Code Clone Detection using Abstract Syntax Tree Technique

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

Master of Engineering

in

Computer Science and Engineering

Submitted By

Kanika Bhardwaj

(Roll No. 801632019)

Under the supervision of:

Mr. Rajkumar Tekchandani

Assistant Professor

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING AND TECHNOLOGY
PATIALA – 147004

June 2018

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"*Code Clone Detection using Abstract Syntax Tree Technique*"**, in partial fulfillment of the requirements for the award of degree **of Master of Engineering** in ***Computer Science and Engineering*** submitted in Computer Science and Engineering Department of **Thapar Institute of Engineering and Technology, Patiala,** is an authentic record of my own work carried out under the supervision of ***Mr. Rajkumar Tekchandani*** and refers other researcher's work which are duly listed in the reference section.

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

Signature:

Kanika Bhardwaj

801632019

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

Signature:

Rajkumar Tekchandani

Assistant Professor,

Computer Science and Engineering Department,

Thapar Institute of Engineering and Technology, Patiala.

Acknowledgement

No volume of words is enough to express my gratitude towards my guide, **Mr. Rajkumar Tekchandani**. I thank my supervisor for his time, patience, discussions and valuable comments. He has been very concerned and have aided for all the material essential for the preparation of this thesis report. His enthusiasm and optimism made this experience both rewarding and enjoyable.

I am equally grateful to **Dr. Maninder Singh**, Head of Computer Science and Engineering Department and **Mr. Ashutosh Mishra**, P.G Coordinator, for motivation and inspiration that triggered me for the thesis work.

I also want to express my gratitude to **Dr. S S Bhatia**, Dean of Academic Affairs, 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 help, cooperation and affection, which made my stay at Thapar Institute of Engineering and Technology memorable.

Last but not the least, I would like to thank my parents and Almighty for showing me the right direction, without their blessings none of this would have been possible.

Abstract

Software systems are getting more complex day by day and to manage such systems is an essential topic for the software industry today. Cloning is one of the major reasons which is making maintenance of software more troublesome. Code Cloning is a mechanism of replicating code fragments by copy-paste. In the early stages of the development of project, developers find it easy to copy and simply paste the code.

Although various modern programming languages offers different deliberation systems to encourage reuse of the code fragments, is as yet a broadly utilized reuse methodology. This frequently prompts various duplicated code fragments which are known as “Clones” in the large software systems. However, code cloning is extremely hazardous for programming support as it superfluously increases the program size. Since the vast majority of the support endeavors coexist with the program size, this additionally expands the maintenance effort because changes to one clone, for example, bug settling commonly should be made to alternate clones too, again expanding the maintenance exertion.

To be more precise, the thought behind this idea of cloning is to make another software product that duplicates the perspective and convenience of the original software. It is critical issue to comprehend that cloning need not need to include any source code in the original software. The objective in cloning is to make another software program that behaves exactly same as the original software and the way in which it behaves.

Here on the implemented approach, the clone detection method is implemented using Abstract Syntax trees (ASTs). The approach is based on identifying the code clones on the basis of common sub expression. The location of code clones and their classification is carried out. Categorization is done on the basis of levenshtein distance. The approach involves abstract syntax tree based clone detection for python scripts and later comparison with the tool.

Table of Contents

List of Figures

List of Abbreviations

AST	Abstract Syntax Tree
PDG	Program Dependence Graph
CP	Clone Pair
CC	Clone Class
CSV	Comma Separated Values

Chapter 1

Introduction

Code duplication or replicating a code fragment and after that reusing it with or with no alterations is an outstanding method in software maintenance. A few investigations demonstrate that around 5% to 20% of a software system contains copied code, which are essentially the consequences of recreating the already existing code parts and utilizing with or without minor alterations. One of the noteworthy drawbacks of the copied code segments is that if a bug is identified in a specific code fragment, then various sections like this are found to check the possible presence of a similar bug in the comparable pieces. Refactoring of the repeated code is another important issue in programming support. Although various surveys asserts that the refactoring of specific clones are not wanted and there is a danger of evacuating them. In any case, it is additionally generally concurred that clones ought to in any event be distinguished.

Code clones are used most of the time since it can be generated quickly and effectively embedded with minimum cost. Code clones may influence software maintenance but it may also present poor design, increment the system size, lead to the wasting of time to repeatedly understand a fragment of a poorly written code, and resurrect errors which are available in the already existing code fragment. These make it a troublesome activity to handle a huge software system.

1.1 Code Cloning

Code cloning can be characterized as a demonstration of reusing a code fragment by duplicating it from one segment of the software and after that using it with or without some slight alterations into different sections of the software. It is an essential method for software reuse. The issue with such replicated code fragment is that a bug recognized in the original code must be checked in each clone for a similar mistake.

Reusing software by methods for reorder is a continuous action in programming advancement. The copied code is called as *Software Clone* and the procedure is called as Code Clone. An example of code clone is shown in Figure 1.1.

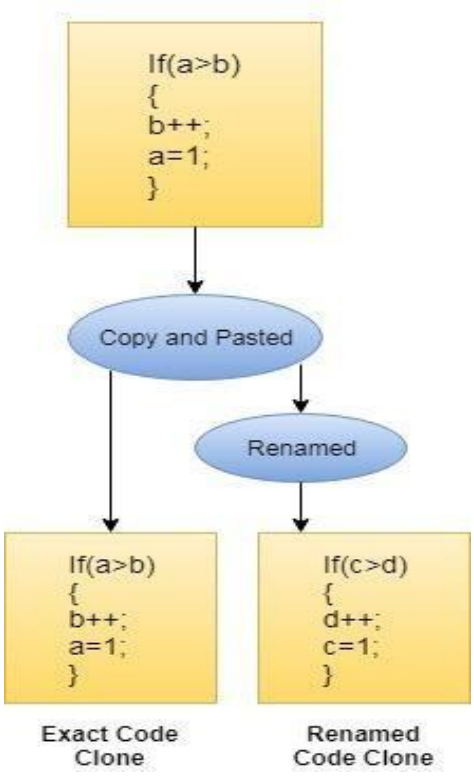


Figure 1.1: Example of Code with Clones.

In any case, the code quality investigation (enhanced quality code), replication identification, virus recognition, bug exposure and facet mining are the some other programming designing undertaking which needs the mining of the syntactically or semantically similar code fragments to make the detection of clone significant for software analysis.

1.2 Basic Terminology

As the name suggests code clones are the duplicated regions of code. Be that as it may, not at all like biological clone, a software clone could possibly be precisely the same [11]. For a given clone relation, if the clone connection holds between the segments a couple of code portions is called as clone pair. The equivalent class of clone relation is called clone class [6].

Techniques and tools for detecting duplicate code are of main concern in software maintenance research. Some of the definitions related to code-cloning are discussed below:

A. **Code Fragment:** It is defined as some sequence of code lines having different types of similarity between various code fragments in its source code. These similar code fragments may have comments or without comments. For example: sequence of

statements, begin-end block, etc.

B. **Clone Set:** It is defined as a set of all the identical or similar code fragments.

C. **Clone Pair:** If the two code fragments are analyzed and there exists a clone relation between the pair of the code fragments then at that point it is known as a Clone Pair.

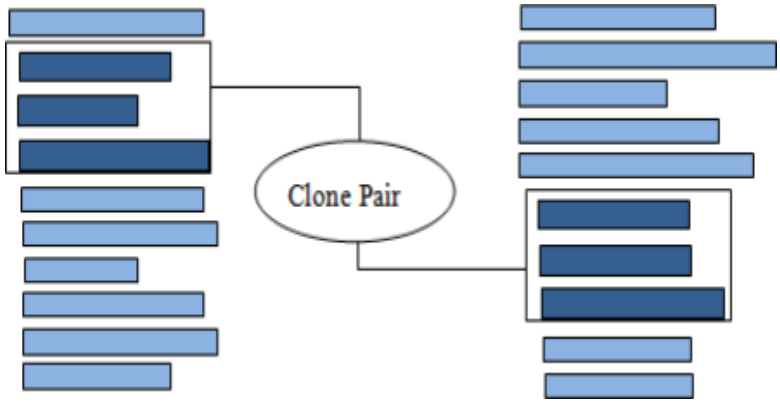


Figure 1.2: Clone pair

D. **Clone Class:** It is defined as a set of all the clone pairs in which the existing clone pairs having some clone relationship between them is known as clone class.

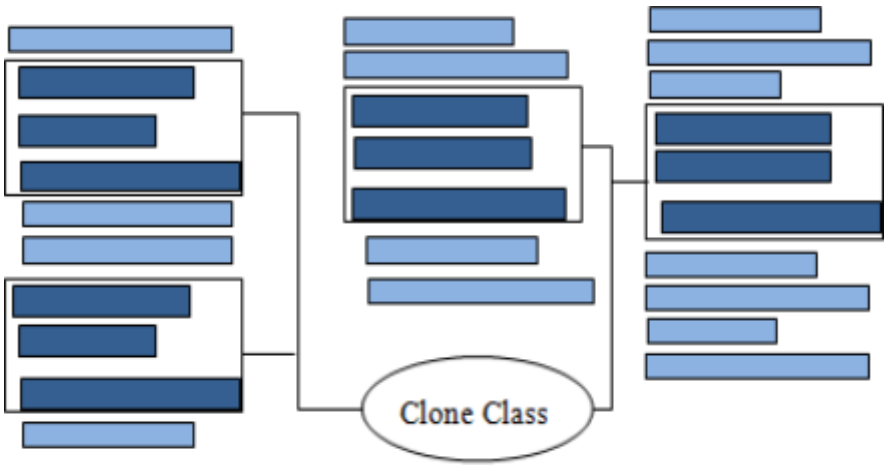


Figure 1.3: Clone Class

1.3 Code Clone Types

Code Clone classification is utilized for development of reengineering and detection methods. Figure1.4 describes the classification of code clones. These can be categorized depending on the three factors which are given below. Code clones are ordered based on following three angles such as:

- Similarity among the code segments,
- Refactoring opportunities with the repeated code, and
- Clone instance position in a program

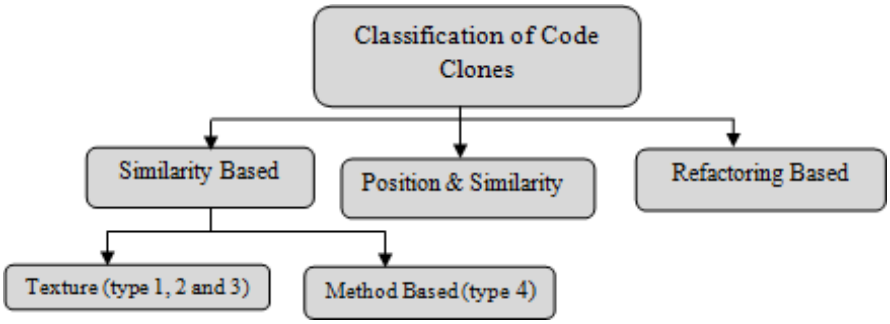


Figure 1.4: Taxonomy of the Code Clones

The Similarity based clones are of Code Clones are basically of two kinds as follows:

- The two code segments can be indistinguishable based upon the likeness of their content and program
- The two code segments can be identical in their usefulness without being textually identical.

However, the textual similarity based clones are predominantly of three following types:

1.3.1 Type 1(Exact Clones):

These clones are the similar code segments without modification except for the change in comments and whitespaces.

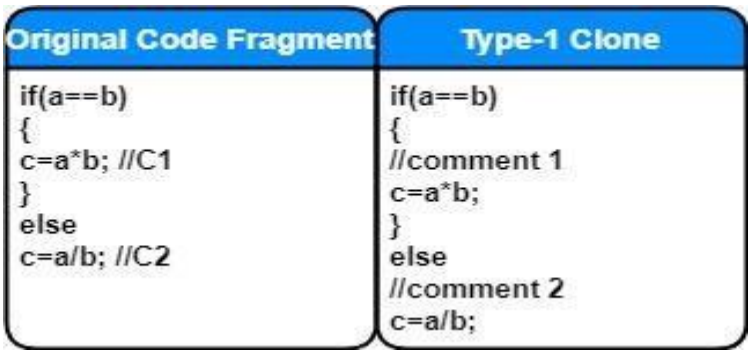


Figure 1.5: Example of Exact Clones.

1.3.2 Type 2 (Renamed/Parameterized): The clones which are syntactically similar except for the changes in names of function types, variables and identifiers.

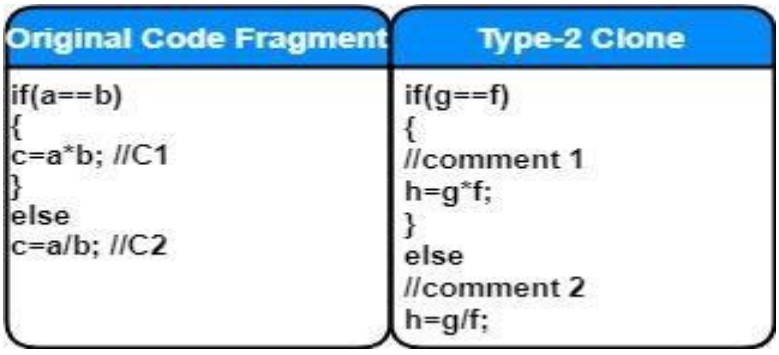


Figure 1.6: Example of Renamed/ Parameterized Clones.

1.3.3 Type 3 (Near miss clones/Gapped clones): These clones are the similar code fragments with or without any other modifications expect if only some statements are added, removed or changed.

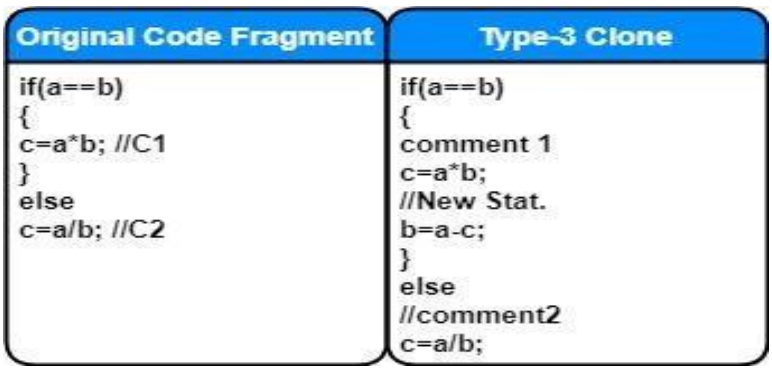


Figure 1.7: Example of Near Miss Clone.

The syntactic sections are to be assessed in a specific order which is balanced by the developers after replication. For example, the functions which are same except for name which are indistinguishable for the kinds of parameters coordinated in high-similarity code clones.

1.3.4 Type 4 (Semantic clones): If two code segments perform the same functionality but they are having different syntax, then they are said to be type-4 or semantic clones.

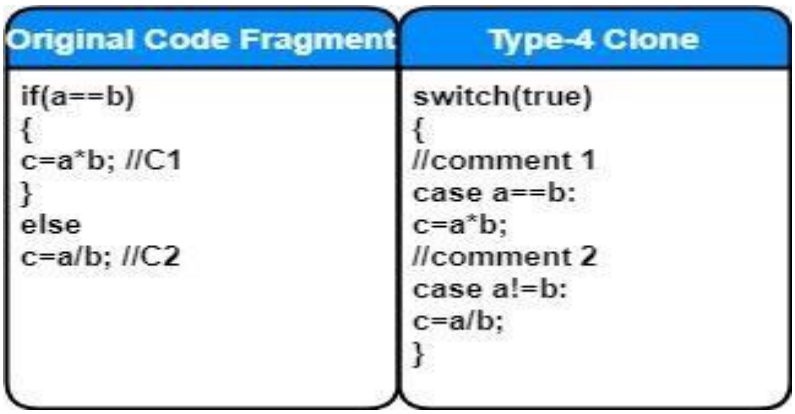


Figure 1.8: Example of Semantic Clones

1.4 Code Clone Detection Process

A clone indicator must attempt to discover bits of code of high similitude in a framework's source content. The principle issue is that it isn't known already which code parts might be repeated. Along these lines the locator should contrast each possible fragment and each other possible fragment. A clone detection process includes following steps:

1.4.1 Preprocessing

Towards the start of any clone identification approach, the original code is separated and the basis of the comparison is determined. This method ousts uninteresting bits of code, changes over source/original code into a fewer units and determines the comparison units. The three fundamental assignments of this stage are given beneath.

- Expel the unwanted bits of the code. Each segment in the source code which is not critical for the examination design are cleared or sifted through in this stage.
- Recognize the units of source code. The remaining of the source code is separated into additionally source pieces, which will then used to check for the presence of direct clone relations with each other.
- Recognize the comparison units. The source units can be separated into smaller units depending on the comparison algorithm.

1.4.2 Transformation

This stage is utilized by every other methodology with the exception of textual based approaches for the detection of clones. This stage changes the source code into a particular intermediate portrayal for correlation.

There are different kinds of portrayals relying upon the procedure. The various steps involved in this phase are given below.

- Extraction of Tokens**:** Token extraction or Tokenization is performed amid the lexical analysis by compiler front end in the programing languages. Single line of source code is changed over into a sequence of tokens.

- Abstract Syntax Tree Extraction: All of the source code is parsed to change over into a unique abstract syntax tree for sub-tree comparisons.

- Extraction of PDG: A Program Dependence Graph (PDG) represents the data dependencies and control dependencies. The nodes of a PDG represent the conditions and statements in a program. A data dependency represents the data flow of information in a program whereas a control dependency represents the flow of control information within the program.

1.4.3 Match Detection

Output of the transformation phase is the input to the match detection phase. Each changed code fragment is contrasted with every single other section utilizing a comparison algorithm to

discover similar code fragments. The yield of this step is a set of a combination of clone pairs in single class or in a single group.

1.4.4 Formatting

In this step, it changes over a clone pair list got by the comparison algorithm from the Match Detection step into another clone pair list identified with the original source code.

1.4.5 Filtering

It is not necessary for every clone detectors go through this step. This phase includes the extraction of clones and the filtering out of the false-positive clones. This phase is also as known Manual Analysis phase because a human expert is required to filter out the false-positive clones.

1.4.6 Aggregation

As the vast majority of the tools straightforwardly distinguish clone classes, a large portion of them match clone pair accordingly. This progression can be performed to decrease the amount of data, perform consequent analysis or to aggregate the clones into clone classes.

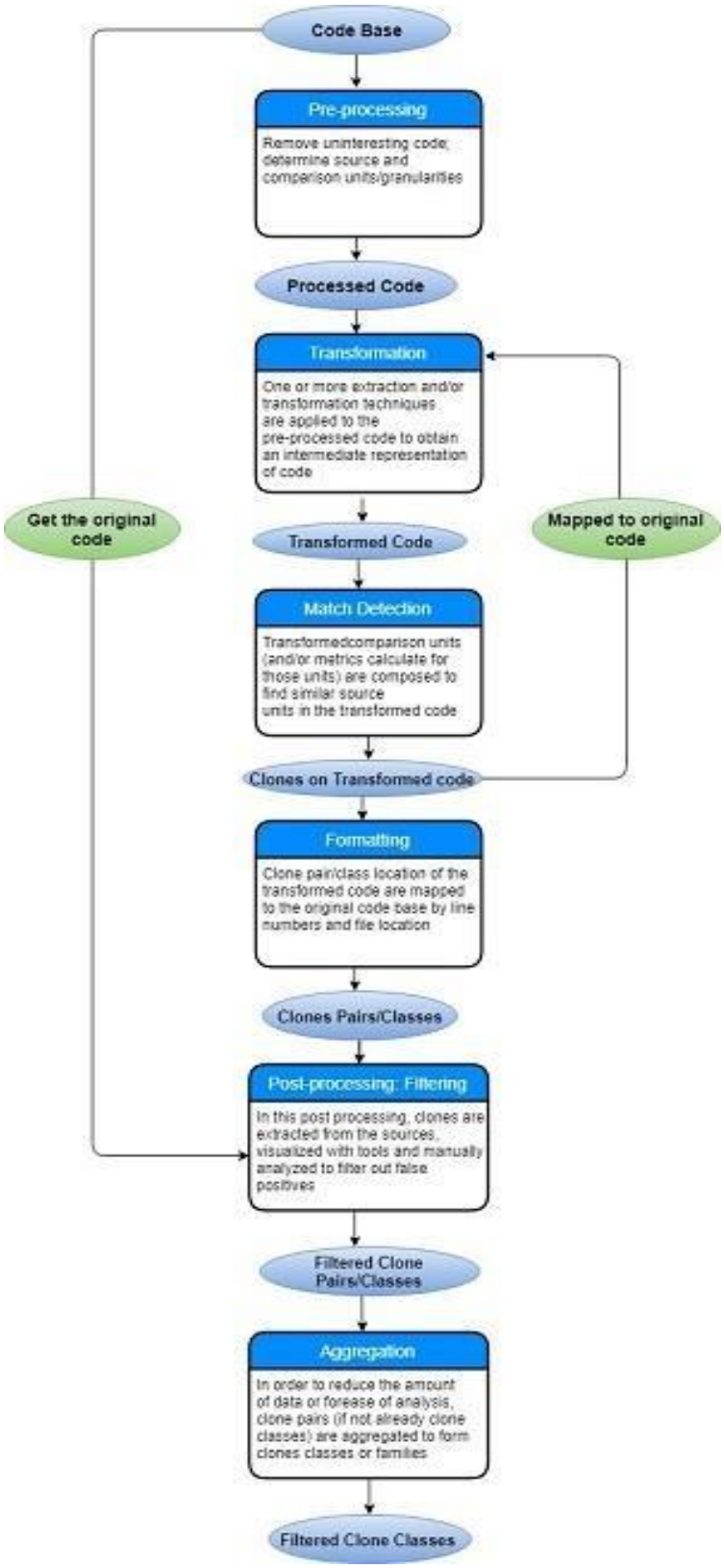


Figure 1.9: Code Clone detection process

1.5 Reasons for Code Cloning:

There are number of reasons why developers clone the source code. Cloning for the most part happens in light of the fact that for developers it is less demanding to reorder the code as opposed to composing it starting with no outside help. Developers may sometimes duplicate the code because of time constraints, these constraints may be forced by deadlines.

Programming clone has various negative impacts on the nature of the product. Other than expanding the measure of the code which should be kept up, it additionally builds the bug likelihood.

Some of the reasons of code cloning are mentioned as follows:

- **Time Limit**

  It is the major cause of code cloning as the certain time confine is relegated to developer to finish a project. So developer basically reorders the current one and as indicated by their present need.

- **Trouble in Understanding Large System**

  By and large, it is hard to comprehend an extensive programming framework. Along these lines, it powers the programmers to utilize the illustration arranged programming by adjusting already existing code.

- **Hazard in the new code**

  There is dependably a danger of blunders in new code pieces and in light of the fact that officially introduce code is tried and there is bring down danger of mistake. In this way, it is encouraged to developers to reuse the officially existing code by replicating and changing the current code as per the new item's necessity.

- **Performance of a Developer**

  Here and there the execution of a developer is estimated by the quantity of lines delivered by him. In such conditions the developer's concentration is to increase the quantity of lines, so the developer attempts to reuse a similar code.

- **Language Limitation**

  Sometimes, the developers copy because of the limitation of developer’s knowledge in the particular language.

1.6 Drawbacks of code duplication

Code Clones has serious effect on the viability, reusability and nature of a product framework. The accompanying are the disadvantages of having cloned code in a framework.

- **Higher likelihood of bug propagation:** On the off chance that a code section contains a specific bug and that portion is reused by replicating and the reused with or without changes, the error in the first portion will be available in all the glued fragments in the product framework and therefore, the likelihood of bug propagation will increment essentially in the system [32].
- **Higher likelihood of presenting another bug:** n the majority of the cases, just the structure of the copied piece is reused by the designer to satisfy the present need. This procedure can be erroneous and may present the new bugs in the framework [33].
- **Large likelihood of bad design:** Code Cloning can likewise present absence of good inheritance structure or deliberation and terrible design. Along these lines, it turns out to be exceptionally hard to reuse this part in future undertakings.
- **Difficulty in framework change/alteration:** Due to the recreated code in the framework, extra time and consideration is expected to comprehend the current cloned execution and approaches to be adjusted, and in this way, it turns out to be extremely hard to include new functionalities in the framework.
- **Higher upkeep cost:** On the off chance that a cloned fragment is observed to be contained a mistake, the majority of its counterparts should be explored for redressing the bug as there is no assurance that the bug has been as of now been expelled from other comparative parts at the season of reusing or amid upkeep. [27]

1.7 Advantages of Code Clone Detection

- **Supports in understanding of program:** The usefulness of a cloned piece is comprehended, to get a general thought of different documents containing comparative duplicates of the section
- **Helps perspective mining research:** Recognizing code clone is additionally essential in perspective mining to identify the cross-cutting concerns. The code of cross-cutting

concerns is commonly copied over the whole application that could be related to clone discovery devices [39, 40].

- **Detection of malicious software:** In detection of a pernicious programming code clone identification systems assumes an indispensable part. By examination of one pernicious programming to another it is conceivable to discover the regions where parts of the one programming framework coordinate parts of another. [36].
- **Detection of plagiarism and copyright encroachment:** Finding comparable code is additionally helpful in distinguishing the plagiarism and copyright encroachment [36, 37, and 38].

1.8 Motivation

Code cloning is hazardous for the support period of the product on the grounds that if the cloned code comprises of a mistake then we have to identify and amend a similar blunder from the clones of that code which shows that it is beneficial as if the tested or bug free code is inserted in other places then it is easier to maintain the source code. So that is the reason cloning is important. Recognition of code clones is beneficial much of the time, for example, code understanding, compacting programming size, and distinguishing pernicious programming and use designs. For recognition of cloned code, various strategies have been proposed so far that are quite efficient in detecting clones. But some of these techniques are very complex and some techniques take a huge amount of time. So, there is a need for a simple, time efficient and space efficient clone detection technique.

1.9 Thesis Outline

This thesis work is organized into six chapters. Each chapter is written to be largely self-contained and complete. The summary of the rest of the chapters are given below.

Chapter 2 provides the overview of the various existing techniques and compares Abstract Syntax Tree based code clone detection techniques, their advantages, disadvantages and scope. This chapter provides a way to find the research gap and to propose an idea for thesis work which is discussed in the Chapter 3.

Chapter 3 defines the problem statement and the research objectives of this thesis. On the basis

of the objectives, the problem and complete methodology is formulated in various subparts which is discussed in the chapter 4.

Chapter 4 explains the proposed approach and its execution. A complete methodology of the code clone detection is discussed.

Chapter 5 gives the experimental results of executing the proposed approach on a software system. This chapter includes the obtained experimental outcomes and related discussions.

Chapter 6 concludes the thesis and gives bearings for future work

Chapter 2

Literature Survey

This chapter gives a review of various existing code cloning detection techniques. To detect the cloned code, numerous methodologies have been proposed which are explained in the below sections.

Clone detection attempts to find out the duplicate code within whole software, which may be exactly-copied or modified somewhere. Several techniques are available to detect duplicate code.

2.1 Text Based Code Clone Detection Technique

In this approach, the whole source code is accepted as the arrangement of strings. A solitary line of code is compared with another line by applying string matching algorithms, and identical strings are reported as code clones. The raw source code is used for detection because this method is purely textual there is no other transformation to source code can be performed. Raw source code may contain white spaces and comments. However, it may be needed some time to remove white spaces and comments etc.

Ducasse et al. [6] proposed an approach in which, source code is changed into internal format by the removal of comments and white spaces then further comparisons are done.

S. Lee et al. [24] developed SDD (Similar Data Detection) algorithm that finds exact clones. SDD has controlled complexity using Inverted Index and an index. Authors revealed that SDD shows better results than PMD, which is a source code analyzer that discovers basic programming imperfections like unnecessary object creation, empty catch blocks, unused variables, and so forth. SDD also detects modified clones by using N-neighbor distance concept. Moreover, SDD is language independent.

J.R. cordy et al.[25] discussed light weight text based approach to detect near-miss clones. Basically they applied Pretty printing and Code normalization technique to find code clones. Code lines are broken into parts and clones are extracted by comparing the broken text and by applying code normalization. Basically UPI (Unique percentage of Items) is calculated and on the basis of those unique line gaps are detected. Whole technique is implemented in a tool NICAD, which is parser based and language specific but reasonably light weight using simple

line matching. Several case studies are carried out that Abyss [2] of 1500 lines and Weltab [3] of 11000 lines. These two are taken as test beds because results are already published for these.

2.2 Token Based Code Clone Detection Technique

Kamiya et al. [8] described the process of token-based technique. This procedure comprises of four stages:

Lexical Analysis: Every line of source files is separated into tokens according to lexical rules of respective programming language. The generated tokens from all the source files are concatenated to form one single sequence of tokens. It will be easy then to perform analysis of this single token sequence. White spaces, comments and tabs are removed from source code during preprocessing.

1. **Transformation:** Identifiers are then replaced with customized tokens by using transformation rules. And this replaced information is kept at back up for future formatting into original text.
2. **Match Detection:** At that point on changed token arrangement, the sequence of lines is then compared efficiently using similarity detection algorithm (token suffix tree). Then the similar lines of sequences are reported as clone pairs.
3. **Formatting:** Every area of clone pair is changed into line numbers on the first source records.

A clone detection tool Dup [21] used a sequence of lines for the representation of source code and it detects clones line-by-line. It replaces identifiers present in the source code with common identifier name as id1,id2….idn and returns code clones using suffix tree algorithm. The line-by-line method has a limitation in the line-structure modification.

Ref: [23] Token-suffix trees scales very well in time and space, because of its linear complexity. Studies [9, 11] have shown that token based clone detection approaches suffer from many false positives, but this technique has high recall value with low precision.

Ueda et al. [20] developed Gemini in which maintenance support environment is used for visualization of clones on the output of CCFinder. Gemini specify GUI (scatter plot and metrics graph about code clones). This is basically used for the visualization of detected code clones. The scatter plot graphically demonstrates the areas of code clones among source codes. The measurements diagram indicates metric estimation of every clone. Utilizing Gemini, we can indicate the code clones that ought to be paid heed in the maintenance stage.

2.3 Abstract Syntax Tree (AST) Based Code Clone Detection Technique

In this approach, Abstract Syntax Tree (AST) of a program is produced utilizing a parser of a dialect. Then tree matching technique is applied on that AST generated to identify identical sub trees. When a match is found between two sub trees, then source code of identical sub trees are returned as clone-pairs. Baxter *et al.,* [5] uses a hash function to partition sub trees of the abstract syntax tree of a program. Then sub trees in the similar partition are compared using tree-matching technique. A comparable strategy was additionally proposed by Yang [2] utilizing dynamic programming to distinguish contrasts among different adaptations of same files. Jiang *et al.,* [23] presented an approach that has been implemented through tool *Deckard.* This tool is platform independent. Characteristic vectors of AST are calculated in a Euclidean space and then those vectors are merged to compute similarity among sub trees. LSH (Local Sensitive Hashing) has been used to cluster similar vectors that can hash two similar vectors to the same hash value with arbitrary high probability and two distant vectors with arbitrary low probability and hence find clones.

2.4 Metric Based Code Clone Detection Technique

In this approach different software metrics are gathered and on the basis of similar values of these metrics, clones are detected. At first an arrangement of programming measurements are assigned to syntactic units such as classes and functions .Then estimations of these measurements were thought about. If the two syntactic units are having the same metric value, then these can be regarded as a clone pair.

Mayrand et al. [4] used various metrics to detect clones. Functions with similar metric values are returned as clone-pairs. Metrics are calculated from expressions, control flow of functions, layouts and names.

2.5 Program Dependence Graph (PDG) Based Code Clone Detection Technique

Program Dependence Graph (PDG) is used to show control flow and data flow dependencies of a program. The isomorphic sub graphs in a program dependence graph are named as clone-pairs. PDG based technique can detect Type-3, 4 clones, because semantic information is carried out in PDG. Lieu et al. [10] has implemented a plagiarism detection algorithm and Gplag tool is implemented, which is based on PDG approach.

Related Comparitive Studies:

Rysselberghe *et al.,* [13] compared three techniques: parameterized matching, simple line based matching and metric fingerprints. Research process used during experiment is based on Goal-Question-Metric worldview like what sorts of matches are found? How accurate are the results and how useful information is gained? etc. The conclusions has been drawn that simple line matching is purely language independent on the other hand all other techniques need some kind of configuration. Function block duplication is found by metric fingerprint technique and general duplication is found by other techniques. No false matches are found by basic line matching, few false matches are found by parameterized technique and even more false matches are found by metric fingerprint (characterization of expressions which lacks accuracy is responsible for this problem). False matches for metric fingerprint thus depends on the way expressions are characterized and the length of the code fragments under consideration, While number of recognizable matches are high for this technique.

F. Zibran *et al.*, [26] discussed a focused approach of a selected code segment which is known as seed segment rather than the location of all clones from the whole code-base rather than the location of all clones from the whole code-base. The limitations of available techniques are to find out type-1 and 2 clones and mainly implemented as stand-alone tools which uncovers the area of clone-aware development. Seed fragment is compared with the search space (whole source code) to find out type-3 clones. Mainly fingerprinting technique is used to generate finger prints for the unique lines and then syntax tree or suffix tree is generated for the whole fingerprint sequence. Suffix tree is generated for the generalized fingerprint sequence using Ukkonen's online algorithm. Eclipse's JDT API's are used to generate the ASTs (Abstract Syntax Trees). Fabio Calefato *et al.*, [27] described a semi automated approach to find clones in scripting code of web applications. The approach is useful to select function clones and to inspect selected script functions. The Semi-automated approach is both effective and efficient for identification of function clones in web applications. Muhammad Asaduzzaman [28] addressed that in spite of

number of clone detection tools are available, even then there is one challenge of handling raw clone data, because of textual nature and large in volume. To address this issue, a framework VisCad is proposed for performing large scale code clone analysis. It also acts as a maintenance support environment. In VisCad various visualization techniques, number of metrics and data filtering options are available, therefore users can analyze and identify distinctive code clones.

Chapter 3

Problem Statement

Various surveys show that a large portion of source code in some substantial scale applications contains code clones. The presence of code clones can present much instability within a software application, unnecessary duplicates may create ambiguity. These insecurities can confound routine maintenance tasks, since an adjustment in one strategy may prompt changes crosswise over numerous techniques. Moreover, necessary duplicates can possibly instigate the spread of bugs and keep changes from being engendered. Subsequently, keeping in mind the end goal to anticipate and treat these insecurities, we must figure out where potential clones occur. There are different techniques for detecting code clones which are discussed in the literature survey. Each technique has its own benefits and limitations.

As the Text based techniques provides the simplest way of detecting the clones but they can only identify Type 1 clones. Though the Token based techniques can identify both type-1 and type-2 clones but these techniques need a lexical analyzer to transform the code into tokens. A significant amount of time is consumed in tokenization. Metric- based code clone detection techniques need a PDG generator or parser to find out the metrics values. The two code fragments having the same metric values may not have the identical code fragments dependent only on the metrics. PDG based techniques can find Type 3 clones but these techniques take a tremendous measure of time and are extremely complex. To convert a program into its PDG representation, its data flow graph and control flow graph are required. In AST based techniques, the source code is parsed into an abstract syntax tree (AST) using a parser and then the generated sub- trees are compared to detect the clones using tree-matching algorithms. The AST based linear representations of source code is effective than some any other comparison algorithms.

One of the advantages of the working on the lexical level is that the lexical stream better represents the “structure" of a program. The parse tree or Abstract syntax tree built from the lexical analysis of a program also shows the structure of the underlying program.

3.1 Gap Analysis

i. An efficient approach for code clone classification is required to classify code clones into respective categories.

ii. An AST or parse tree contains the whole information about the source code. In spite of the fact that the names of the variables and literal values of the source are disposed of in the tree representation, more refined strategies for the detection of clones.

iii. Abstract syntax tree have the entire information about the code. The result obtained from this technique is quite efficient.

3.2 Objective

The basic issue in clone detection is the disclosure of code fragments that figure the "same" result. To do this, first section the program in parts we will analyze, and a short time later decide whether fragment pairs are equivalent. Since determining that even a single fragment halts is impossible, it cannot determined that the two arbitrary program fragments halt under the similar circumstances, and accordingly it is impossible in theory to conclude that they compute identical results.

More straightforward meanings of code equivalence may do the trick if unreasonably various false positives are not created. This suggests the clone detection by more syntactical methods. One can go the level of comparing the source lines. Source line equality expect that the cloning process introduced no adjustments in comments, spacing, identifiers, or other non- semantic changes, and in this manner it limits clone detection to exact matches.

To solve the above stated problem, an approach is proposed for identifying the code clones. The main objective is to create a program that can find out the location of code clones and their types. There are various approaches involved in checking code clones, we are going to use Abstract Syntax Tree (AST) because ASTs offer syntactic knowledge which can be leveraged to filter certain types of clones. We implemented String Distance currently to label the clones as Type 1 and Type 2. Categorization is done on the basis of master expression strings.

Chapter 4

Proposed Approach and Implementation

In order to find clones using AST based approach we have to compare each subtree to each other subtrees in AST. Figuring out the similarities of all subtree pairs are not productive, where complexity of computation is O (N^3), where N is the number of nodes in AST. The advantage of the working with the AST is that it better represents the structure of a program other than any technique. So, AST based technique is more reliable for clone detection.

4.1 Techniques Used

The tree matching approach depends on the idea of comparing the sub trees in order to find the similarities among them with the purpose of detecting the clone from the code [12]. The tree represents the keywords, literals, identifiers, variable names and tokens which are abstracted from the source code [9]. The following techniques were used to find out the similarity among the code fragments on the basis of which the clones were identified.

4.1.1 Abstract Syntax Tree

An AST is a data structure that represents a piece of code in a hierarchical manner, and it greatly facilitates reasoning, analyzing, and even modifying the code in an automated way.
In order to translate code from one language to another, or to analyze and reason about code, the first step is to *parse* the code. In simple terms, parsing is like grammatically dissecting a text passage, sentence by sentence, clause by clause, word by word, and assigning *a category* to each word, such as noun, verb, adjective, etc., as well as properties, such as subject, object, etc. Parsing code results in an *Abstract Syntax-Tree (or AST),* which is a tree-shaped data structure that represents the code, but is easier to manipulate than plain text. Once the code is in AST form, a sequence of analyses and transformations are applied to the AST as it is gradually translated to the target language.

For example, parsing the python expression x * (3+y) will produce the following AST:

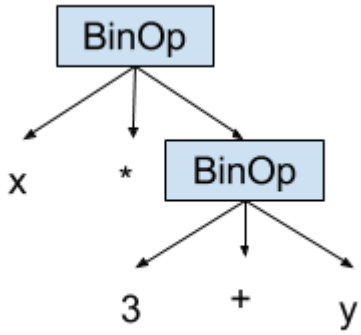


Figure 4.1: Example of AST

where BinOp is a kind of expression that is made up of a binary operator, a left hand side (LHS)expression, and a right hand side(RHS) expression. The LHS and RHS expressions can in turn be *identifiers* (e.g., x, y), *constants* (e.g., 3), other binary expressions, or other types of expressions.

The python AST module provides tooling to convert Python code from textual form into its corresponding Abstract Syntax Trees, and for manipulating such trees.

Basic Approach

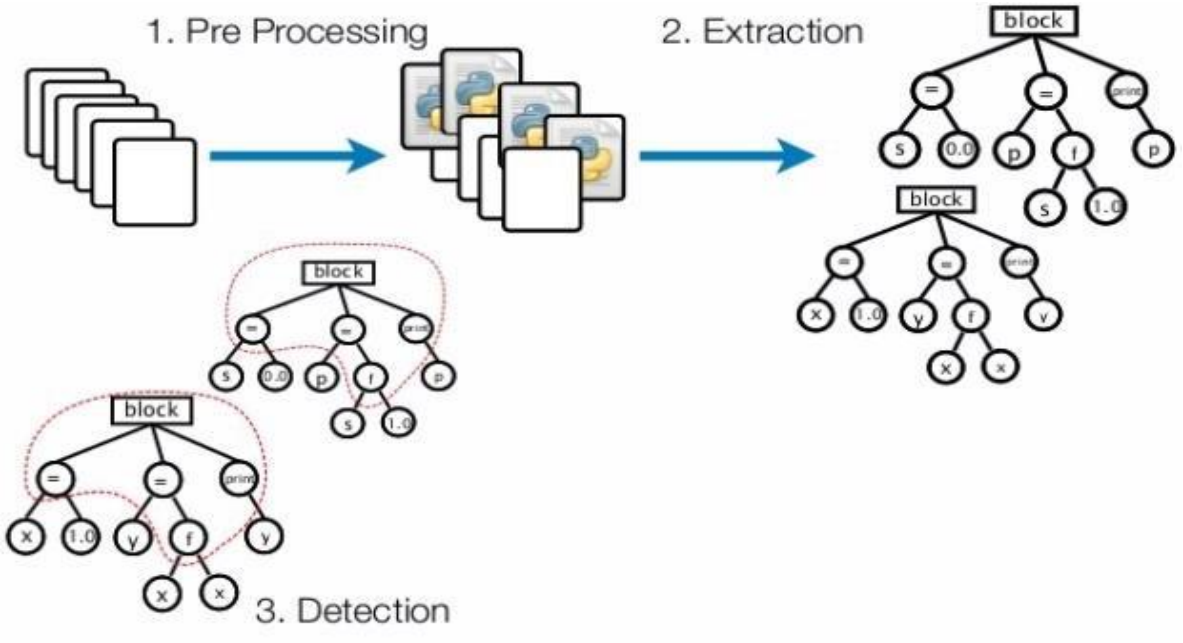


Figure 4.2: Overall process

4.1.2 Levenshtein Distance

Levenshtein distance is named after the Russian scientist Vladimir Levenshtein, who devised the algorithm in 1965. Levenshtein Distance is also sometimes called as edit distance.

Whenever a program or an application is required to use some kind of spell checking or error correction this technique is used.

The Levenshtein distance among the two strings a and b is given by $leven_{a,b}(len(a), len(b))$ where $leven_{a,b}(i, j)$ is equal to

- max(i, j) if min(i, j)=0
- otherwise:

  min(levena,b(i-1, j) + 1,

  levena,b(i, j+1) + 1,

  levena,b(i-1, j-1) + 1ai≠bj)

where 1ai≠bj</sub> is the indicator function which is equal to 0 when ai=bj and equal to 1 otherwise, and levena,b(i, j) is the distance between the first i characters of a and the first j characters of b.

The Levenshtein distance has the following properties:

- Its value is zero iff the strings are equal.
- It gives the least difference between the sizes of the two strings.
- It gives at most the length of the longest string.

4.2 Proposed Approach

The current work presents an approach to detect the number of clones and its classification.

1. For implementation a python script “test.py” is taken as an input and firstly comments are removed. A file named “clonedlib.py” is generated after the comment removal. Comments are removed because they tend to be repetitive and they would

appear a lot in the clones, the main focus is to concentrate on the code rather than comments. In the comment removal process tokenization and lexical analysis techniques are used.

2. Then the updated source code file without comments is passed to generate the AST. The Abstract Syntax Trees are generated and then compared with each other to find out the clones.

3. Then we store the clones generated in the previous step and get the list of clones. We iterate over this list, and mark the lines and the strings which consist of the clone. As there are multiple lines in the code, where clone is detected, we count those and put those back in a separate .csv file for the ease analyze the code clones and their classification.

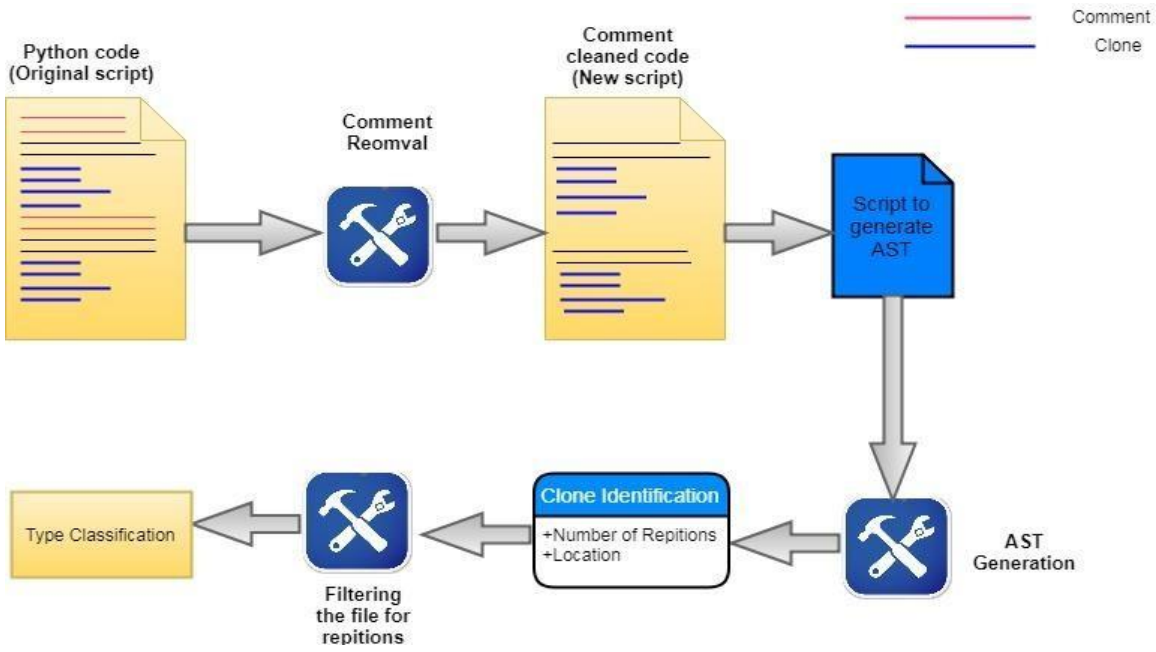
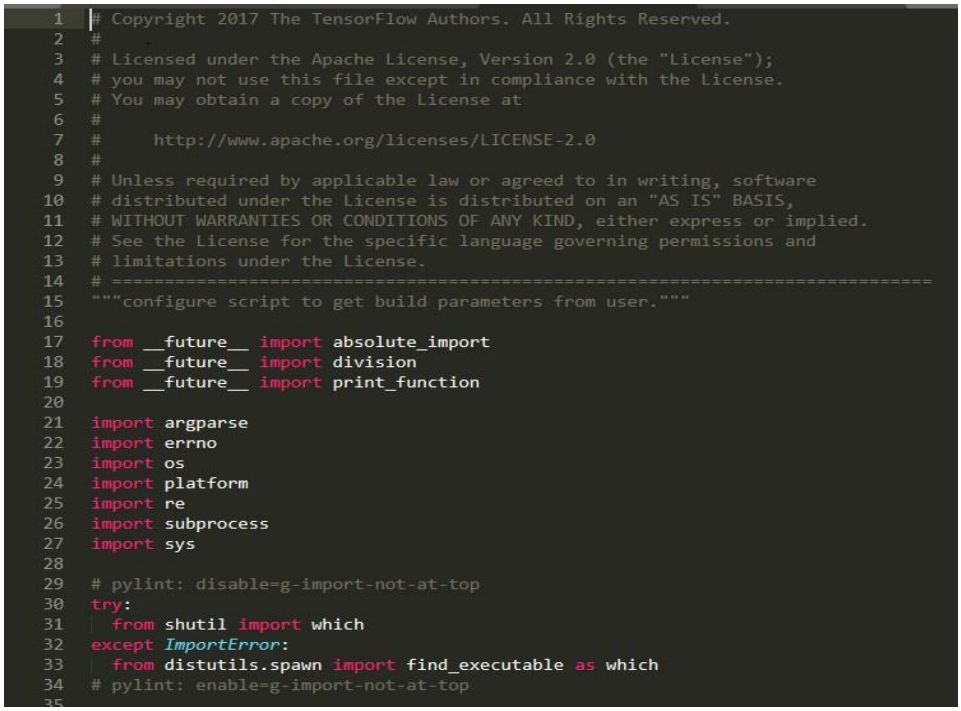


Figure 4.3: Block Diagram of Implementation

4.3 Execution of the Proposed Work

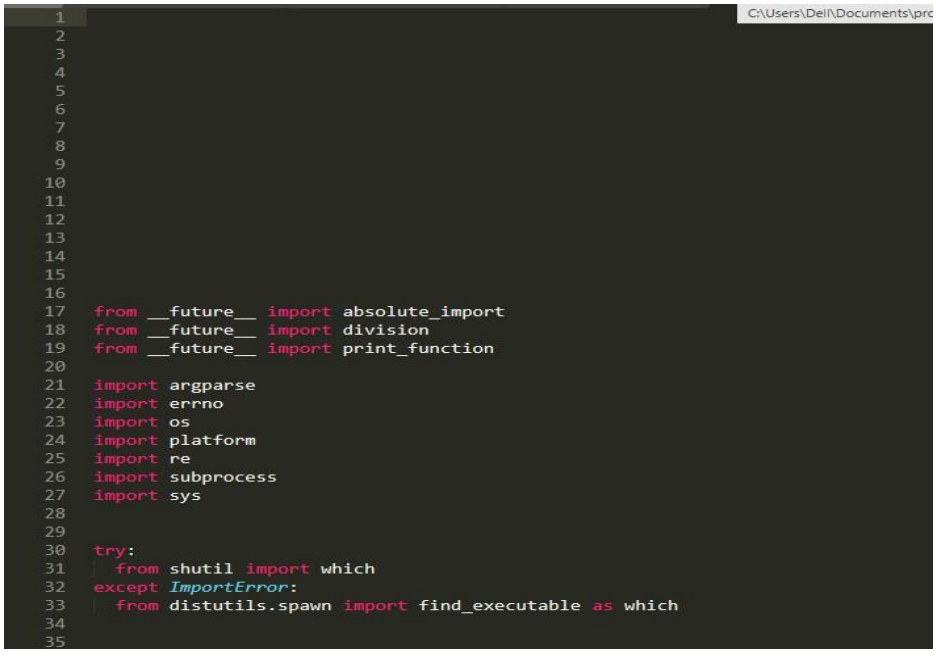
The proposed approach is implemented in python. At the very first step we input a python script “test.py” to get comments free file because we are more focused on finding duplication in the code not in the comments.



```
# Copyright 2017 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
# ==============================================================================
"""configure script to get build parameters from user."""

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import errno
import os
import platform
import re
import subprocess
import sys

# pylint: disable=g-import-not-at-top
try:
  from shutil import which
except ImportError:
  from distutils.spawn import find_executable as which
# pylint: enable=g-import-not-at-top
```

Figure 4.4: Input file which includes the comments

After the input file gets passed for comment removal, the comments get removed and a new file “cleanedlib.py” is created without comments.



```
C:\Users\Dell\Documents\pro

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import errno
import os
import platform
import re
import subprocess
import sys


try:
  from shutil import which
except ImportError:
  from distutils.spawn import find_executable as which
```

Figure 4.5: Output file generated after comment removal

Now the comment removed file is passed to the script that generates the AST and searches for the code clones. These clones are stored in a .csv file and we get the list of clones. Then we iterate over the list and will get the lines and the master string expression which contains the clone. The Master String Expression is basically a string expression in balanced parenthesis form.

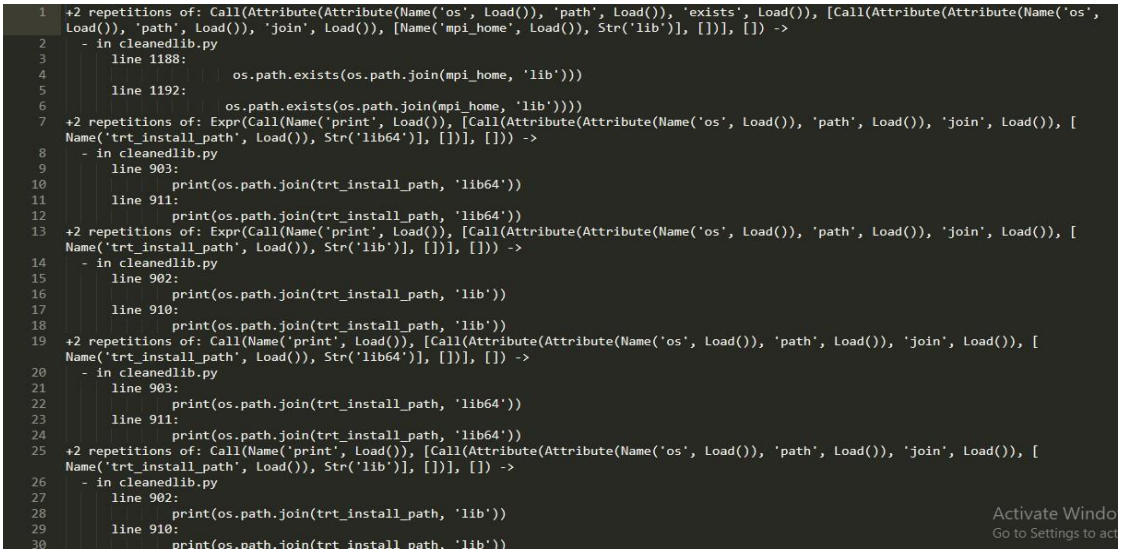


Figure 4.6: Count of clones including the string and the line number

As there are various lines in the code where the clones are detected, so a .csv file is generated which contains the count of the clones with respect to the string.

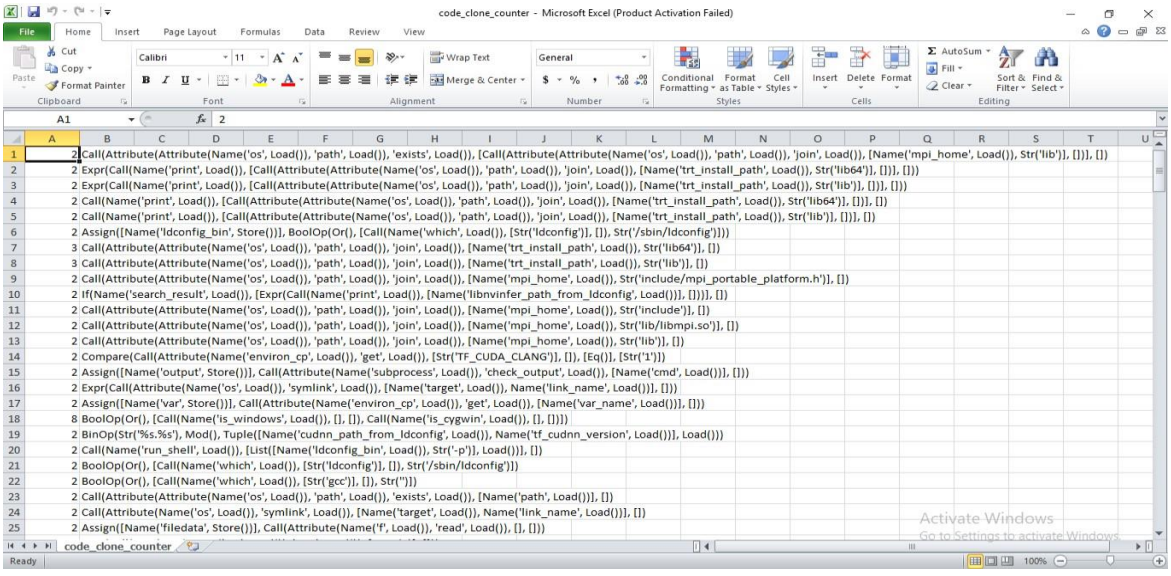


Figure 4.7: Number of clones detected

At the next step code clones are classified into defined categories as type 1 and 2. Then we

iterate over the list and will get the type of clone and the string.

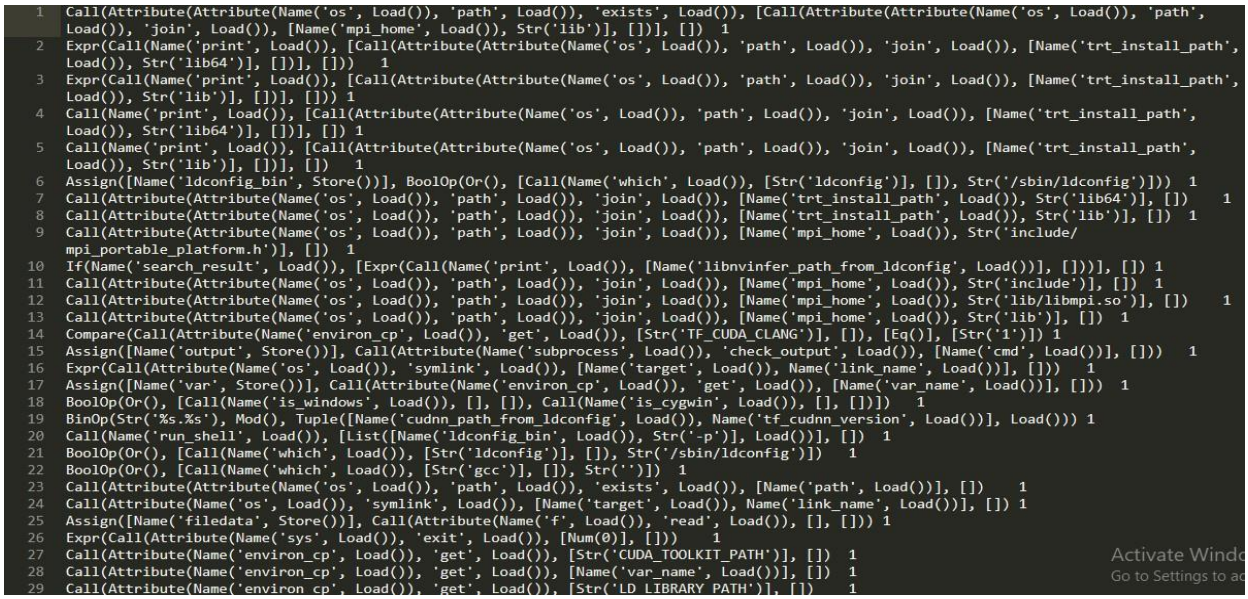


Figure 4.8: Classification of Code Clone

Now the results are stored into a .csv file.

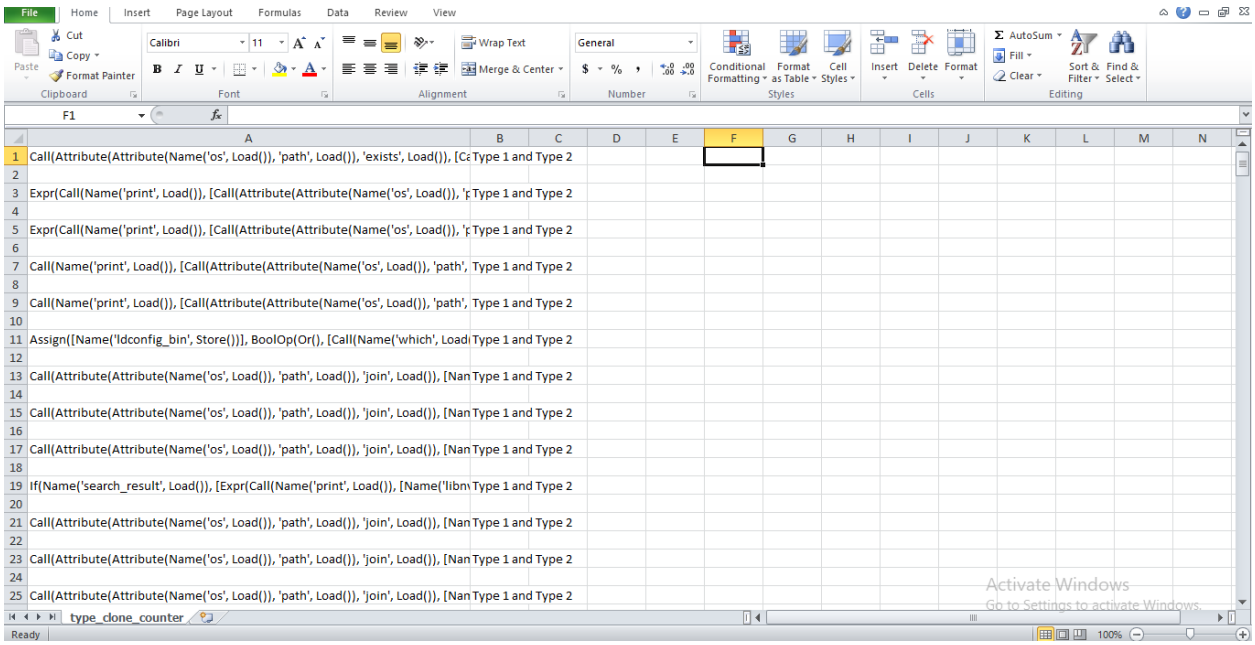


Figure 4.9: .csv file of the type of clone

Further, in next chapter results of proposed approach are verified on subject system and compared with tool “Clone Digger”.

Chapter 5

Experimental Results

As per the proposed approach clones has been classified. Now the results have been compared with the results of Clone Digger on subject system.

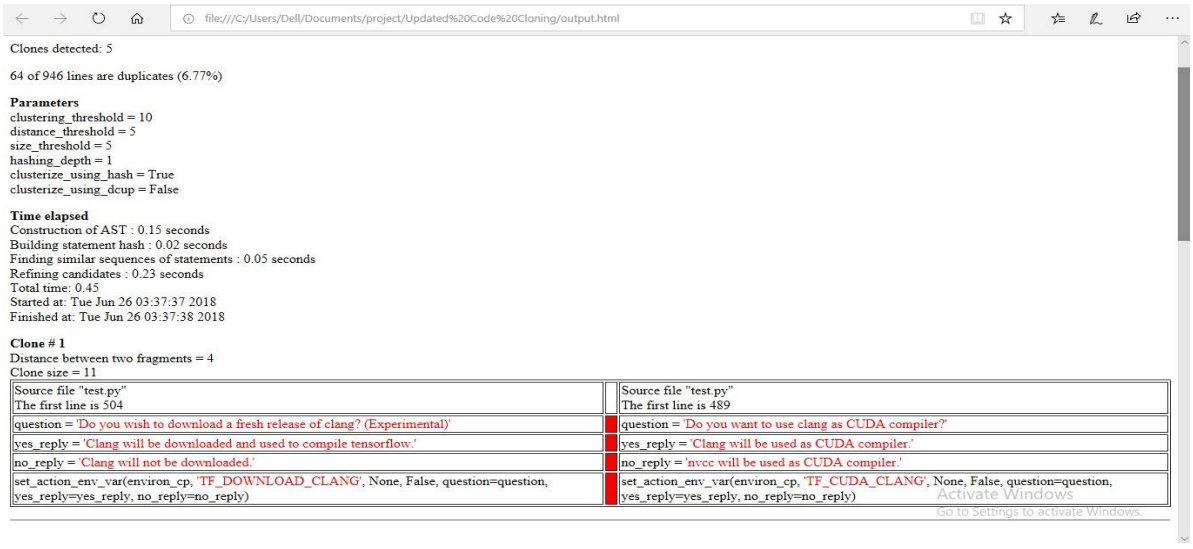
Clone Digger is a duplicate code detection tool, which supports Python and Java languages. Discovered clones can differ in small sub expressions, comments and whitespaces are ignored. Clone digger is platform-independent and is implemented in Python language.
Clone Digger is a tree-based clone detection approach. At first it generates Abstract Syntax Tree (AST) by parsing the source code. Then matches the sub-trees to detect the clones.

Clone Digger:

- ignores the comments and whitespaces
- supports clone parameterization,
- allows the changes of function names, variable and constants

Clone Digger is not efficient to find out the gapped clones.

Although Clone Digger is a commercial tool, it detects only the code clones and the line number at which the clone is detected but doesn't detect the type of the clone.


file:///C:/Users/Dell/Documents/project/Updated%20Code%20Cloning/output.html

Clones detected: 5

64 of 946 lines are duplicates (6.77%)

Parameters
clustering_threshold = 10
distance_threshold = 5
size_threshold = 5
hashing_depth = 1
clusterize_using_hash = True
clusterize_using_dcup = False

Time elapsed
Construction of AST : 0.15 seconds
Building statement hash : 0.02 seconds
Finding similar sequences of statements : 0.05 seconds
Refining candidates : 0.23 seconds
Total time: 0.45
Started at: Tue Jun 26 03:37:37 2018
Finished at: Tue Jun 26 03:37:38 2018

Clone # 1
Distance between two fragments = 4
Clone size = 11

Source file "test.py" The first line is 504		Source file "test.py" The first line is 489
question = 'Do you wish to download a fresh release of clang? (Experimental)'		question = 'Do you want to use clang as CUDA compiler?'
yes_reply = 'Clang will be downloaded and used to compile tensorflow.'		yes_reply = 'Clang will be used as CUDA compiler.'
no_reply = 'Clang will not be downloaded.'		no_reply = 'nvcc will be used as CUDA compiler.'
set_action_env_var(environ_cp, 'TF_DOWNLOAD_CLANG', None, False, question=question, yes_reply=yes_reply, no_reply=no_reply)		set_action_env_var(environ_cp, 'TF_CUDA_CLANG', None, False, question=question, yes_reply=yes_reply, no_reply=no_reply)

Activate Windows
Go to Settings to activate Windows.

Figure 5.1: Clone detection result by Clone Digger

As the proposed approach gives both the number of clones and the lines at which the clones are present and also specifies the types of the clones whereas the tool “Clone Digger” detects the clones and the line number where the clones are present but it does not specify the type of the clone.

The following are the results of comparison between the proposed approach and Clone Digger for the types of clones detected and the percentage of lines of duplication.

The proposed approach is capable of detecting up to type-2 clone whereas Clone Digger is not able to detect the type of clone.

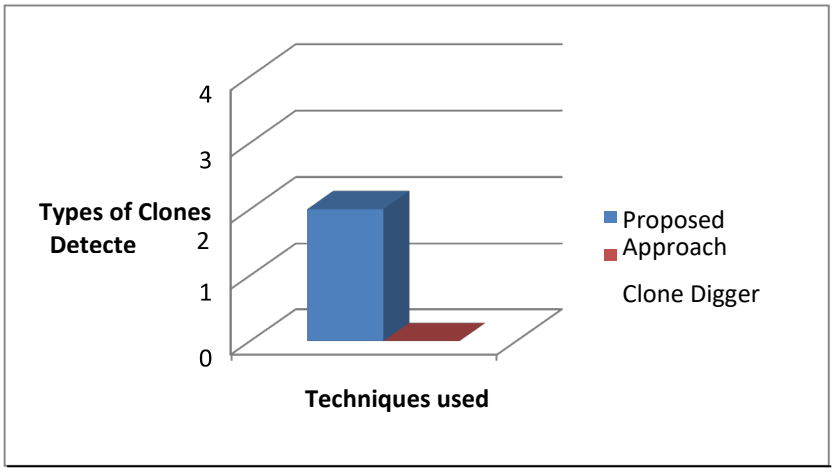


Figure 5.2: Comparison of Techniques for Types of Clones Detected

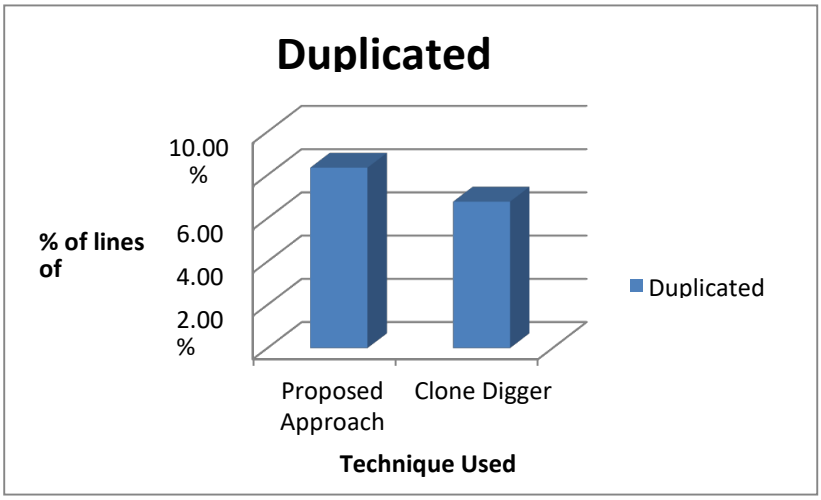


Figure 5.3: Comparison between the Techniques for the % of lines are duplicates

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

The proposed approach generates the count, line number and type of clone. In the proposed approach a python script was taken as an input and first of all comments are removed. Then the modified file is passed to the script that can generate AST and detects the clones. Now, as a result of this we get the list of clones. As there are numerous lines in the code at where the clones are detected, we count those and put those back in a separate CSV file. Types of clones are also stored separately. Now the results are compared with the clone detection tool- Clone Digger which can only specify the count and line number of the clone but unable to detect the type of the clone.

6.1 Future Scope

- The most obvious next step is to automate the removal of detected clones from the source.
- The proposed approach can be extended so that it can detect Type 3 and Type 4 categories of code clones.
- The proposed approach can be applied to the larger subject system and object oriented software systems.
- The proposed approach can be extended to detect the clones in other programming language also.

REFERENCES

[1] B. Hummel, E. Juergens, et al. “Index-Based Code Clone Detection: Incremental, Distributed, Scalable”, *In Proc. of the 26th IEEE International Conference on Software Maintenance*, pp 1–9. 2010.

[2] B. Stefan, K. Rainer, et al., “Comparison and Evaluation of Clone Detection Tools*”, IEEE Transactions on Software Engineering*, pp.577–591, vol.33, no.9, 2007.

[3] C.K. Roy and J.R. Cordy. “A Survey on Software Clone Detection Research. School of Computing TR”, *Queen’s University*, pp 1-115. 2007.

[4] H. Yoshiki, U. Yasushi, et al., “Incremental Code Clone Detection: A PDG-based Approach”, IEEE18th *Working Conference on Reverse Engineering*. pp 3 – 12, 2011.

[5] I.Mai,O. Shunsuke and N. Takuo, “Token-based Code Clone Detection Technique in a Student’s Programming Exercise” , *2012 Seventh International Conference on broadband, Wireless Computing,* Communication and Applications, Victoria, BC, pp. 650-655. 2012.

[6] J. Krinke. , “Is Cloned Code more stable than Non- Cloned Code?”, In Proc. of the 8th IEEE International Working Conference on Source Code Analysis and Manipulation, pp. 57–66, Oct. 2008.

[7] J. Krinke, “Identifying Similar Code with Program Dependence Graphs”, *Working Conference on Reverse Engineering.* pp. 301-309, 2001.

[8] K. Rainer. Raimar, F. Pierre, “Clone Detection Using Abstract Syntax Suffix Trees”, *Working Conference on Reverse Engineering,* pp 253-262. 2006.

[9] M. Hiroaki, H. Keisuke et al. , “Folding Repeated Instructions for Improving Token-based Code Clone Detection”,*2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation,* Trento, pp. 64 – 73. 2012.

[10] M. Kazuaki, “An Extended Line-Based Approach to Detect Code Clones Using Syntactic and Lexical Information”, *IEEESeventh International Conference on Information Technology*, Vegas, NV, pp.1237 – 1240, 2010.

[11] Nguyen H.A., Nguyen T.T. et.al,"Clone Management for Evolving Software", *IEEE transactions on software engineering*, vol 38 no 5, pp 1008-1026. 2012.

[12] Salwa K. Abd-El-Hafiz ,A Metrics-Based Data Mining Approach for Software Clone Detection,2012 IEEE 36th International Conference on Computer Software and Applications, pp35-41,Izmir. 2012.

[13] Sarkar, M.; Chudamani, S.; Roy, S.; Mukherjee, N. et al, “A hybrid clone detection technique for estimation of resource requirements of a job”, *Third International Conference on Advanced Computing & Communication Technologies*, Rohtak, pp. 174-181. 2013.

[14] J.W.Hunt and M.D.McIlroy, “An Algorithm for Differential File Comparison”, Bell Laboratories, Murray Hill, New Jersey, 1976.

[15] http://clonedigger.sourceforge.net/

[16] C.K. Roy and J.R. Cordy, NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization, in: Proceedings of the 16th IEEE International Conference on Program Comprehension, ICPC 2008, pp. 172-181 (2008). [20] P. Bulychev and M. Minea, Duplicate Code Detection Using Anti-Unification, in: Spring Young Researchers Colloquium on Software Engineering, SYRCoSE 2008,4 pp. (2008).

[17] Minhaz F. Zibran, Chanchal K. Roy: IDE-based Real-time Focused Search for Near-miss Clones in SAC’12 March 25-29, 2012, Riva del Garda, Italy. Copyright 2011 ACM 978-1-4503-0857-1/12/03

[18] Fabio Calefato, Filippo Lanubile, Teresa Mallardo, "Function Clone Detection in Web Applications: A Semiautomated Approach", Journal of Web Engineering, Vol. 3, No.1, pp.003-021, 2004.

[19] Muhammad Asaduzzaman, “Visualization and Analysis of Software Clones”, A Thesis in the Department of Computer Science University of Saskatchewan Saskatoon January 2012.

[20] R.Wettel and R. Marinescu, Archeology of Code Duplication: Recovering Duplication Chains From Small Duplication Fragments, in:Proceedings of the 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC, **(2005)**, p. 8.

[21] 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 1999, (1999), pp. 109-118.

[22] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code, IEEE Transactions on Software Engineering, vol. 32, no. 3, (2006), pp. 176-192.

[23] I.D. Baxter, A. Yahin, L.M. de Moura, M. Sant’Anna, L. Bier, Clone detection using abstract syntax trees, in: ICSM’98: Proceedings of the 14th IEEE International Conference on Software Maintenance, IEEE Computer Society, 1998, pp. 368–377.

[24] R. Geiger, B. Fluri, H.C. Gall, M. Pinzger, Relation of code clones and change couplings, in: FASE’06: Proceedings of the 9th International Conference of Funtamental Approaches to Software Engineering, Springer, 2006, pp. 411–425.

[25] E. Juergens, F. Deissenboeck, B. Hummel, S. Wagner, Do code clones matter? in: ICSE’09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, IEEE Computer Society, 2009, pp. 485–495.

[26] T. Kamiya, S. Kusumoto, K. Inoue, Ccfinder: a multilinguistic token-based code clone detection system for large scale source code, IEEE Transactions on Software Engineering 28 (7) (2002) 654–670.

[27] K. Kontogiannis, R. de Mori, E. Merlo, M. Galler, M. Bernstein, Pattern matching for clone and concept detection, Automated Software Engineering 3 (1–2) (1996) 77–108.

[28] A. Lozano, M. Wermelinger, Assessing the effect of clones on changeability, in: ICSM'08: Proceedings of the 24th IEEE International Conference on Software Maintenance, IEEE, 2008, pp. 227–236.

[29] F. Rahman, C. Bird, P.T. Devanbu, Clones: what is that smell? in: MSR'10: Proceedings of the 7th IEEE Working Conference on Mining Software Repositories, IEEE, 2010, pp. 72–81.

[30] S. Thummalapenta, L. Cerulo, L. Aversano, M.D. Penta, An empirical study on the maintenance of source code clone, Empirical Software Engineering 15 (1) (2010) 1–34.

[31] E. Duala-Ekoko, M.P. Robillard, Tracking code clones in evolving software, in: ICSE'07: Proceedings of the 29th International Conference on Software Engineering, IEEE Computer Society, 2007, pp. 158–167.

[32] S. Thummalapenta, L. Cerulo, L. Aversano, M.D. Penta, An empirical study on the maintenance of source code clone, Empirical Software Engineering 15 (1) (2010) 1–34.

[33] T. Kamiya, S. Kusumoto, and K. Inoue, CCFinder: A multi-linguistic tokenbased code clone detection system for large scale source code IEEE Transactions on Software Engineering, 28(7):654-670, 2002.

[34] Katsuro Inoue, "Code Clone Analysis and Its Application", Software Engineering Lab, Osaka University

[35] Matthias Rieger. Effective Clone Detection Without Language Barriers. Ph.D. Thesis, University of Bern, Switzerland, June 2005

[36] Andrew Walenstein and Arun Lakhotia. The Software Similarity Problem in Malware Analysis. In Proceedings Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, 10 pp., Dagstuhl, Germany, July 2006

[37] Brenda Baker. On Finding Duplication and Near-Duplication in Large Software Systems. In Proceedings of the Second Working Conference on Reverse Engineering (WCRE'95), pp. 86-95, Toronto, Ontario, Canada, July 1995

[38] Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue. CCFinder: A Multilinguistic Token- Based Code Clone Detection System for Large Scale Source Code. Transactions on Software Engineering, Vol. 28(7): 654- 670, July 2002

[39] Elizabeth Burd and Malcolm Munro. Investigating the maintenance implications of the replication of code. In Proceedings of the 13th International Conference on Software Maintenance (ICSM'97), Bari, Italy, September 1997

[40] Matthias Rieger. Effective Clone Detection Without Language Barriers. Ph.D. Thesis, University of Bern, Switzerland, June 2005

APPENDIX-A:

PLAGIARISM REPORT

Kanika Thesis

ORIGINALITY REPORT

14%	4%	10%	5%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	www.python-course.eu Internet Source	2%
2	Harpreet Kaur, Raman Maini. "Identification of Recurring Patterns of Code to Detect Structural Clones", 2016 IEEE 6th International Conference on Advanced Computing (IACC), 2016 Publication	1%
3	"Smart Trends in Information Technology and Computer Communications", Springer Nature, 2016 Publication	1%
4	Dhavleesh Rattan, Jagdeep Kaur. "Systematic Mapping Study of Metrics based Clone Detection Techniques", Proceedings of the International Conference on Advances in Information Communication Technology & Computing - AICTC '16, 2016 Publication	1%
5	iosrjen.org Internet Source	

		1%
6	Dhavleesh Rattan, Rajesh Bhatia, Maninder Singh. "Software clone detection: A systematic review", Information and Software Technology, 2013 Publication	1%
7	Submitted to Madan Mohan Malaviya University of Technology Student Paper	1%
8	K. Inoue. "Gemini: maintenance support environment based on code clone analysis", Proceedings Eighth IEEE Symposium on Software Metrics METRIC-02, 2002 Publication	1%
9	Ritesh V. Patil, Shashank. D. Joshi, Sachin V. Shinde, Digvijay A. Ajagekar, Shubham D. Bankar. "Code clone detection using decentralized architecture and code reduction", 2015 International Conference on Pervasive Computing (ICPC), 2015 Publication	1%
10	T. Kamiya, S. Kusumoto, K. Inoue. "CCFinder: a multilinguistic token-based code clone detection system for large scale source code", IEEE Transactions on Software Engineering, 2002	<1%

	Publication	
11	Submitted to California State University, San Bernardino Student Paper	<1%
12	Minhaz F. Zibran, Chanchal K. Roy. "IDE-based real-time focused search for near-miss clones", Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC '12, 2012 Publication	<1%
13	Raimar Falke. "Empirical evaluation of clone detection using syntax suffix trees", Empirical Software Engineering, 12/2008 Publication	<1%
14	Yerragudipadu Subbarayadu, G. Geetha, N.M. Deepika, Syed Umar. "VANET for leveraging for caching network content distribution", 2017 International Conference on Intelligent Computing and Control Systems (ICICCS), 2017 Publication	<1%
15	Submitted to University of Nottingham Student Paper	<1%
16	James R. Cordy. "Scenario-Based Comparison of Clone Detection Techniques", 2008 The 16th IEEE International Conference on Program Comprehension, 06/2008	<1%

	Publication	
17	Submitted to Institute of Technology, Nirma University Student Paper	<1%
18	dspace.thapar.edu:8080 Internet Source	<1%
19	semanticdesigns.com Internet Source	<1%
20	homes.cs.washington.edu Internet Source	<1%
21	Submitted to Malaviya National Institute of Technology Student Paper	<1%
22	Raminder Kaur, Satwinder Singh. "Clone detection in software source code using operational similarity of statements", ACM SIGSOFT Software Engineering Notes, 2014 Publication	<1%
23	www.cs.caltech.edu Internet Source	<1%
24	Biswas, Swarnendu Mall, Rajib Satpathy, . "Regression test selection techniques: a survey. (Report)", Informatica, Sept 2011 Issue Publication	<1%
25	Tekchandani, R.K., and K. Raheja. "An Efficient	<1%

	Code Clone Detection Model on Java Byte Code Using Hybrid Approach", Confluence 2013 The Next Generation Information Technology Summit (4th International Conference), 2013. Publication	
26	James R. Cordy. "NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization", 2008 The 16th IEEE International Conference on Program Comprehension, 06/2008 Publication	<1%
27	Seulbae Kim, Seunghoon Woo, Heejo Lee, Hakjoo Oh. "VUDDY: A Scalable Approach for Vulnerable Code Clone Discovery", 2017 IEEE Symposium on Security and Privacy (SP), 2017 Publication	<1%
28	shodhganga.inflibnet.ac.in Internet Source	<1%
29	Bharavi Mishra, K. K. Shukla. "chapter 7 Data Mining Techniques for Software Quality Prediction", IGI Global, 2013 Publication	<1%
30	Lecture Notes in Computer Science, 2002. Publication	<1%
31	Yoshiki Higo. "Refactoring Support Based on Code Clone Analysis", Lecture Notes in	<1%

	Computer Science, 2004 Publication	
32	C. Stoermer, L. O'Brien, C. Verhoef. "Practice patterns for architecture reconstruction", Ninth Working Conference on Reverse Engineering, 2002. Proceedings., 2002 Publication	<1%
33	Paul Freeman, Ian Watson, Paul Denny. "Chapter 10 Inferring Student Coding Goals Using Abstract Syntax Trees", Springer Nature, 2016 Publication	<1%
34	A.A. Reyes, J.R. Espino, V. Mohan, M. Nadkar. "Ad hoc software interfacing: enterprise application integration (EAI) when middleware is overkill", Proceedings 27th Annual International Computer Software and Applications Conference. COMPAC 2003, 2003 Publication	<1%
35	Gehan M.K. Selim, King Chun Foo, Ying Zou. "Enhancing Source-Based Clone Detection Using Intermediate Representation", 2010 17th Working Conference on Reverse Engineering, 2010 Publication	<1%
36	Wang, Bin Yang, Xiaochun Wang, Guoren. "Detecting copy directions among programs	<1%

using extreme learning machines.(Research Article)(Report)", Mathematical Problems in Engineering, Annual 2015 Issue
Publication

Exclude quotes	On	Exclude matches	< 5 words
Exclude bibliography	On		