CCFinder 47

Figure 5.11 Token Based Comparison in terms of Scatter Plot using CCFinder 47

List of Tables

Table 2.1 Clone Detection Approaches 23

Table 4.1 Class Level Metrics 34

Table 4.2 Function Level Metrics 34

Table 5.1 Class Metric values for Tested Programs 42

Table 5.1 Function Metric values for Tested Programs 43

Abbreviations

MCD	Metrics Based Clone Detection
CP	Clone Pair
CC	Clone Class
LOC	Line of Code
GUI	Graphical User Interface
LSH	Locality Sensitive Hashing
AST	Abstract Syntax Tree
PDG	Program Dependence Graph
DP	Dynamic Programming

Chapter 1

Introduction

In simple terms, when someone copies a part of software, they are creating a clone of the original software. Unfortunately, this definition is a little vague, making a backup copy of the source code using Windows Explorer is not cloning. To be more precise, the idea behind software cloning is to write a new portion of software that duplicates the appearance and functionality of the original software as strongly as possible.

It is important to recognize that cloning does not have to involve any source code from the original software. In fact software cloning normally occurs when there are rigid constraints for programmers. Software cloning does not imply source code copying. The goal of cloning is to create a new software program that mimics everything the original software does and the way in which it does it.

Is software cloning illegal? Well, unfortunately, at this time this is not an easy answer. The two most common ways of addressing cloning is through copyright and trademark laws. Unfortunately, neither one of these is adequate to handle the case of cloning. Copyright laws tend to be more concerned with source code copying and start to break down when source code is not an issue. Trademark laws tend to address product names and specific features rather than an entire product.

1.1 Code Cloning

Code duplication occurs normally during the development and evolution of large software systems. This ad-hoc form of reuse consists of copying and modifying a block of existing code that implement a piece of required functionality. Duplicate blocks are called **clones** and the act of copying including minor modifications is called as ***cloning*.**

Copying code fragments and then reuse by pasting with or without minor modifications or adaptations are common activities in software development. This type of reuse approach of existing code is called code cloning and the pasted code fragment

(with or without modifications) is called a clone of the original.

Definition : A code clone is a code portion of source files that is identical or similar to another [1].

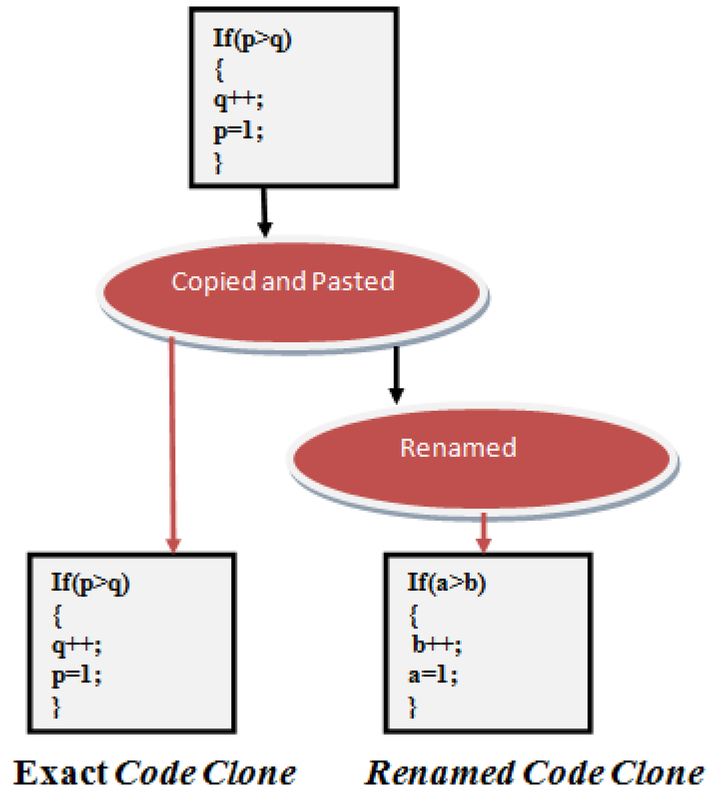


Figure 1.1: Code with Clones.

Figure-1.1 shows the example of code with clones. The results of several studies [1] indicate that a considerable fraction (5-10%) of the source code in large software systems is duplicate code.

The reason behind code cloning can be intentional or unintentional. To exacerbate the situation cloning is performed rapidly and without a care about the context. This means that error free code becomes erroneous after cloning [3]. Furthermore programmer often copies the other's code without fully understanding it.

The software life cycle comprises of three steps: first we have to clearly define the requirements, implement these requirements and then we have to maintain the software and evolve it according to user's requirements. But from the development point of view maintenance is the most crucial activity in terms of cost and effort. Code clones are

considered one of the bad smells of software system [2] and indicators of poor maintainability. Various studies show that the software system with code clones is difficult to maintain as compared to non cloned code software system [1] [33].

1.2 Reason of Code Cloning

There are a number of reasons why developers clone source code. Cloning generally occurs because programmers find that it is easier to use the copy and paste feature than writing the code from scratch. Sometimes programmers intent on implementing the new functionality find some working code that performs a computation nearly identical to the one desired and copy it entirely and then reengineer it [5].

Developers may duplicate code because they are under time constraints; these constraints may be forced by deadlines, or by LOC performance assessment. Another reasonable circumstance where developers duplicate code is they do not completely understand the problem, but they are aware of code that can do some or all of the required functionality.

While this is actually good reuse practice, it complicates the maintenance process. Code cloning is considered a serious problem in industrial software [4]. 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

Figure 1.2 displays various factors for which clones can be introduced in the source code and a short description of some of the factors are discussed below [5]:

- **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 these developers just copy and paste the existing one and adapt to their current need.

- **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.

- **Difficulty in Understanding Large System**

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

- **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.

- **By Accident**

  Code cloning may be performed accidentally. There may be a case that two software developers may come with the 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 accidentally repeat a common solution for similar kind of problems using the common solution pattern of his memory of such similar problems. Therefore, some clones may unknowingly be formed to the software systems.

- **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.

- **Risk in New Code**

  There is a high risk of software errors in new code fragments and because the 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.

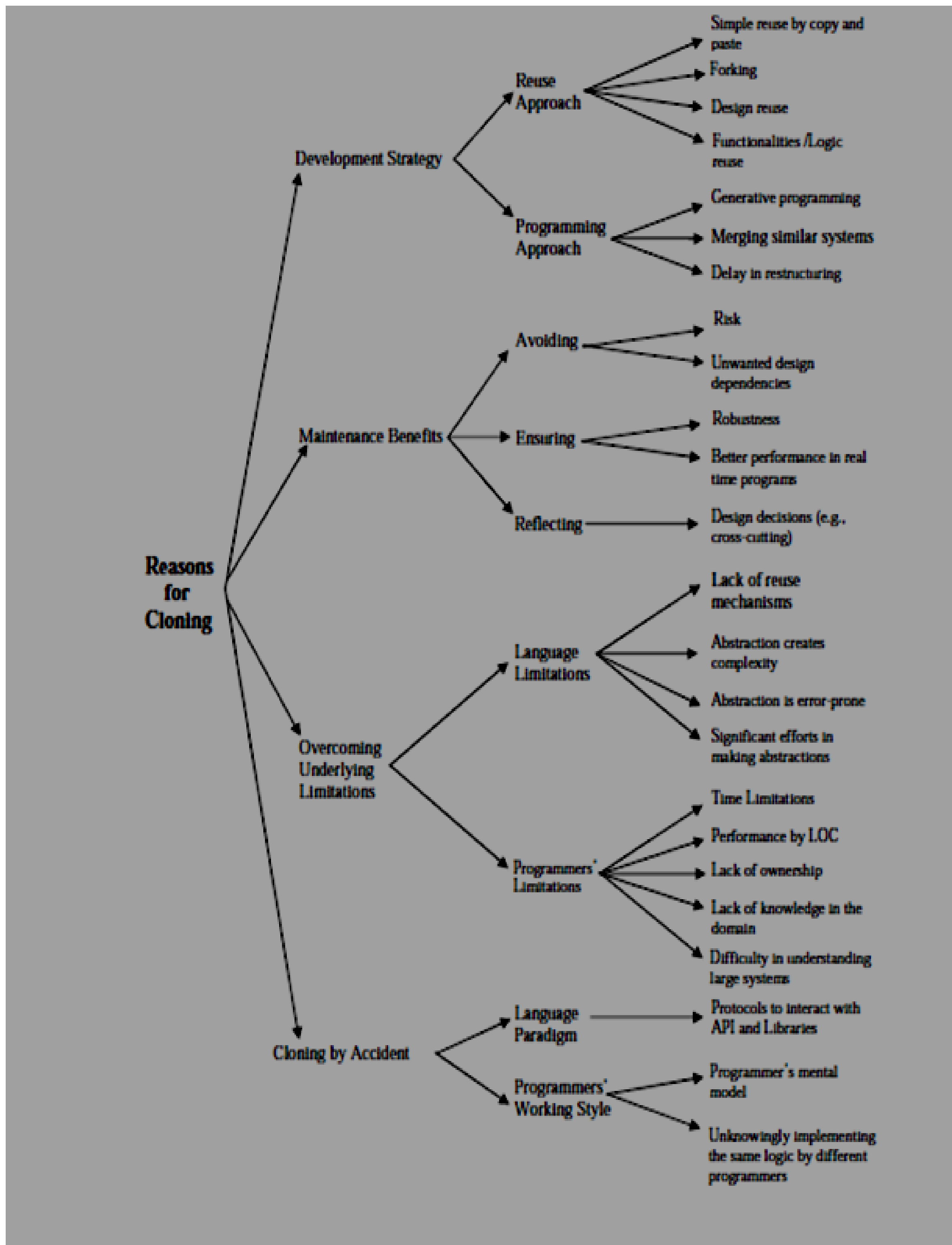


Figure 1.2: Tree diagram for the reason of cloning [5].

1.3 Problems with clones in Code

The unjustified duplicated code gives rise to severe problems:

- If one repairs a bug in a system with duplicate code all possible duplications of that bug must be checked.
- Code copying increases the size of the code, extending compile time and expanding the size of the executable.
- Code duplication often indicates design troubles like missing inheritance or missing procedural abstraction.
- Errors in the organized renaming can lead to unintentional aliasing resulting in latent bugs that emerge up much later.

Thus the software with code clone is difficult to maintain as compared to non cloned code [4].

Considering all these factors, and from the perspective of maintenance it is beneficial to detect clones and remove them by constantly monitoring software. Code clone detection could be useful for maintenance and re-engineering.

1.4 Consequence of Code Cloning

While it is beneficial to perform cloning, software clone has a number of negative effects on the quality of the software. Code clones can have rigorous impacts on the quality, maintainability and reusability 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 [5].

- **Increased Maintenance Work and Cost**

  Because of duplicated code in the system one needs extra 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, then maintenance work should be performed 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 as there is no surety that this bug has been already eliminated from other similar parts at the time of copying or during maintenance.

- **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.

- **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, particularly when the system is running in a busy hardware environment. For example, telecommunication switch, which the software system upgrading will lead an upgrading in hardware as well.

- **Increased Probability of Bad Design**

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

1.5 Motivation and Objective

Maintenance cost of software is generally more than implementation cost. According to Brooks [8] more than 90% of the software cost belongs to software maintenance activities. According to Roy et al. [10] between 7% up to 23% of software systems contain clone code.

Cloning is challenging for software maintenance for several reasons [5]:

- Cloning without cause increases program size. Since many maintenance efforts associate with program size that increases the maintenance effort.
- If changes to duplicate source code segments are performed inconsistently, this can

introduce bugs.

Code clone detection could be useful in several ways such as for the purpose of plagiarism detection, aspect mining as well as decreasing the cost of software maintenance activities. Detection of duplicate code segment increases understandability of software systems and may help system maintainers to enhance code quality of the existing system. Detection of duplicate code segments improves the efficiency of the software maintenance process and decreases maintenance cost [5].

The aim of the thesis is to design and implement a hybrid clone detector tool to detect clones. The novel aspect of the work is by using metric based and token based approach on Java softwares. For calculating metrics Java byte code is used.

In order to achieve this aim the following objectives must be fulfilled:

I. To study and compare the existing software clone detection techniques and various types of clones.

II. To propose a novel code clone detection techniques for Java programs.

III. To design and implement the tool that can automate the work of clone detection for Java based programs.

1.6 Outline of 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 the problem statement and the proposed work.

Chapter 4- Explain the Proposed model and its implementation.

Chapter 5- Explains the experiments performed, evaluate and compare the results achieved.

Chapter 6– Concludes the thesis and describes the future work.

Chapter-2

Literature Survey

This chapter describes in detail the literature survey. It covers basic terminology of Software Clone detection, clone relation terminology, various types of clones, advantage of detecting clones, clone detection process, and various detection techniques and tools.

2.1 Basic Concepts of Clone detection

Clones as the name implies are copied regions of code. However, unlike a biological clone, a software clone may or may not be exactly the same [11]. For a given clone relation, a pair of code portions is called clone pair if the clone relation holds between the portions. And the equivalence class of clone relation is called clone class [6].

2.1.1 Clone Relation Terminologies

Clone detection tools report clones in the form of Clone Pairs (CP) or Clone Classes (CC) or both. These two terms address about the similarity relation between two or more cloned fragments. The relationship between the cloned fragments is an equivalence relation (reflexive, symmetric and transitive relation) [5].

2.1.1.1 Clone Pair

If a clone relation exists between a pair of code fragment is called a Clone Pair. For instance, the first code segment (a) in fragment 1 of Figure 2.2, together with a segment (a) in fragment 2, forms a clone pair. Clones may subsume each other but typically the interest is only in the maximally long clones. For instance, a segment (a) could be joined with the subsequent segment (b) in fragment 1 of Figure 2.2 to form a larger clone together with the joined segments (a) and (b) in fragment [5]. A maximally long clone pair is one whose two fragments can be extended neither to the left nor to the right to form a larger clone.

Figure 2.1 shows the clone pair and clone class relations.

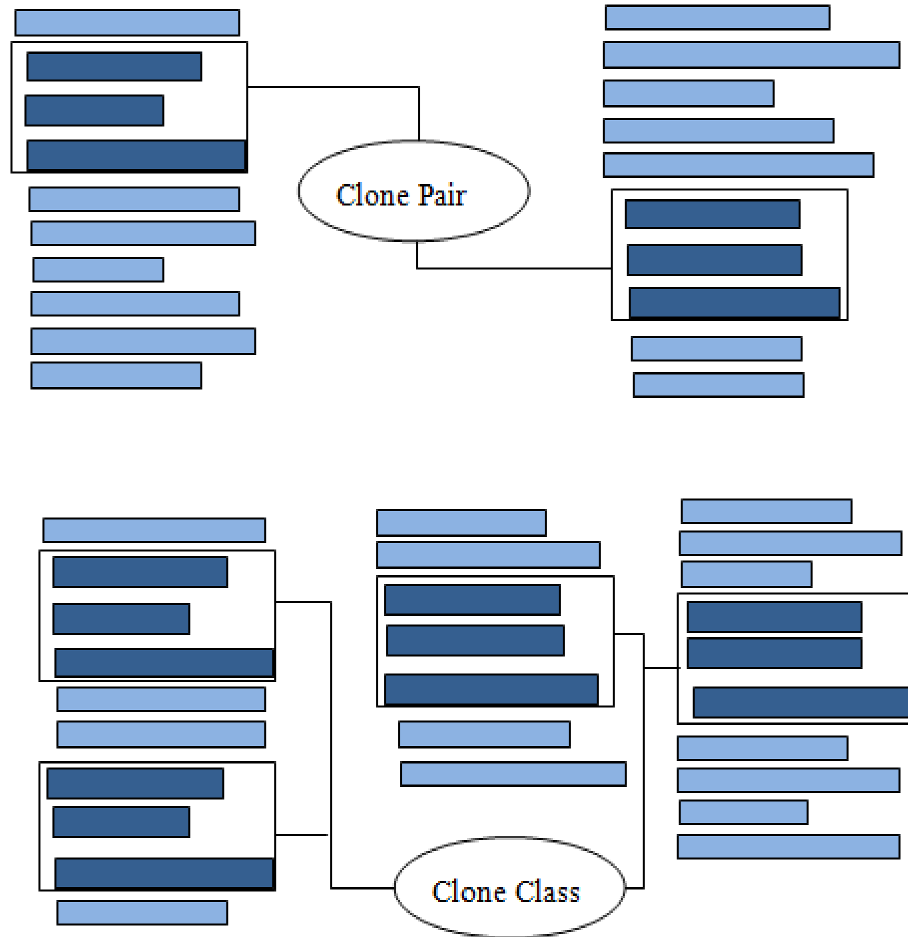


Figure 2.1: Clone Pair and Clone Class [14].

2.1.1.2 Clone Class

The maximal set of code fragments in which any two of the code fragments hold a clone-relation (form a clone Pair) is a Clone Class [10]. For the three code fragments of Figure 2.2, we get the clone class <F1(b), F2(b), F3(a)> where the three code fragments F1(b), F2(b) and F3(a) each form clone pairs with the others, it means, there are three clone pairs, <F1(b), F2(b)>, <F2(b), F3(a)> and <F1(b), F3(a)> [5]. A clone class is therefore the union of all clone pairs which have code portions in common. Clone classes are also called clone communities.

2.1.1.3 Clone Class Family

The group of all clone classes that have the same domain is called a clone class family. Such a clone class family is also termed super clone by Jiang et al [6]. In their context, multiple clone classes between the same source entities are aggregated into one large super clone.

Fragment 1: F1

```
...
for (int i=1; i<n; i++) {      (a)
    sum = sum + i;
}
if (sum <0 ) {                 (b)
   sum =  n - sum;
}
...
```

Fragment 2: F2

```
...
for (int i=1; i<n; i++) {      (a)
    sum = sum + i;
}
if (sum < 0 ) {                (b)
    sum = n - sum;
}
while ( sum <  n ) {           (c)
   sum  =  n / sum ;
}
...
```

Fragment 3: F3

```
...
if (result  < 0 ) {            (a)
   result = m - result;
}
while (result < m ) {          (b)
   result  = m / result
}
...
```

Figure 2.2: Code Example showing Clone pair and Clone Class [5].

2.1.2 Code Clone Types

The general answer of the definition of clone is that two code fragments if they are identical or similar is termed as a clone [5].

According to the different similarities, the clone can be classified into two categories: One type of similarity considers the textual similarity and other considers the semantic similarity in which the clone code must have the same behaviors, means functional similarity. The first kind of clones is often the result of copying a code fragment and then pasting to another location. In this section we consider clone types based on the kind of similarity two code fragments can have:

2.1.2.1 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 [5].

- **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) comments, types and layout and the sentence structures are essentially the same as the original one.
- **Type III:** A Type III clone is a copy with further modifications, e.g. a new statement can be added, and some statements can be removed. The organization of code fragment may be changed and they may even look or behave slightly differently. This kind of clone is hard to be detected, because the fuller context understanding is needed.

2.1.2.2 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 [5]:

- **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.

2.2 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 [5]. The list of some of these is as follows:

- **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. So, this fragment can be incorporated into a library, to announce its reuse potential officially.

- **Bug Detection**

  There is also a close relation between clone detection and software bug detection. Copy pasted software bugs can be successfully detected by clone detection tools.
- **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 [3] 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.
- **Code Compaction**

  Clone detection techniques can assist with fitting code into compact devices (e.g., mobile devices) by reducing source code size [5].

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

2.3 Clone Detection Process

A clone detector must try to find pieces of code of high similarity in 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 [10]. Figure 2.3 displays the phases of the clone detection process.

i. **Preprocessing**

   Initially in the clone detection process the target source is portioned and comparison field is analyzed. The major objectives to be considered in this phase are removing uninteresting segments, then determining the source units and comparison unit. All the source code uninteresting to the comparison phase is filtered in this phase. After deleting the uninteresting code, the left over 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. Such units may need to be further partitioned into smaller units depending on the comparison function of a method.

ii. **Transformation**

In the next stage, the source code 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 [12] or could be very complex by generating PDG representation [5] 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.

iii. **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 similarity. The order of comparison units is used to sum up the adjacent similar units to form bigger units. The output of the comparison unit is a list of matches with respect to the normalized 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 by the location information of the matched fragments in the transformed code.

iv. **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 per location of the preceding phase, it is then converted into line numbers of the original source files.

v. **Preprocessing**

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

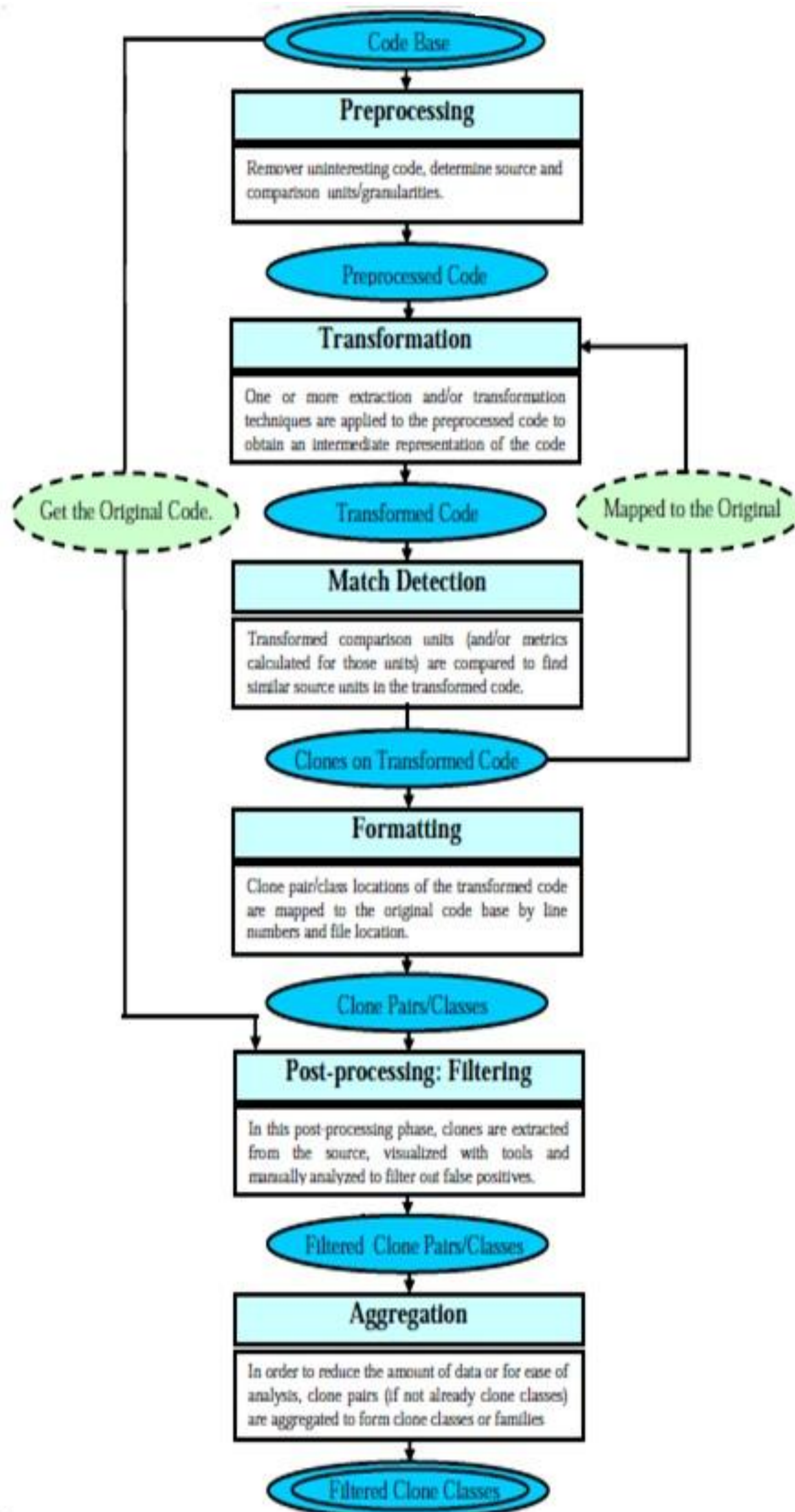


Figure 2.3: Clone Detection Process [5].

vi. **Manual Analysis**

After extracting the original source code the raw code of the clones of the clone pairs subjected to the manual analysis, in order to filter out the false positive.

vii. **Visualization**

The obtained clone pair list is used to visualize the clones with the help of the visualization tool. To speed up the manual analysis in filtering out the false positives visualization tool is used to visualize the clone pair.

viii. **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.

2.4 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 [5] [18]:

2.4.1 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 a sequence of lines or strings. And then, two code fragments are compared with each other to find the matched sequences of text. When a match is found, i.e. two or more code segments are found to be alike, 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 unrefined source code is directly used in clone detection process without any conversion.

However, few of the latest text based clone detection techniques use some of the transformation such as comment removal and whitespace removal. Because text based technique does not perform any syntactic or semantic analysis of the source code it is one of the fastest clone detection approaches. It can easily deal with type 1 clones, with additional data transformation the type 2 clones can also be taken care.

- **Johnson [3]** proposed text-based clone detection technique. This approach uses “fingerprints” on substrings of the source code. Firstly 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 sequence of lines having the same hash value as clones.
- **Ducasse et al. [13]** proposed one of the 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 as diagonals in dot plots. 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.

2.4.2 Token Based Technique

The token based technique is similar to the 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 tokens [7]. After data values and identifier are substituted by various special tokens the token sequences of these segments are compared efficiently through a suffix tree algorithm. The output is also presented in dot plot graph. This technique is slightly slower than text based method because of the tokenization step. However, applying the suffix tree matching algorithm, the time complexity is similar as text based technique. By breaking the line into tokens it can easily detect both type 1 and type 2 clones [5][18].

- **Kamiya et al. [7]** has proposed a clone detection algorithm and implemented a tool named CCFinder (Code clone finder). The tool makes a token sequence from the input code through a lexical analyzer, applies the rule-based transformation to the sequence and uses 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 subsequence. The clone detection is performed by searching the leading nodes on the tree.
- **Zhenmin Li et. Al. [15]** proposed another state of the art token-based clone detection technique ie *CP-Miner* where a frequent subsequence mining technique

[5] is used for identifying a similar sequence of tokenized statements. Due to sequential analysis in *CCFinder* and *Dup*, they are generally fragile to statement reordering and code insertion. A reordered or inserted statement can break a token sequence which may otherwise be regarded as duplicate to another sequence. These limitations are overcome in *CP-Miner* by using a frequency subsequence mining technique where a frequent subsequence can be interleaved in its supporting sequences.

- **Baker [12]** proposed parameterized matching using suffix trees with Dup tool. 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, structure or literal. The output of this first step is a parameterized string or p-string. Once the p-string is formed, a criterion is to be decided that two sequences in this p-string are a parameterized match or not. Two line segments are a parameterized match if one can be transformed into the other by applying a one-to-one mapping of 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 and parameterized matches. All that is left for the last step is to locate maximal paths in the p-suffix tree that are longer than a predefined character length.

2.4.3 Abstract Syntax Tree (AST) – Based Technique

In the tree-based approach a program is parsed into a parse tree or an abstract syntax tree (AST) with a parser of the language of interest. Similar sub trees are then searched in the tree with some tree matching techniques and the corresponding source code of the similar sub trees are returned as clones pairs or clone classes. The parse tree or AST contains the complete information about the source code. Although the variable names and literal values of the source are discarded in the tree representation, more sophisticated methods for the detection of clones still can be applied.

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.

- **Yang [17]** has proposed approach for finding the syntactic differences between two versions of the same programs by generating a variant of the parse tree for both the versions and then applying the dynamic programming approach in searching similar sub trees.
- **Baxter et al. [19]** 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 sub trees by characterization metrics based on a hash function through tree matching [12]. Source code of similar sub trees is then returned as a clone. The hash function enables one to do parameterized matching, to detect gaps clones and to recognize clones of code portions in which some statements are reordered.
- **Wahler et al. [20]** 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.

2.4.4 Program Dependence Graph (PDG) – Based Technique

Control and data dependencies of a program can be represented as a program dependency graph. Because it records the relationship between the data and structure, it can be used to trace the modification after programmer’s copy and paste activities [21]. The PDG based technique takes one step further than AST based method that is 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 is obtained from a program, isomorphic sub graphs matching algorithm is implemented in order to find similar sub graphs which are returned as semantic clones. The clone pair can be extracted from the isomorphic sub graph. With control and data dependency information, the 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 the AST parsing process is a very expensive process on both time and memory.

- **Komondoor et al. [21]** proposed an approach that uses 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 [22]** presented an approach to identify similar code fragments in programs based on finding similar sub graphs in attributed directed graphs. This technique 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 sub graphs in the PDG.
- **Liu et al. [7]** 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.
- **Gabel et al. [23]** proposed a scalable detection algorithm for finding semantic clones. This algorithm is based on selecting PDG sub graph based on its related structured syntax. This algorithm is implemented in a tool and its focus is only on semantic clones rather than finding copy paste activities.

2.4.5 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. In the beginning 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 metric values are compared to find clones. Then the metric were calculated from names, layout, expression and simple control flow of the function. A clone is detected only when pair of whole function bodies that have similar metric values are identified.

- **Jean Mayrand et al. [16]** the main goal of this paper was to detect functional clones automatically from the source code of any language using the Metric-based code clone detection technique. This can be accomplished by using a DATRIX tool framework which is a source code analyzer tool set whose main purpose is to change the source code into some Intermediate Representation Language. Out of which only control flow metrics and data flow metrics were selected because they

provide internal characteristics information about functions. For automatic detection, this proposed approach experimented on two telecommunication monitoring system in which for finding function clone, following points of comparison for the clone are performed :

- Name of Function
- Layout of Function
- Expression in Function
- On the basis of these points of comparison any two functions can be considered as equal, similar or distinct.
- **Kontogiannis et al. [24]** proposed two different ways of detecting clones. One approach uses a direct comparison of metric values as a surrogate for similarity at the granularity of begins end blocks. Five well known metrics 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.
- **Patenaude et al. [25]** use a very similar method-level metrics to extend the Bell Canada Datrix tool to find Java clones. In this approach the software evaluation technique find parts in large Java systems that have unusual characteristics. This will help the software evaluator to find specific sections of a system that will require more attention and to remove false positives manual operations are performed.
- **F. Calefato et al. [26]** proposed a Metric- based approaches that have also been applied to finding a clone in web documents .The idea is using a semi automated approach to identify the cloned function within scripting code of web applications and the results obtained is quite effective in identifying functional clones in web applications and can be applied to prevent clone from spreading.

2.4.6 Hybrid Based 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 [27]** provides a hybrid approach that combines syntactic techniques based on AST metrics and semantic techniques (using call graphs) in

combination with specialized comparison functions.

- **Tairas et al. [28]** proposed a method to find function-level clones in a program that combines abstract syntax trees (ASTs) and suffix trees approach. The AST provides the structural image of the code after the lexical analysis process. The AST nodes are used to create a suffix tree which allows analysis on the nodes to be performed rapidly. This approach uses 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 created, which in turn reveal matches in the code.
- **Koschke et al. [29]** proposed an approach in which, instead 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.
- **Jiang et al. [30]** 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.
- **Kodhai. E et al. [31]** proposed the metrics based method with a simple textual analysis technique called a hybrid technique which is used to detect function clones from C language taken as source code. This approach is able to detect just type -1 and type- 2 clones from the system and a tool is developed on the basis of this approach using a Java programming language. The Textual comparison is performed to detect exact matched clones of Type-1 and Metric value method helps in detection of near miss clones of Type-2. In metrics technique a set with some of interested method level metric is defined and based on this set descriptive statistics of the metric values calculated for the various identified methods.
- **Cordy et al. [32]** proposed a hybrid method that combines language-sensitive parsing with language-independent similarity analysis to yield structurally meaningful near-miss clones and implemented it as Nicad tool. It is a 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.

Table 2.1: Clone Detection Approaches.

Type of Comparison	Portability	Type of clones	Efficiency	Integrality
Text Based	Good	Syntax (Type 1 Clone)	High	Depends on algorithm
Token Based	Average	Syntax (Type1 and Type 2)	Low	Good
Abstract Syntax Tree Based	Poor	Syntactic(Type1, Type 2 and Type3)	High	Depends on algorithm
Program Dependence Graph Based	Poor	Syntactic and Semantic	High	Medium
Metric Based	Good	Syntactic	High	Medium

2.5 Cloning and Software Maintenance

Code clones are considered one of the bad smells of software system [2] and indicators of poor maintainability. Various studies show that the software system with code clones is difficult to maintain as compared to non cloned code software system [1] [33] [34]. The impact of clones is of special concern from a software maintenance point of view. [33] [34].

Fowler [2] suggests that code duplication or cloning is a bad smell and thus one of the major indicators of poor maintainability [33]. Without tracking clones over time, identifying and consistently changing clones can be problematic and increases the maintenance cost [35]. An intrinsic property of software in a real-world environment is its need to evolve. As the software is enhanced, modified and adapted to new requirements, the code becomes more complex and drifts away from its original design. That's why the major part of the total software development cost is devoted to software maintenance [36]. The impact of cloning on software maintenance is problematic and is of great concern. So the need for identification of clones is necessary and this process is called Clone detection.

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 [3][13] with little or no code normalization and other techniques, use a laxer to make a token sequence [7] 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 [19][20].

Some of the other techniques [25] 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 [21] [22].

Discussions in the literature review indicate that most of the existing hybrid techniques used tree based detection. Abstract Syntax tree (AST) based techniques, is bulky process and require sub tree comparison and full parser. Text Based techniques are also used in a hybrid approach to remove some false positives. But it can detect only type1 clones.

The metric based detection technique is more scalable and accurate for large software system. The metric based comparison is straightforward as compared with AST/PDG based clone detection technique and Token based techniques can find clones with high accuracy and precision. The token based comparison is applied to remove false positives.

Problem identification:

i. A better approach is needed for the detection of clones.

ii. An approach that can detect clone with the help of hybrid approach and the process of clone detection is light weight too.

iii. An approach that can be used as a pre-screening step to reduce the complexity of the approaches like AST/PDG based approach.

iv. An automated approach is needed to detect potential clones. This will definitely contribute in reducing the maintenance efforts.

v. A tool is needed that can calculate metrics for a Java program and these

metrics are helpful in clone detection for Java based programs.

This thesis presents a hybrid clone detection approach with user friendly interface that is capable of detecting clones in software by analyzing the various software metrics and then comparing each line of one file (application program) with every line of the second file (application program) with respect to tokens.

The presented approach use metric based and token based technique to detect clones and divided into two stages. In the first stage metric based technique is used for the selection of potential clone. Potential clones are selected on the basis of metric match and after this potential clones is further processed with token based technique. The potential clones are compared by a token based approach to find whether they are actually clones or not. A clone detector tool MCD Finder is proposed to calculate the metrics of Java code and these metrics are compared to detect potential clones

Chapter 4

Proposed Work and Implementation

Cloned software is a major problem in large software systems that have evolved over a long period of time. The problem is not acquiescent 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 an automated tool for the detecting clone is developed.

Current work presents a practical method, for detecting exact clones for Java program's source code. In this work a hybrid approach using a combination of metrics based and token based technique is used for detection of clones.

Firstly metric based approach is applied to detect the potential clones and only on these potential clones token based approach is applied to detect clones.

A tool MCD finder is proposed to calculate metrics of Java programs and compare these metrics to find the potential clones. MCD Finder takes byte code as an input rather than the Java source code because the byte code is platform independent. The opinion is derived from the fact that while transforming the code into byte code the compiler would map ‘for’ and ‘while’ loops into the same unified representation. As compiler will generate a unified representation of code and this representation reduces dissimilarity in code segments, so it yields better results.

4.1 Design of Solution

This Code Clone detection tool takes two Java byte code files as an input and gives the output as the function and class metrics. After selecting the potential clones through metrics based approach, the token based comparison is applied on the potential clones to find clones. Figure 4.1 and Figure-4.2 explains the flow diagram and architecture of the tool respectively.

4.1.1 Flow Diagram Of Proposed Work

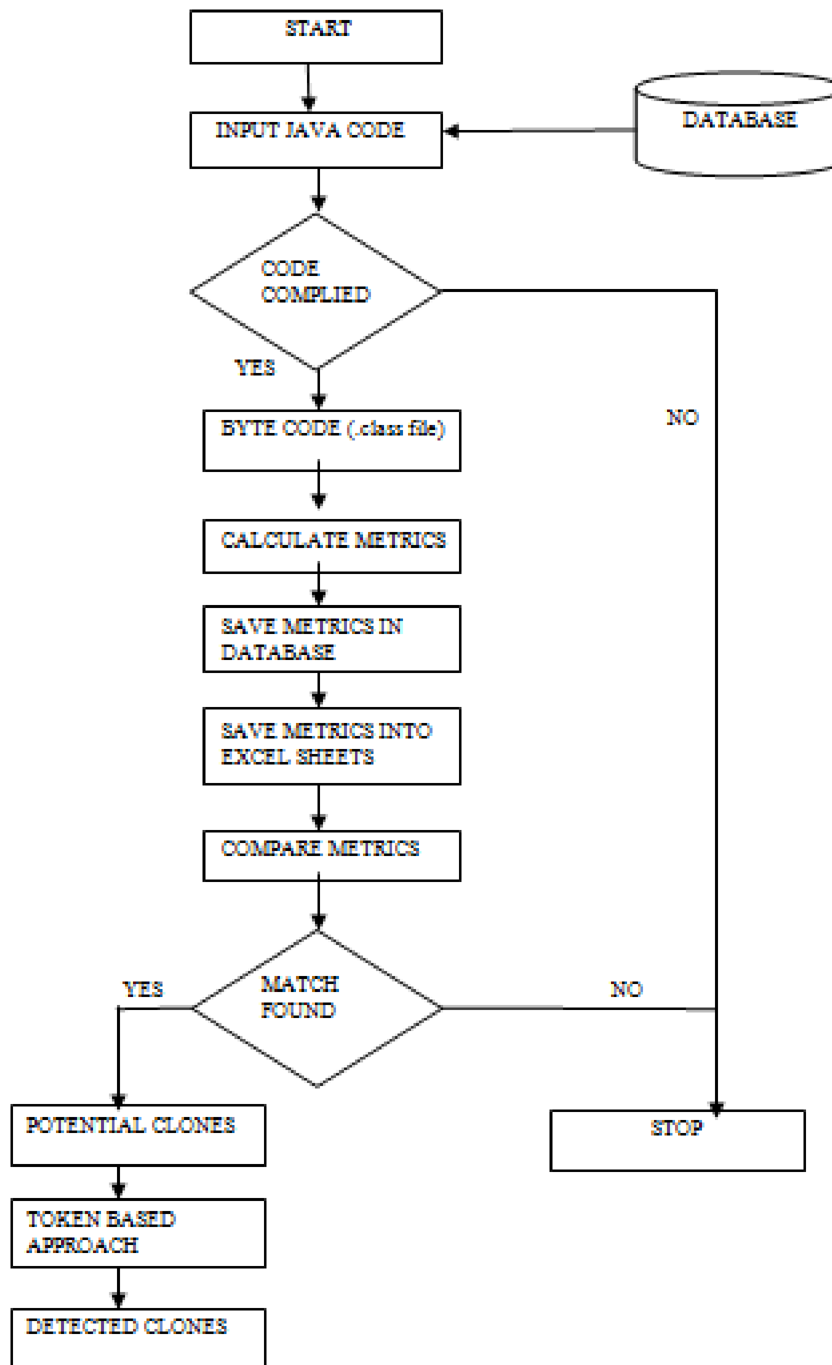


Figure 4.1: Flow Diagram of MCD Finder.

4.1.2 Architecture of Proposed Work

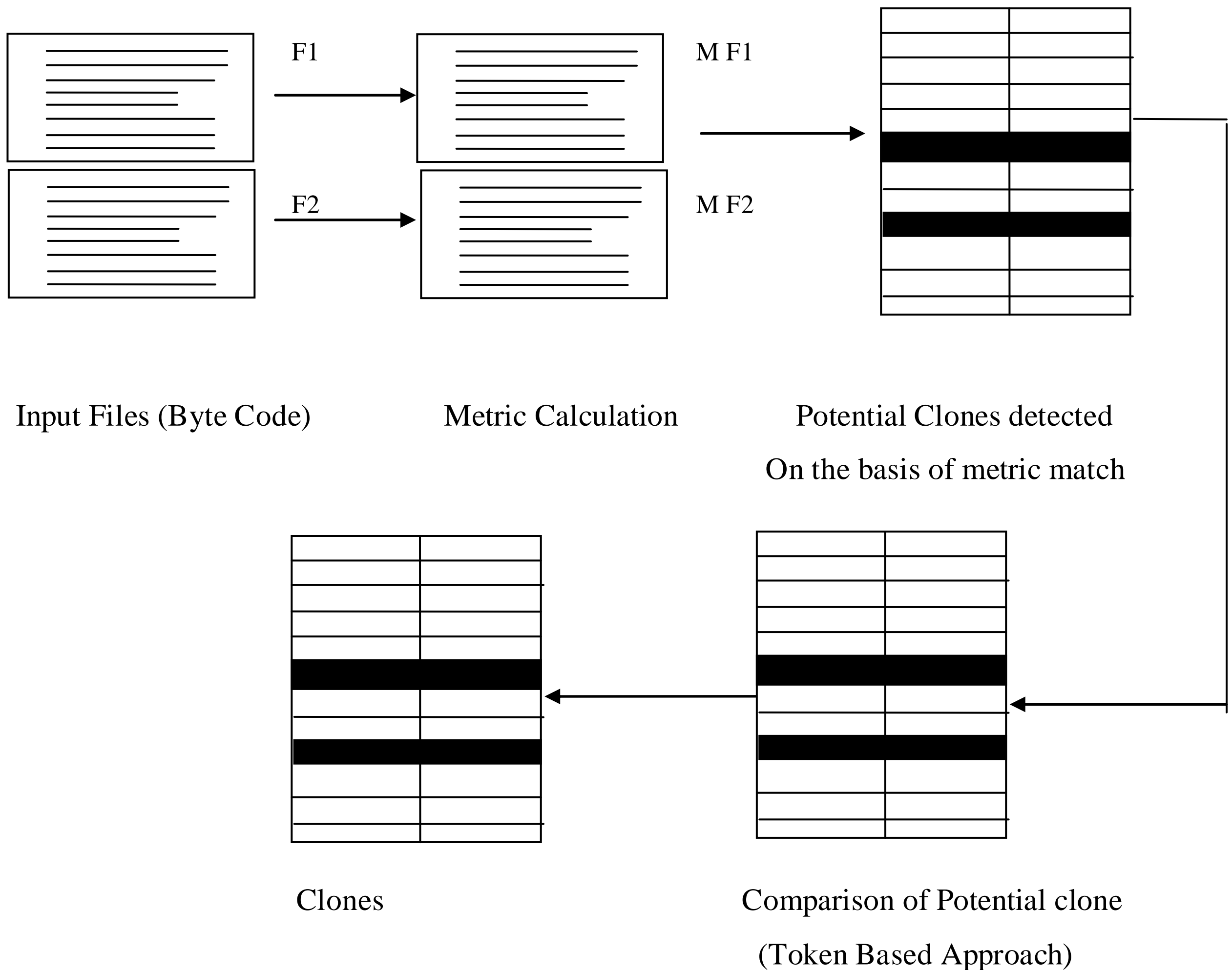


Figure 4.2: Architecture of Proposed Work.

4.2 MCD Finder

MCD Finder stands for Metric based Clone Detection. MCD Finder takes byte code of the source code as an input rather than the Java source code. This tool is developed in Java language. After calculating metrics of a Java program it compares the metrics to find potential clones. Software metric is one of the important aspects of software engineering acts as an indicator for software attributes [37]. It plays an important role in understanding the important concepts in the field of software engineering. The concept of software metrics came into existence in 1970, the credit of which goes to Wolverton

who performs a research on production ratio of the programmer by using the concept of LOC i.e. line of code. The proposed method consists of three phases as Adaptation Phase, Computation Phase and Measurement Phase. Before describing the implementation of MCD Finder, its algorithm should be described first which is described in section 4.2.1 and section 4.2.2.

4.2.1 Algorithm for Metric Calculation

1. **Input**: Byte Code File for Java Program
2. **Output**: Calculation of various metrics
3. **Precondition**: Java Program should be compiled before calculating the metrics.
4. Input class name from User.
5. Use Reflection to get the method names their access modifier and number of arguments. Use Method class for it.
6. while i<methods.length do
7. methodName <- Get the name of method
8. functionCounter++;
9. return methodName after typecasting.
10. parameterType<- Get the name of Parameters
11. while k<paramettertypes.length do
12. Typecast all the parameters as String and return.
13. Use Reflection to get the Constructor names.
14. while i < constructor.length do
15. ConstructorName <- Get the name of constructor
16. Class[] parameterTypes = constructor[i].getParameterTypes();
17. while k < parameterTypes.length do
18. parameterName <- get the parameter name
19. Use Reflection to get the Variable names and their acess modifier with the help of Field class.
20. while i < fields.length do
21. fieldName <- get the name of field
22. ClassVariableCounter++;

23. if(fields[i].getModifiers()==2)
24. privateVar++;
25. if(fields[i].getModifiers()==0)
26. friendVar++;
27. if(fields[i].getModifiers()==1)
28. publicVar++;
29. if(fields[i].getModifiers()==4)
30. protectedVar++;
31. End
32. Parse the class and get the method name without access modifier. This line will be a call to the function. Increment the counter and dump the data into database
33. Read the Java file of the class to calculate the Line of Code (LOC).

4.2.2 Algorithm for Comparison of Excel sheets

1. **Input** 2 .xls for which the comparison is to be done.
2. **Output:** Similar Metrics
3. Repeat the step 4 until the file is read.
4. For each cell of excel file call ValidCode method, which is used to check whether data is present in cell or not.
5. if Valid== Null then
6. buffer <- Null
7. else
8. buffer<- Value that is read
9. Enter the data in the database.
10. Repeat step 4 to 9 for second .xls file.
11. For each cell in the database
12. if xls1.data==xls2.data then
13. MsgBox " Potential Clone
14. else
15. MsgBox "no clone"
16. End

4.3 Implementation of Proposed Tool

4.3.1 Adaptation Phase

This is the first phase of the proposed method. In this phase the code is adapted according to requirement of tool means tool requires as input the .class file for the program. So the Java source code should be complied to make it adaptable according to tool's requirement. The tool takes a byte code as an input program rather than the Java source code because the byte code is platform independent. Rather than using AST and PDG approach to detect type-3 and type-4 clones which are quite complex, the proposed approach tries to detect clones by applying token based technique on potential clones detected with the help of proposed tool. The perception is derived from the fact that while converting the code into byte code the compiler would map 'for' and 'while' loops into the same unified representation. As compiler will generate a unified representation of code by reordering, and this unified representation reduces dissimilarity in code segments, so it yields better results. And the adaptation process is shown in Figure 4.3.

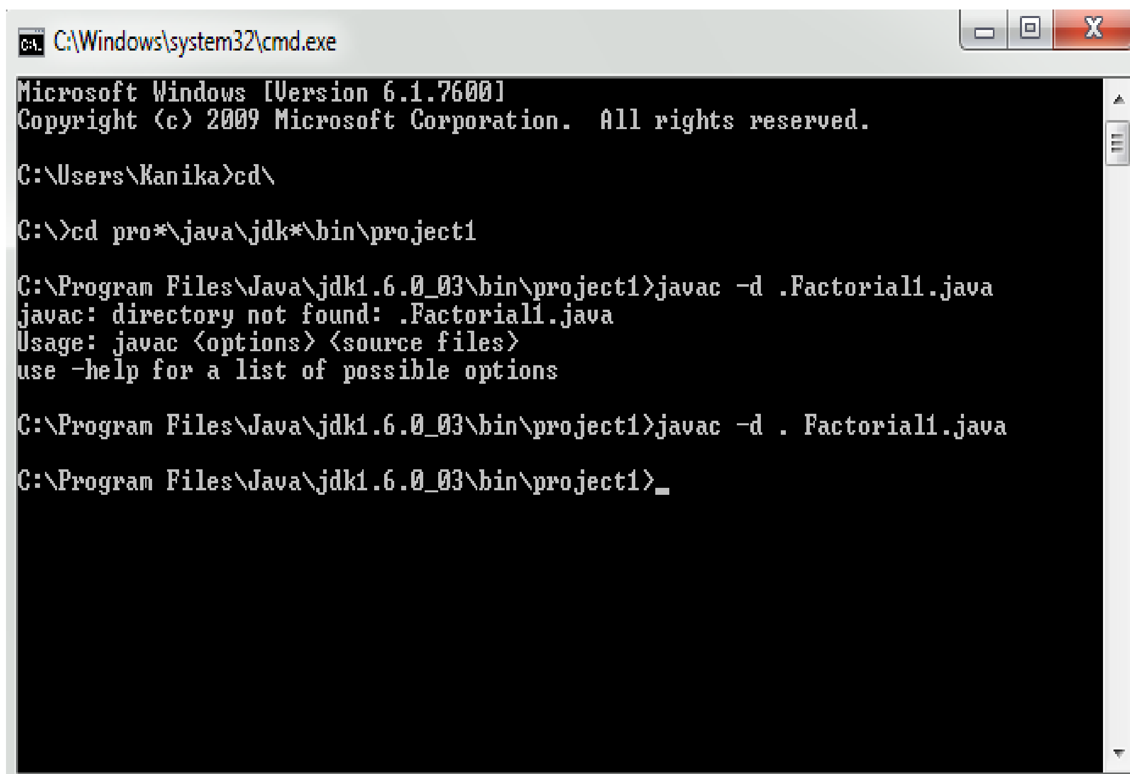


Figure 4.3: Adaptation Phase- Compilation of Java program.

Figure 4.3 shows the compilation of Java program. And the byte code is generated and the representation of byte code is shown in Figure 4.5. The program for which the byte code is generated is shown in Figure 4.4.

```
package a;

import java.io.*;

class Factorial1{

  public static void main(String[] args) {

  try{

  BufferedReader object = new BufferedReader(new
InputStreamReader(System.in));

  System.out.println("enter the number");

  int a= Integer.parseInt(object.readLine());

  int fact= 1;

  System.out.println("Factorial of " +a+ ":");

int i=1;

while(i<=a)

  {

  fact=fact*i;

i++;

  }

  System.out.println(fact);

  }

  catch (Exception e){}

  }
```

Figure 4.4: Java Program to find Factorial.

4.3.1.1 Byte Code Overview

The Byte code is similar to other low level language. Java byte code uses machine instruction to stimulate basic functionalities. Java byte code is generated by using javac compiler provided by JDK environment. In the proposed approach byte code is used to calculate various metrics for the source code as it is platform independent and the other reason to use byte code is that a unified representation of source code is generated and various syntactic dissimilarities of various loops and conditions are also changed in a unified format shown in Figure 4.5 So up to some extent it can detect some semantic clones and can also eliminate some challenges of semantic clones.

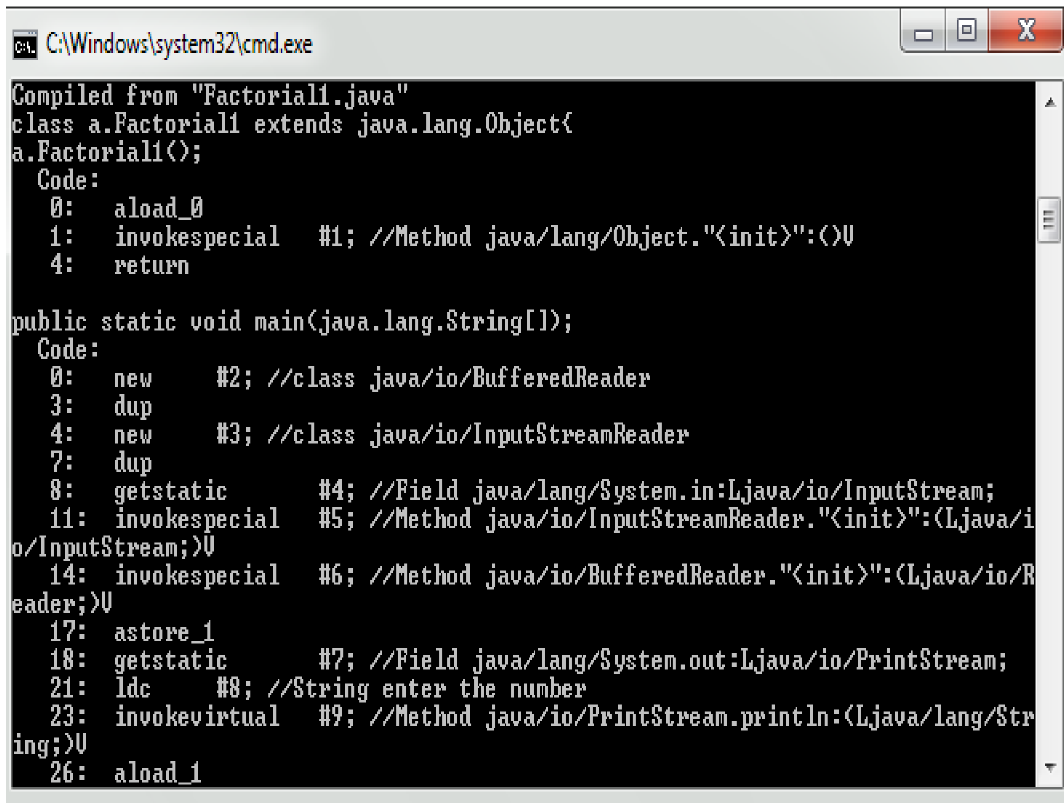


Figure 4.5: Byte Code Representation of Program.

4.3.2 Computation Phase

After the adaptation phase the next step is to perform computation. Computation is in the form of various metrics that helps in identifying the potential clone. Bruno Lague et al.

[25] provided various metrics to perform Java software evolution and clone detection and these metrics are calculated with the help of this tool. Besides these metrics some object oriented metrics are also calculated from the proposed tool that helps in identifying potential clones. Figure-4.5 depicts that we have to open a file and the metrics that were listed by Bruno Lague et al. [25] is calculated. And the metrics that are calculated with the help of proposed tool are in the form of class metrics and function metrics as shown in Table 4.1 and Table 4.2.

4.3.2.1 Class Level Metrics

Table 4.1: Class Level Metrics.

Abbreviations	Description
Totalfunc	Total Number of functions in class
Totalif	Total number of conditional statements in class
LOC	Line of Code
TotalV	Total number of variables in class
pubV	Total Number of public variable in class
priV	Total Number of private variable in class
proV	Total Number of protected variable in class
friV	Total number of friend variables in class

4.3.2.2 Function Level Metrics

Table 4.2: Function Level Metrics.

Abbreviations	Description
Functionname	Name of functions present in class
Localvariable	Number of variables present at function level
LOC	Total numbers of lines in a function
Return Type	Return type of function
Total Parameter	Number of arguments passed to function
Call To Function	How many times the function is called

4.3.2.3 Implementation of Computation Phase

Now, the implementation part of computation phase is shown below. Figure 4.6 shows the first page of MCD Finder. And the first page that is opened is filechooserdemo in which .class file is chosen for calculating metrics.

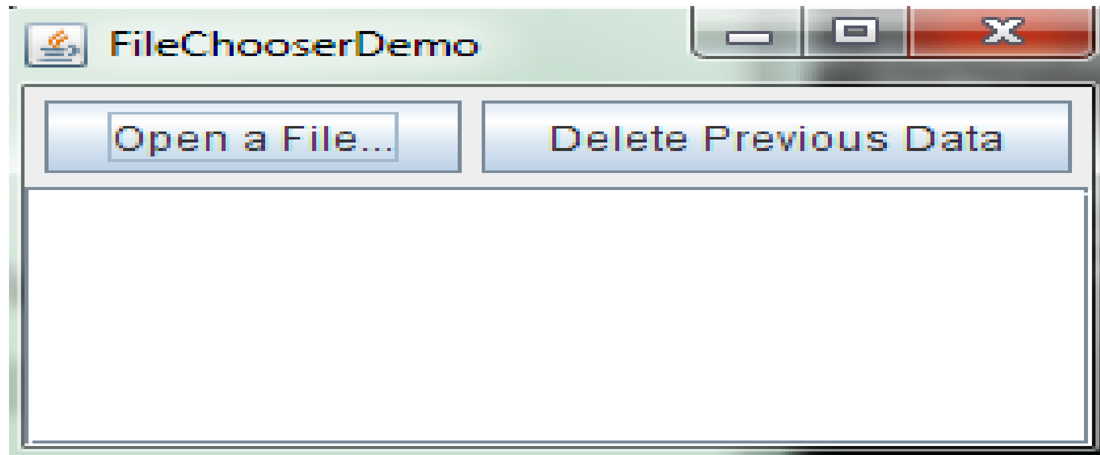


Figure 4.6: Startup Page.

The second button delete is provided to delete the previous data that is present in the database because MCD Finder provides a provision to save the calculated metrics in a database.

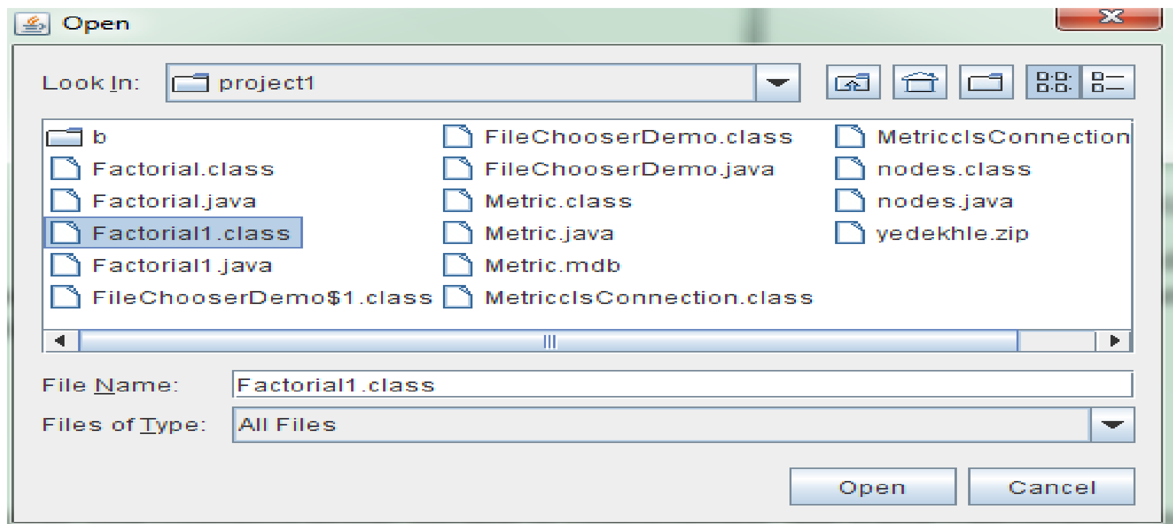


Figure 4.7: Choosing .class file.

When we open a file the metrics calculated will be stored in a database. Now these calculated metrics are mapped into the excel sheets. This mapping to excel sheet is made to store the data for future use and comparison is applied to these excel sheets. And the calculated metric is shown in Figure 4.8 and Figure 4.9. Metrics are calculated both at the function level and class level.

ID	ClassName	totalfunc	totalif	LOC	totalV	pubV	priV	friV	proV
23	Factorial.java	1	0	18	0	0	0	0	0

Figure 4.8: Metrics at Class Level.

classname	functionnam	localvariable	LOC
Factorial.java	main	2	14

Figure 4.10: Metrics at Function Level.

4.3.3 Measurement Phase

After the computation phase is completed, and the calculated metrics are mapped into excel sheets and then the comparison is performed to select potential clones. The comparison is performed on the basis of similarity of the metric value. This tool represents as output which metric's value is same and on the basis of these results potential clones are identified.

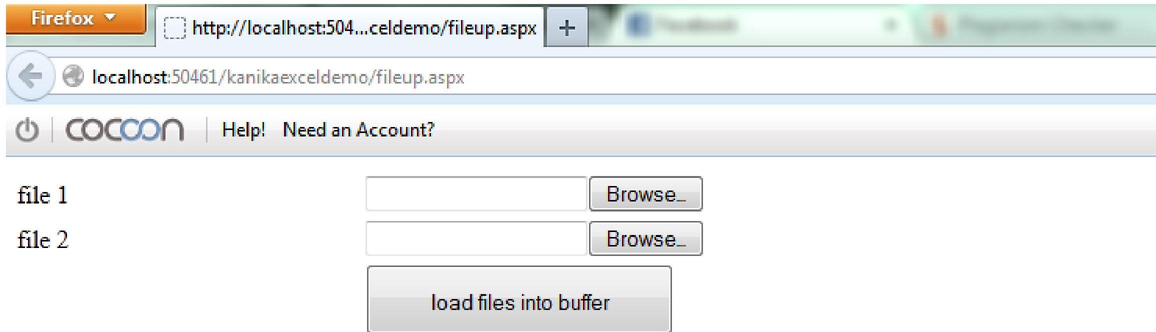


Figure 4.10: Add two .xls Files for comparsion.

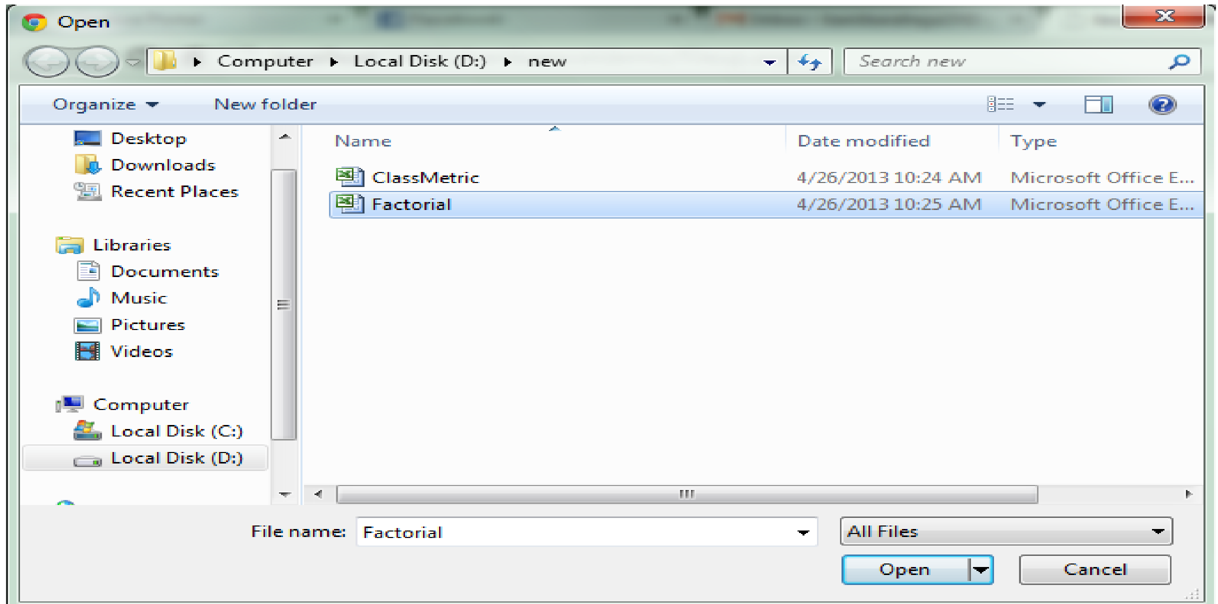


Figure 4.11: Choosing .xls File.

Figure 4.10 shows that selecting the two excel files for which comparison is performed. Figure 4.11 shows browsing the file and then selecting it. Figure 4.12 shows that after loading the files in buffer, there is an option to compare the files. Figure 4.13 shows the comparison of various class level metric on the basis of similarity. Similarly the function level metric is calculated and compared to find the similarity. After analyzing the results obtained from a comparison of metrics at class level and function level, potential clones are detected.

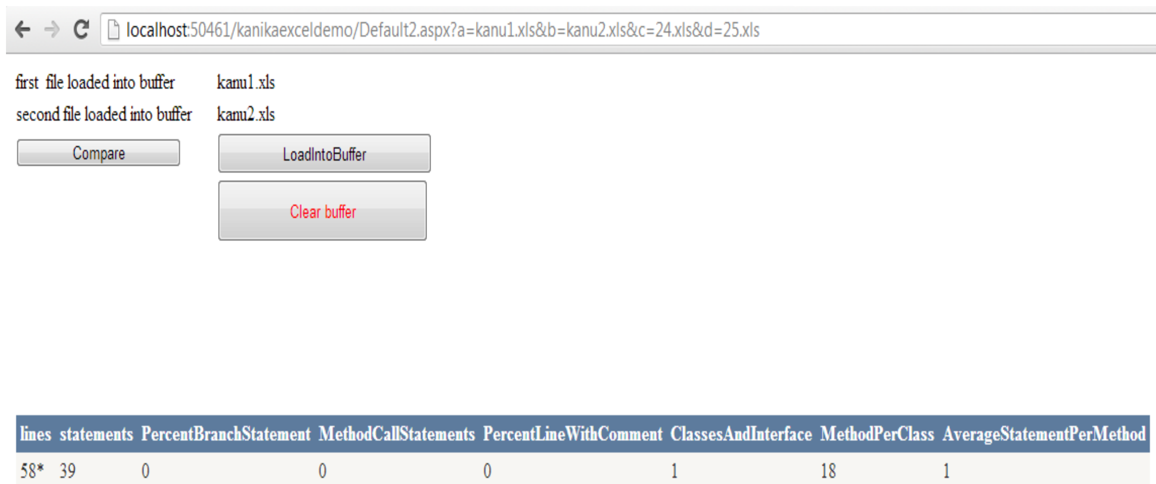


Figure 4.12: Loading Excel Files in Database.

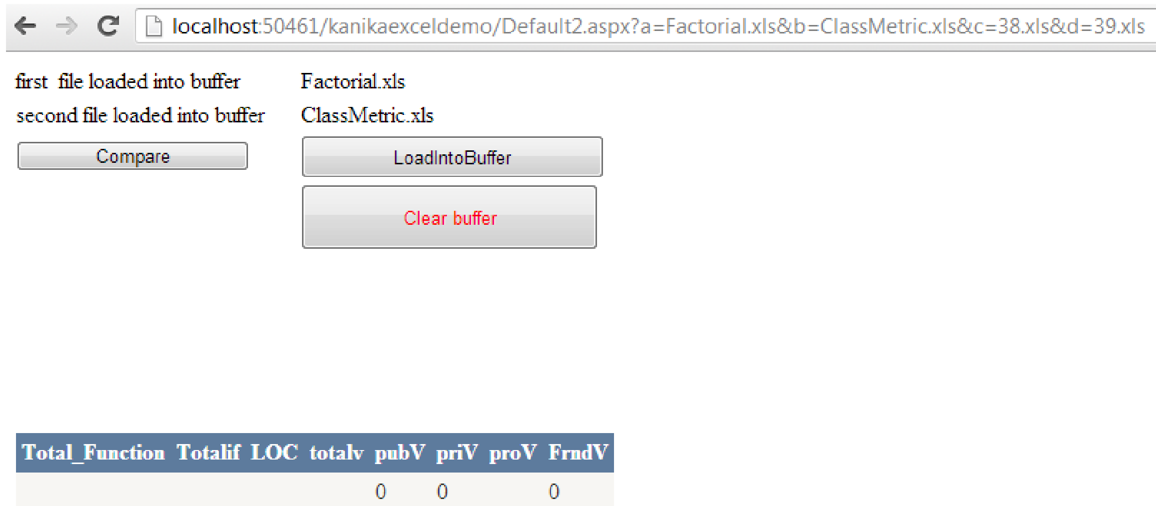


Figure 4.13: Comparison of Class Level Metrics.

4.4 Token Based Approach on Potential Clones

In the last step if the metric value matches the token based technique is applied to the Java source file.

Clone detection based on lexical tokens involves minimal code normalization and gives efficient results, but is computationally expensive because of the large number of tokens that needs to be compared [38]. But in the proposed approach MCD Finder is used before applying the token based approach as a prescreening step to reduce complexity and the proposed work is lightweight also.

And CCFinder [7] is used to detect clones. CCFinder is a command-line code clone detection tool. Gemini is a GUI application which provides interactive code clone analysis with a scatter plot [7].

4.4.1 Clone Detecting Process Using Token Based Approach

Clone detection is a process in which the input is source files and the output is clone pairs. The entire process of our token-based clone detecting technique is shown in Figure 4.14.

The entire process completes in four steps: Lexical Analysis, Transformation, Match Detection and then clone detection in terms of clone pairs/clone class.

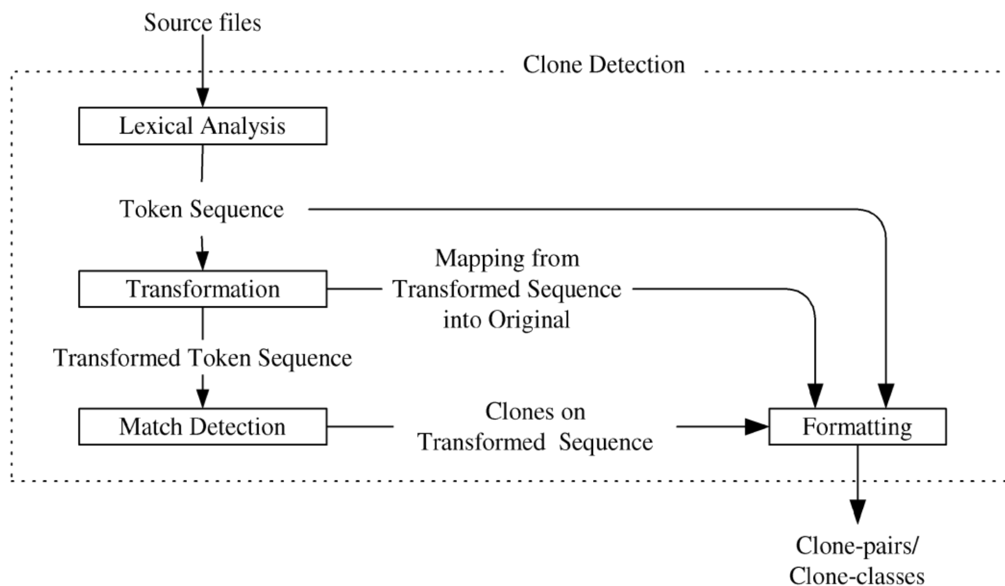


Figure 4.14: Clone Detection Approach using Token based Approach [7].

4.4.2 Implementation using Token based Approach

The token based approach has been applied to remove false positives as metric based approach does not work directly on the code.

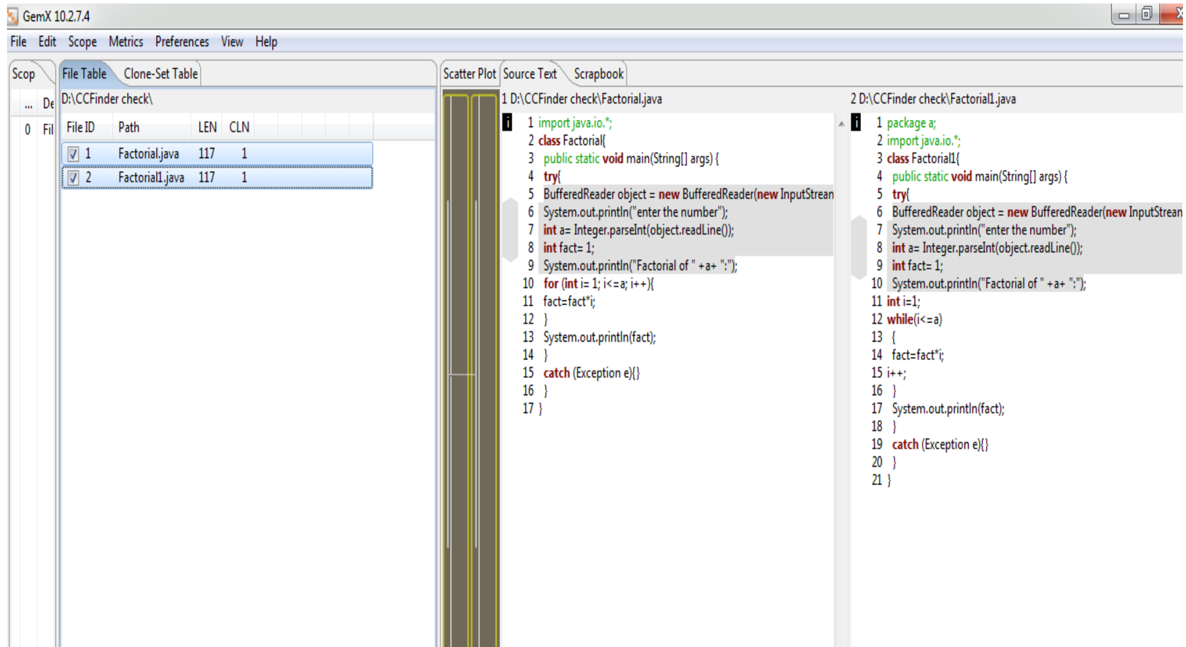


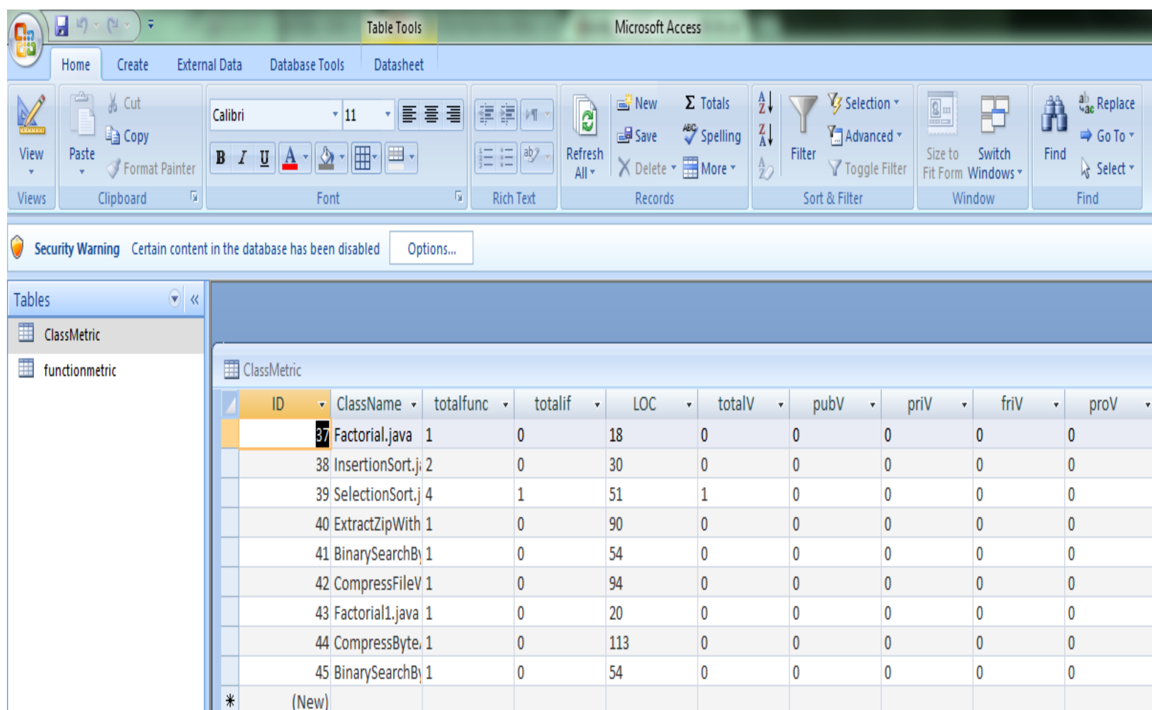
Figure 4.15: Result of Clone Detection Using CCFinder [7].

Chapter 5

Experimental Results and Discussion

The files written in Java language are taken as input. Moreover byte code is used as input to the tool and byte code uses the same unified representation. The opinion is derived from the fact that while transforming the code into byte code the compiler would map ‘for’ and ‘while’ loops into the same unified representation. As compiler will generate a unified representation of code and this unified representation reduces dissimilarity in code segments, so it yields better results. So up to some extent the tool will be able to detect semantic clones as well.

First of all metric based approach is applied to detect potential clones. The class metric's value obtained through this tool is shown in Figure 5.1. The function metric’s value is shown in Figure 5.2.



ID	ClassName	totalfunc	totalif	LOC	totalV	pubV	priV	friV	proV
37	Factorial.java	1	0	18	0	0	0	0	0
38	InsertionSort.j	2	0	30	0	0	0	0	0
39	SelectionSort.j	4	1	51	1	0	0	0	0
40	ExtractZipWith	1	0	90	0	0	0	0	0
41	BinarySearchB	1	0	54	0	0	0	0	0
42	CompressFileV	1	0	94	0	0	0	0	0
43	Factorial1.java	1	0	20	0	0	0	0	0
44	CompressByte	1	0	113	0	0	0	0	0
45	BinarySearchB	1	0	54	0	0	0	0	0
(New)									

Figure 5.1: Class Metrics Value in Database.

Now the function metrics are also calculated by the tool which is used for finding similarity at function level. Now these metric's values are compared one by one to identify similarity. These metrics values are mapped to excel sheets for comparison. Various calculated metrics values are shown in Table 5.1 and Table 5.2.

classname	functionnam	localvariable	LOC	ReturnType	TotalParame	CallToFuntic
Factorial.java	main	2	14	void	1	0
InsertionSort.j	insertion_srt	2	11	void	2	1
InsertionSort.j	main	2	16	void	1	0
ExtractZipWith	main	2	65	void	1	0
BinarySearchCl	main	3	37	void	1	0
CompressFileV	main	3	70	void	1	0
Factorial1.java	main	3	17	void	1	0
CompressByte.	main	2	90	void	1	0
BinarySearchCl	main	3	37	void	1	0
SelectionSort.j	main	2	9	void	1	0
SelectionSort.j	fillArray	0	3	void	2	1
SelectionSort.j	selectionsort	3	15	void	2	1
SelectionSort.j	printarray	1	4	void	2	1

Figure 5.2: Function Metrics Value in Database.

Table 5.1: Class Metrics Values for tested program.

Name of Program	**Total func**	**Tota lif**	**LOC**	**total V**	**pub V**	**pri V**	**pro V**	**frn V**
Factorial (while)	1	0	18	0	0	0	0	0
Factorial (for)	1	0	20	0	0	0	0	0
Insertion Sort	2	0	30	0	0	0	0	0
Selection Sort	4	1	51	1	0	0	0	0
Adler32-Compress File	1	0	94	1	0	0	0	0
Deflater- Compress byte array	1	0	113	0	0	0	0	0
CRC32- Extract Zip	1	0	90	0	0	0	0	0
BinarySearchByteArray	1	0	54	0	0	0	0	0
BinarySearchCharArray	1	0	54	0	0	0	0	0

Table-5.2 Function Metrics Values for tested Programs.

Name Of Program	Function Name	Local Var	LOC	Return Type	Total P	CallToFunc
Factorial (while)	main	2	14	void	1	0
Factorial (for)	main	3	17	void	1	0
Insertion Sort	main	2	16	void	1	0
Insertion Sort	Insertion_srt	2	11	void	2	0
Selection Sort	main	2	9	void	1	0
Selection Sort	fillarray	0	3	void	2	1
Selection Sort	Selectionsort	3	15	void	2	1
Selection Sort	pritnarray	1	4	void	2	1
Adler32-Compress File	main	3	70	void	1	0
Deflater-Compress byte array	main	2	90	void	1	0
CRC32- Extract Zip	main	2	65	void	1	0
BinarySearchByteArray	main	3	37	void	1	0
BinarySearchCharArray	main	3	37	void	1	0

After calculating various metric values these programs are compared one by one to find potential clones. The total number of potential clones found with respect to class metrics and function metrics is shown in Figure 5.3. As an experiment, a comparison is shown between Adler32 and Deflater32 to compress files. The metric values are shown in Table 5.1 and Table 5.2.The database values are mapped to excel sheet for the comparison. The excel sheet for Adler32 is shown in Figure 5.4 and Figure 5.5 and the excel sheets for Deflater32 is shown in Figure 5.6 and Figure 5.7.

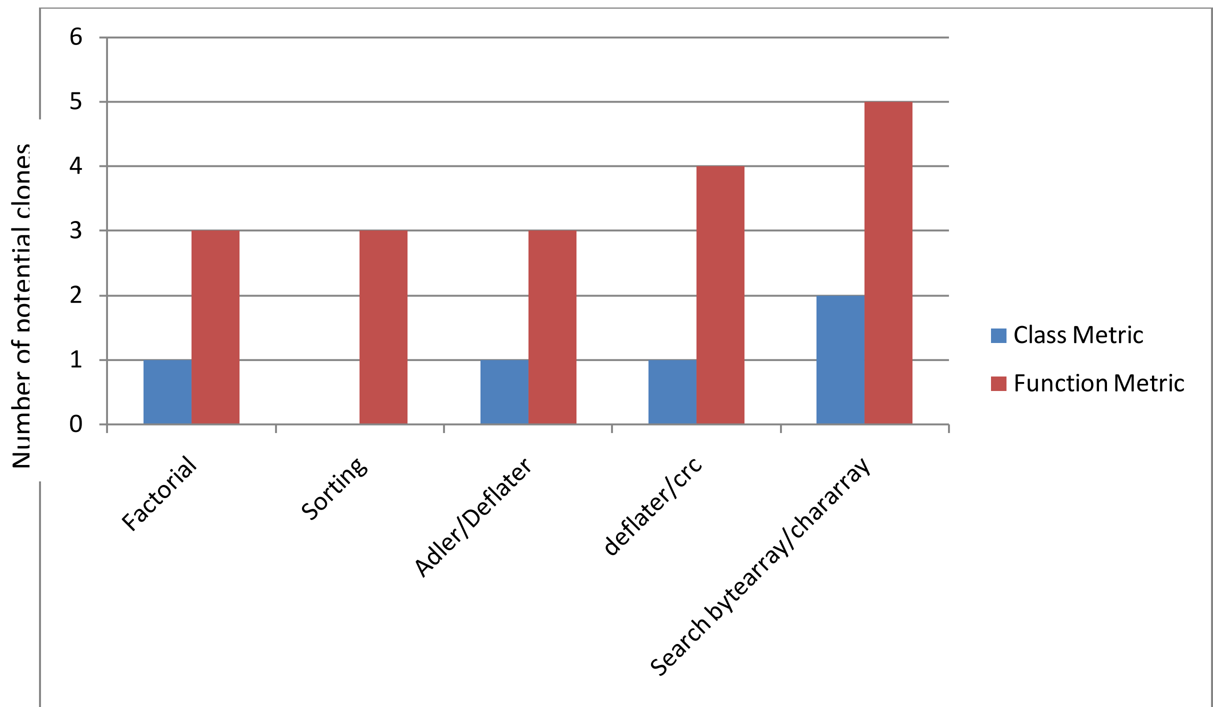


Figure 5.3: Number of Potential Clones Detected on the basis of Metric Comparison.

	A	B	C	D	E	F	G	H	I	J	K
1	ID	ClassName	totalfunc	totalif	LOC	totalV	pubV	priV	friV	proV	
2	51	CompressFileWithAdler32Che	1	0	94	0	0	0	0	0	

Figure 5.4: Class Metric of Adler32 mapped in excel sheets.

classname	functionname	localvariables	LOC	ReturnType	TotalParameters	CallToFuntion
CompressFileWithAdler32	main	3	70	void	1	0

Figure 5.5: Function Metric of Adler32 mapped in excel sheets.

ID	ClassName	totalfunc	totalif	LOC	totalV	pubV	priV	friV	proV
50	Deflater32	1	0	113	0	0	0	0	0

Figure 5.6: Class Metric of deflater32 mapped in excel sheets.

classname	functionname	localvariables	LOC	ReturnType	TotalPara	CallToFuntion
Deflater32	main	2	90	void	1	0

Figure 5.7: Function Metrics of deflater32 mapped in excel sheets.

Comparison is performed after mapping the metrics in excel database as shown in Figure 5.8 and Figure 5.9.

localhost:50461/kanikaexc ×

localhost:50461/kanikaexceldemo/Default2.aspx?a=Adlerc.xls&b=deflaterc.xls&c=44.xls&d=45.xls

first file loaded into buffer Adlerc.xls

second file loaded into buffer deflaterc.xls

Compare | LoadIntoBuffer | Clear buffer

Total_Function	Totalif	LOC	totalv	pubV	priV	proV	FrndV
1	0		0	0	0	0	0

Figure 5.8: Comparison of Class Level Metrics of Adler32 and Deflater32.

localhost:49913/exceldem × localhost:49913/exceldem ×

localhost:49913/exceldemo2/Default2.aspx?a=adlerf.xls&b=deflaterf.xls&c=68.xls&d=69

first file loaded into buffer adlerf.xls

second file loaded into buffer deflaterf.xls

Compare | LoadIntoBuffer | Clear buffer

function_name	Loval_var	LOC	return_type	TotalP	Calltofunct	proV	FrndV
main			void	1	0		.

Figure 5.9: Comparison of Function Level Metrics of Adler32 and Deflater32.

By comparing the metric values potential clones are identified and then token based approach is applied to remove false positives as metric based approach does not work directly on the code. Figure 5.10 shows the token based comparison between Adler32 and Deflater32.

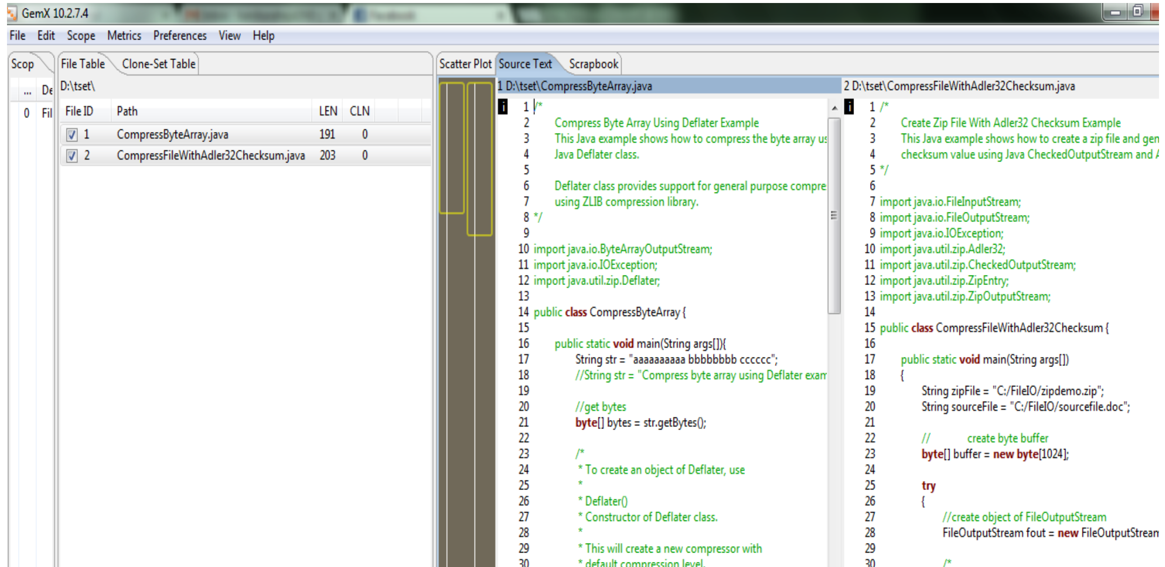


Figure 5.10: Token based Comparison using CCFinder [7]

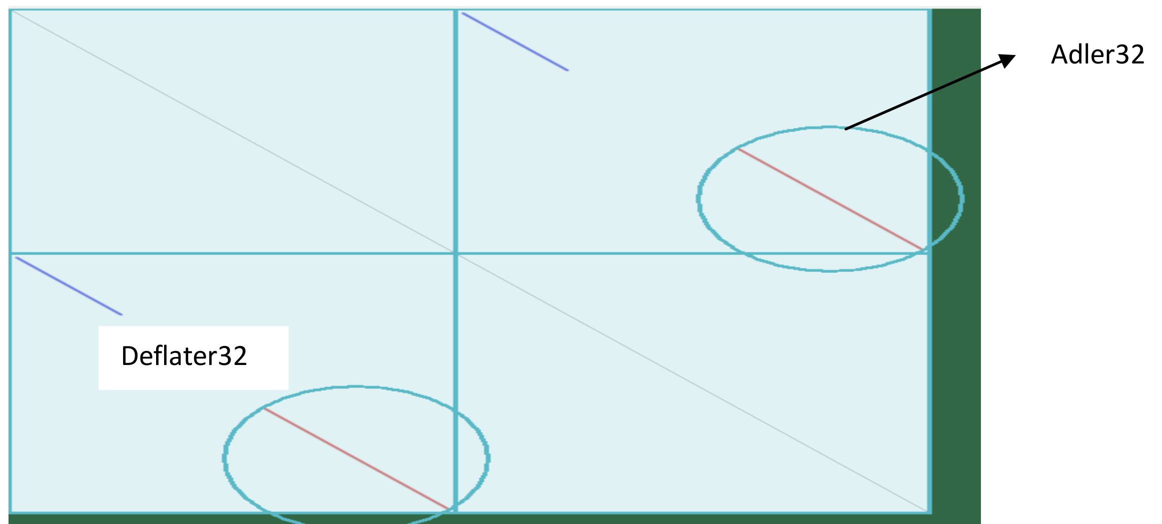


Figure 5.11 Token based comparison in terms of scatter plot using CCFinder [7]

Chapter 6

Conclusion and Future Scope

Software clone is a phenomenon in large software system. It is usually caused by programmer's 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 programmers do not fully understand the program and therefore they re-implement some existing functionality. Another reason is a 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 requirements.

Although it seems to be a simple and effective method these duplication activities are usually have weak documentation that causes a number of negative effects on the quality of the software and increases the amount of the code which needs to be maintained. 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 and clone elimination could translate into substantial savings.

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

6.1 Conclusion

- The proposed technique detects code clones on the basis of metrics based and token based approach for Java programs.
- The proposed technique looks for code clones at the class level and function level.
- The proposed technique detects potential clones on the basis of metric match. Potential clones are compared line by line using a token based approach to determine whether two potential clones detected by metric based comparison are finally the clones.
- In the proposed work Byte Code is used as an input to MCD Finder which uses a unified representation. Due to that MCD Finder is able to detect clones that are semantically equivalent to each other.
- By applying the token based approach on potential clones this tool is well suitable to find type 1 and type 2 code clones.

6.2 Future Scope

- This technique can be further extended to detect type 3 and type 4 clones.
- The output of MCD Finder can be integrated with the approaches other than token based approach.
- The proposed technique detects code clones only. It does not provide the feature to remove those clones to make the system more maintainable.
- Any redesigning approach can be applied after identifying clones with this tool.
- The proposed technique works only for the Java language program. This work can be extended to other languages of interest.

References

[1] Brenda Baker, “On Finding Duplication and Near-Duplication in Large Software Systems”, In Proceedings of the Second Working Conference on Reverse Engineering, pp 86-95,1995.

[2] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 2000.

[3] 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.

[4] Cory Kapser and 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.

[5] Chanchal Kumar Roy and James R. Cordy, “A Survey on Software Clone Detection Research”, Technical Report No. 2007-541, School of Computing Queen's University at Kingston Ontario, Canada, September 26, 2007.

[6] Zhenming Jiang and Ahmed Hassan, “A Framework for Studying Clones in Large Software Systems”, In Proceedings of the Seventh IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'07), Paris, October 2007

[7] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue,” CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code”, IEEE Transaction on Software Engineering vol. 28, NO. 7, JULY 2002.

[8] C. Liu, C. Chen, J. Han and P. S. Yu, “GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis” Conf. On Knowledge Discovery and Data Mining, pp. 872-881, 2006.

[9] Brooks, F., The mythical man-month: essays on software engineering, Reading Mass: Addison-Wesley Pub. Co., 1975.

[10] 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.

[11] Jones M., “Remix and Reuse of Source Code in Software Production”, Ph.D. Thesis, University of Illinois, Urbana, 2010.

[12] B. Baker, “Finding Clones with Dup: Analysis of an Experiment,” IEEE Transactions on Software Engineering, vol. 33, no. 9, pp. 608–621, 2007.

[13] 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.

[14] Yue JIA, “Clone Detection Using Dependence Analysis and Lexical Analysis,”PhD Thesis, King's College London, 2007

[15] Zhenmin Li, Shan Lu, Suvda Myagmar and Yuanyuan Zhou, “CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code”, Software Engineering, IEEE Transactions, vol. 32, pp. 176-192, March 2006.

[16] 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.

[17] W. Yang, “Identifying Syntactic Differences Between Two Programs,” in Software Practice and Experience, vol. 21, no. 7, pp. 739–755, July 1991.

[18] D. Rattan, R. Bhatia and M. Singh, “Software Clone Detection: A systematic review”, Information and Software Technology, Vol. 55, pp. 1165-1199, 2013.

[19] Ira Baxter, Andrew Yahin, Leonardo Moura and 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.

[20] 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

[21] 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.

[22] 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.

[23] Mark Gabel, Lingxiao Jiang and Zhendong Su, "Scalable Detection of Semantic Clones", ACM/IEEE 30th International Conference on Software Engineering, pp. 321-330, May 2008.

[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] J.-F. Patenaude, E. Merlo, M. Dagenais, and B. League, "Extending software quality assessment techniques to Java systems," in Proceedings of the 7th International Workshop on Program Comprehension (IWPC), pp. 4956, Pittsburgh, PA, USA, May 1999.

[26] 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.

[27] A.M. Leitao, "Detection of redundant code using R2D2," in Software Quality Journal, vol. 12, no. 4, pp. 361-382, 2004.

[28] 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.

[29] 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.

[30] 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.

[31] Kodhai. E, Kanmani. S, Kamatchi. A and Radhika. R, "Detection ofType-1 and Type-2 Clone Using Textual Analysis and Metrics",in ITC, 2010 IEEE.

[32] J.R. Cordy and C.K. Roy, "The NiCad Clone Detector," in 19th InternationalConference on Program Comprehension, Kingston, Canada, June 2011.

[33] Rahman F., Bird C. and Devanbu P., “Clones: What is that smell?”, Empirical Software Engineering, an International Journal 2011 Springer-Verlag.

[34] Alkhatib G., “The Maintenance Problem of Application Software: An Empirical Analysis,” Journal of Software Maintenance: Research and Practice, vol. 4, no. 2, pp. 83-104, 1992.

[35] Mondal M., Rahman M., Saha R., Roy C., Krinke J. And Schneider K., “An Empirical Study of the Impacts of Clones in Software Maintenance”, in Proceedings of IEEE International Conference on Program Comprehension, pp. 242-245, 2011.

[36] Coleman D., Ash D., Lowther B. and Oman P., “Using Metrics To Evaluate Software System Maintainability”, IEEE Computer, vol. 27, no. 8, pp. 44–49, August 1994.

[37] Manik Sharma and Gurdev Singh, “Analysis of Static and Dynamic Metrics for Productivity and Time Complexity”, International Journal of Computer Applications (0975 – 8887) Volume 30– No.1, September 2011.

[38] Hamid Abdul Basit, Simon J.Puglisi, William F. Smyth and Andrew Turpin, “Efficient Token Based Clone Detection with flexible Tokenization”, ACM SIGSOFT symposium on the foundations of software engineering , pp. 513-516, 2007.

Publications

Published

[1] Kanika Raheja and RajKumar Tekchandani, "An Emerging Approach towards Clone Detection: Metric based Approach on byte code", International Journal of Advance Research in Computer Science and Software Engineering , Vol 3, pp:881-888, May 2013.

[2] Kanika Raheja and RajKumar Tekchandani, "Detecting Clone and Redesigning", International Conference on evolution of Science and Technolgy, at PPIMT Hisar, March 2013.

Communicated

[1] Kanika Raheja and RajKumar Tekchandani, " An Efficient Code Clone Detection Model Using Hybrid Approach on Java ByteCode", IEEE Conference at Amity University, Sept 2013.

Factorial Program using for loop

```
package b;
import java.io.*;
class Factorial{
 public static void main(String[] args) {
 try{
 BufferedReader object = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("enter the number");
 int a= Integer.parseInt(object.readLine());
 int fact= 1;
 System.out.println("Factorial of " +a+ ":");
 for (int i= 1; i<=a; i++){
 fact=fact*i;
 }
 System.out.println(fact);
 }
 catch (Exception e){}
 }
}
```

Factorial Program using while loop

```
import java.io.*;
class Factorial1{
 public static void main(String[] args) {
 try{
 BufferedReader object = new BufferedReader(new InputStreamReader(System.in));
 System.out.println("enter the number");
```

```
  int a= Integer.parseInt(object.readLine());
  int fact= 1;
  System.out.println("Factorial of " +a+ ":");
int i=1;
while(i<=a)
  {
  fact=fact*i;
i++;
  }
  System.out.println(fact);
  }
  catch (Exception e){}
  }
}
```

Insertion Sort

```
public class InsertionSort{
  public static void main(String a[]){
  int i;
  int array[] = {12,9,4,99,120,1,3,10};
  System.out.println("\n\n RoseIndia\n\n");
  System.out.println(" Selection Sort\n\n");
  System.out.println("Values Before the sort:\n");
  for(i = 0; i < array.length; i++)
  System.out.print( array[i]+"  ");
  System.out.println();
  insertion_srt(array, array.length);
  System.out.print("Values after the sort:\n");
  for(i = 0; i <array.length; i++)
  System.out.print(array[i]+"  ");
  System.out.println();
```

```
 System.out.println("PAUSE");
 }
 public static void insertion_srt(int array[], int n){
 for (int i = 1; i < n; i++){
 int j = i;
 int B = array[i];
 while ((j > 0) && (array[j-1] > B)){
 array[j] = array[j-1];
 j--;
 }
 array[j] = B;
 }
 }
}
```

Selection sort

```
import java.util.*;
 class SelectionSort
 {
     static Scanner in = new Scanner(System.in);
     public static void main(String[]args)
         {
         System.out.println("Enter your Array Size");
         int n = in.nextInt();
         int[]arr = new int[n];
         fillArray(arr,n);
         sellectionSort(arr,n);// callint selection sort method
         printArray(arr,n);
         }
     static void sellectionSort(int[]a,int n)
```

```
        {
        for(int out=0;out<n-1;out++)
            {
            int min=out;
            for(int in=out+1;in<n;in++)
                {
                if(a[min]>a[in]) // sending max element to last.
                    {
                    min=in;
                    }
                }
                int temp = a[out];
                a[out]=a[min];
                a[min]=temp;
            }
        }
    static void fillArray(int []a, int n)// fill array Method
        {
            System.out.println("Enter Your Elements");
            for(int i=0;i<n;i++)
                a[i]=in.nextInt();
        }
    static void printArray(int []a, int n)// print array Method.
        {
            System.out.print("Array Elements = ");
            for(int i=0;i<n;i++)
                System.out.print(a[i]+"\t");
            System.out.println(" ");
        }
}
```

Adler-32 Compress File

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.zip.Adler32;
import java.util.zip.CheckedOutputStream;
import java.util.zip.ZipEntry;
import java.util.zip.ZipOutputStream;
public class CompressFileWithAdler32Checksum {
	public static void main(String args[])
	{
		String zipFile = "C:/FileIO/zipdemo.zip";
		String sourceFile = "C:/FileIO/sourcefile.doc";
				byte[] buffer = new byte[1024]
		 try
		{			FileOutputStream fout = new FileOutputStream(zipFile);
			CheckedOutputStream checksum = new CheckedOutputStream(fout,
new Adler32());
		 ZipOutputStream zout = new ZipOutputStream(checksum);
		 FileInputStream fin = new FileInputStream(sourceFile);
		   zout.putNextEntry(new ZipEntry(sourceFile));
		  int length;
		   while((length = fin.read(buffer)) > 0)
		     {
		       zout.write(buffer, 0, length);
		     }
		     zout.closeEntry();
		        fin.close();
		          zout.close();
		     System.out.println("Zip file has been created!");
```

```
 System.out.println("Adler32 Checksum is : " + checksum.getChecksum().getValue());
        }
        catch(IOException ioe)
        {
            System.out.println("IOException : " + ioe);
        }}
```

Delater-32 Compress byte array

```
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.util.zip.Deflater;
public class CompressByteArray {
    public static void main(String args[]){
        String str = "aaaaaaaaaa bbbbbbbb cccccc";
        byte[] bytes = str.getBytes();
         Deflater deflater = new Deflater();
          deflater.setInput(bytes);
            deflater.finish();
      ByteArrayOutputStream bos = new ByteArrayOutputStream(bytes.length);
        byte[] buffer = new byte[1024];
          while(!deflater.finished())
          {          int bytesCompressed = deflater.deflate(buffer);
                     bos.write(buffer,0,bytesCompressed);
       }
          try
       {
         bos.close();
          }
          catch(IOException ioe)
          {
                  System.out.println("Error while closing the stream : " + ioe);
```

```
}
byte[] compressedArray = bos.toByteArray();
System.out.println("Byte array has been compressed!");
System.out.println("Size of original array is:" + bytes.length);
System.out.println("Size of compressed array is:" + compressedArray.length);
}}
```

CRC-32 Extract Zip File

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;
import java.util.zip.CRC32;
import java.util.zip.CheckedInputStream;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
public class ExtractZipWithCRC32Checksum {
    public static void main(String args[])
    {
     String sourceZipFile = "C:/FileIO/sourceFile.zip
        try
        {               FileInputStream fin = new FileInputStream(sourceZipFile);
}
    CheckedInputStream checksum = new CheckedInputStream(fin,new CRC32());
            ZipInputStream zin = new ZipInputStream(checksum);
            ZipEntry entry = zin.getNextEntry();
            OutputStream os = new FileOutputStream("c:/extractedFile.css");
             byte[] buffer = new byte[1024];
            int length;
             while( (length = zin.read(buffer)) > 0)
            {
```

```
                    os.write(buffer, 0, length);
                }
                 os.close();
                  zin.close();
               System.out.println("File Extracted from zip file");

   System.out.println("CRC32 checksum is: " + checksum.getChecksum().getValue());
                      }
          catch(IOException e)
          {
                System.out.println("IOException :" + e);
          }}}
```

Binary Search Byte Array

```
import java.util.Arrays;
public class BinarySearchByteArrayExample {
 public static void main(String[] args) {
   byte bArray[] = {1,2,4,5};
   Arrays.sort(bArray);
     byte searchValue = 2;
   int intResult = Arrays.binarySearch(bArray,searchValue);
   System.out.println("Result of binary search of 2 is : " + intResult);
    searchValue = 3;
   intResult = Arrays.binarySearch(bArray,searchValue);
   System.out.println("Result of binary search of 3 is : " + intResult);
 }}
```

Binary Search Char Array

```
import java.util.Arrays;
```

```
public class BinarySearchCharArrayExample {
 public static void main(String[] args) {
   char charArray[] = {'a','b','d','e'};
    Arrays.sort(charArray);
     char searchValue = 'b';
     int intResult = Arrays.binarySearch(charArray,searchValue);
   System.out.println("Result of binary search of 'b' is : " + intResult);
     searchValue = 'c';
   intResult = Arrays.binarySearch(charArray,searchValue);
   System.out.println("Result of binary search of 'c' is : " + intResult);
 }}
```