ENHANCING GENERIC CODE CLONE DETECTION MODEL THROUGH PROTECTED ACCESS MODIFIER RULE AND WEIGHTAGE

SITI AN NASIHAH BINTI MOHD NAPI

A thesis submitted in fulfilment of the

requirements for the award of the degree of

Degree of Computer Science (Software Engineering) with Honors

Faculty of Computer Systems & Software Engineering

Universiti Malaysia Pahang

DECEMBER 2016

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : SITI AN NASIHAH BINTI MOHD NAPI
Date of Birth : 30 MAY 1994

Title : ENHANCING GENERIC CODE CLONE DETECTION MODEL THROUGH PROTECTED ACCESS MODIFIER RULE AND WEIGTHAGE

Academic Session : SEMESTER 1 16/17

I declare that this thesis is classified as:

- ☐ CONFIDENTIAL (Contains confidential information under the Official Secret Act 1997)*
- ☐ RESTRICTED (Contains restricted information as specified by the organization where research was done)*
- ☐ OPEN ACCESS I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:

_____	_____
(Student's Signature)	(Supervisor's Signature)
940530-03-5092	_____
New IC/Passport Number	Name of Supervisor
Date:	Date:

NOTE : * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

THESIS DECLARATION LETTER

Librarian,
Perpustakaan Universiti Malaysia Pahang,
Universiti Malaysia Pahang,
Lebuhraya Tun Razak,
26300, Gambang, Kuantan.

Dear Sir,

CLASSIFICATION OF THESIS AS RESTRICTED

Please be informed that the following thesis is classified as RESTRICTED for a period of
Author's Name
Thesis Title

Reasons (i)

(ii)

(iii)

three (3) years from the date of this letter. The reasons for this classification are as listed below.

Thank you.

Yours faithfully,

_____
(Supervisor's Signature)

Date:

Stamp:

Note: This letter should be written by the supervisor, addressed to the Librarian, *Perpustakaan Universiti Malaysia Pahang* with its copy attached to the thesis.

SUPERVISOR'S DECLARATION

I hereby declare that I have read this thesis and in my opinion this thesis/ report is sufficient in term of scope and quality for the award of the degree of Bachelor of Computer Science (Software Engineering).

_____

(Supervisor's Signature)

Full Name : Dr. Al-Fahim Bin Mubarak Ali

Position :

Date :

STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

_____

(Student's Signature)

Full Name : SITI AN NASIHAH BINTI MOHD NAPI

ID Number : CB13058

Date : 06 DECEMBER 2016

ACKNOWLEDGEMENTS

In the name of Allah, the Most Gracious and the Most Merciful

First and foremost, praise to Almighty Allah for all HIS blessings for giving me patience and good health to complete this project successfully. With his blessing and guidance, all obstacles and problems managed to be solved. Alhamdulillah.

Next, the deepest appreciation to my project supervisor, Dr. Al-Fahim Bin Mubarak Ali for his supportive and useful guidance from start until finishes this project. I really appreciated the support that was given by my supervisor throughout this project development. The highest gratitude to my supervisor who never gives up share with me relevant information related with my project title.

To my beloved family, who supports me all the times, thanks for the precious motivations, priceless advices and finance support for me to finish this project and I am grateful beyond words and supports for all that they have given me.

Lastly, I would like to express my high appreciation to my supportive friends who always give me great ideas and solutions for my project. Thank you very much for the endless support, the constructive criticism as well as help offered in completing this project.

ABSTRACT

Code clone is a common term used to refer codes that have been repeated multiple times in a program. There are four types of code clone which are type I, type II, type III and type IV. Code clone detection models have been used to detect clones apart from code clone detection approaches by applying the protected access modifier rule and weightage. The major challenge faced in detecting code clone using models is the lack of generality in detecting all clone types. This is due to the use of different code clone detection approaches in the models that represents different representation of the source codes; hence it affecting the type of code clones detected. Based on this weakness, it is essential to propose a code clone detection model that can support different type of code. To overcome this weakness, Generic Code Clone Detection model that consists of five processes which are Pre-processing, Transformation, Parameterization, Categorization or called as pooling and Match Detection process has been proposed. A prototype has been developed to detect all code clone types in Java. The proposed method was evaluated in two case studies comprised of three Java applications. The result shows the Generic Code Clone Model prototype was able to detect Type I, Type II, Type III and Type IV clone pairs. The results imply that the Generic Code Clone Model was able to detect all code clone types in Java applications and the generated Generic Code Clone Model have better visualization of the code clone detection results.

ABSTRAK

Kod klon adalah istilah umum yang digunakan untuk merujuk kod yang telah diulang beberapa kali dalam program. Terdapat empat jenis kod klon yang jenis I, jenis II, jenis III dan jenis IV. Kod model pengesanan klon telah digunakan untuk mengesan klon selain pendekatan pengesanan kod klon dengan menggunakan akses peraturan pengubahsuai yang dilindungi dan pemberat. Cabaran utama yang dihadapi dalam mengesan kod klon menggunakan model adalah kekurangan keluasan dalam mengesan semua jenis klon. Ini adalah kerana penggunaan pendekatan pengesanan kod klon yang berbeza dalam model yang menghasilkan perwakilan yang berbeza daripada kod sumber; oleh itu ia memberi kesan kepada jenis klon kod dikesan. Berdasarkan kelemahan ini, ia adalah penting untuk mencadangkan model pengesanan kod klon yang boleh menyokong pelbagai jenis kod. Untuk mengatasi kelemahan ini, model pengesanan *Generic Code Clone* yang terdiri daripada lima proses yang Pra-pemprosesan, Transformasi, parameterization, Pengkategorian atau dipanggil sebagai pengumpulan dan proses pengesanan Perlawanan telah dicadangkan. prototaip telah dibangunkan untuk mengesan semua jenis kod klon di Jawa. Kaedah yang dicadangkan telah dinilai dalam dua kajian kes terdiri daripada tiga aplikasi Java. Hasilnya menunjukkan prototaip pengesanan *Generic Code Clone* dapat mengesan Jenis I, Jenis II, Jenis III dan IV Jenis pasangan klon. Keputusan membayangkan bahawa pengesanan *Generic Code Clone* automatik mampu untuk mengesan semua jenis kod klon dalam aplikasi Java dan yang dihasilkan pengesanan *Generic Code Clone* mempunyai visualisasi lebih baik keputusan pengesanan kod klon.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE NO. TITLE PAGE

LIST OF TABLES

TABLE NO. **TITLE** **PAGE**

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Refer to IEEE standard; definition of software development process is concerned primarily with the production aspect of software development, as opposed to the technical aspect, such as software tools. These processes exist primarily for supporting the management of software development, and are generally skewed toward addressing business concerns. Many software development processes can be run in a similar way to general project management processes. A clone occurs when a code fragment is an identical to another code fragment according to some basic criteria. These criteria may be syntactical, semantical, or both of them (El-Matarawy, El-Ramly, & Bahgat, 2013). In software project management, it is common to reuse some code fragments by copying with or without minor modifications for use in different programs or maintained by the same entity.

Most of the software systems consist of a large number of identical code segments. These identical code segments are known as code clones. According to previous research, a software system consists of about 7% to 23% of cloned code (Yuan & Guo, 2011). There are four types of code clone which are type I, type II,

type III and type IV. Below is the diagram to show the difference of four types of code clone.

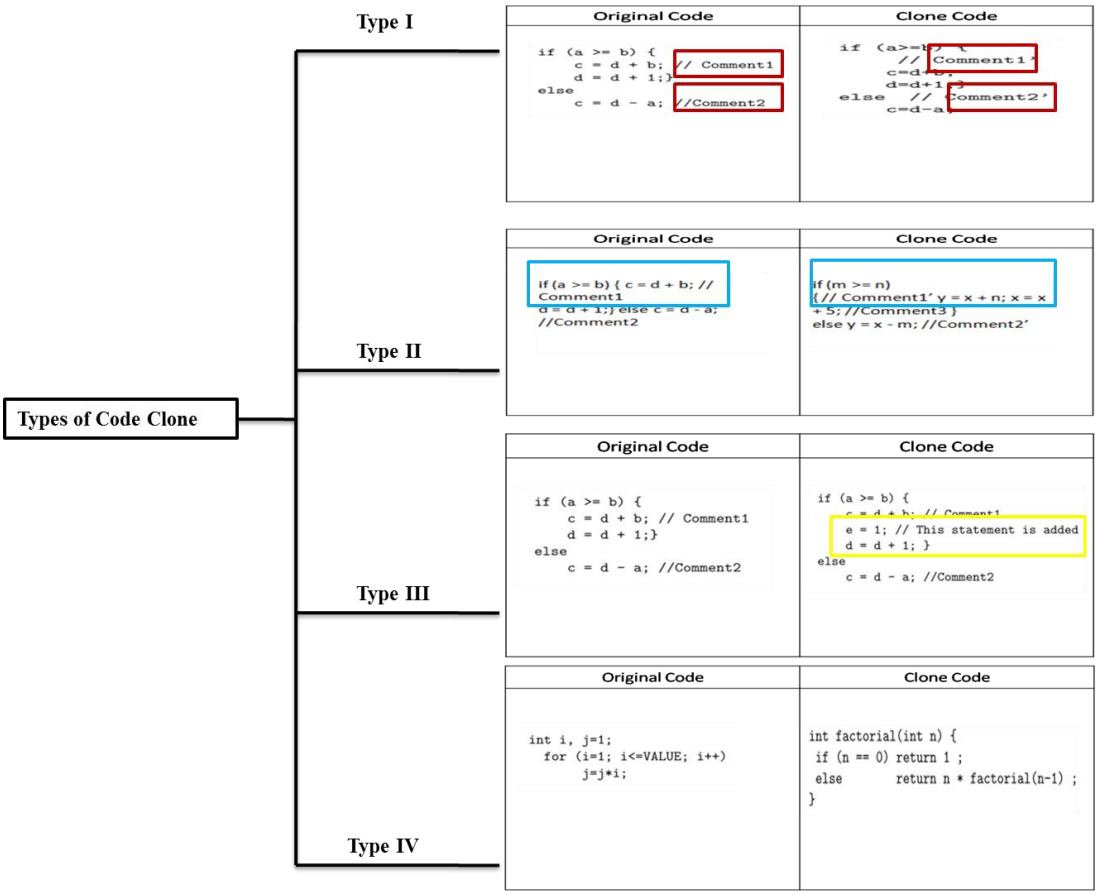


Figure 1.1: The Types of Code Clone

For type I for above figure, these two fragments are textually after removing the whitespace and comments. Next, for the type II of code clone in above figure shows that the two code segments change a lot in their shape, variable names and value assignments. However, the syntactic structure is still similar in both segments. Type III in the figure of types of code clone above shows that the two fragments and from the corresponding difference, all the original statements are used directly or after being changed in their identifiers or literals with one insertion in the first line, making this code fragment as Type III of code clone. Without this inserted statement, this copied fragment could be a Type II code clone. For type IV of code clone in the figure above shows that from the semantics point of view both the code fragments

are similar in their functionality and termed as Type IV semantic clones although one is a simple code fragment and another is a recursive function with no structural similarities between the statements of the two fragments.

Clones are considered harmful in software maintenance and should be removed or detected at least. However, it would have been much better if there is no clone at all in the developed system so that we would not have to think about neither removal nor detection of clones. The idea is to use a clone detection tool in the normal development process to avoid cloning in the software right from the beginning. There are two ways of how to use a clone detection tool in the development process for avoiding clones. One way is the preventive control where a new function is added to the system only after being confirmed that this new function is not a clone to any existing one or there are specific reasons of adding that function as a clone to the system. The other way is the problem mining where any modification to a function must be consistently propagated to all of its similar functions in the system. Therefore, no clones are created unnecessarily, and the probability of update anomalies is reduced significantly(Roy & Cordy, 2007).

1.2 PROBLEM STATEMENT

Over the last decade many techniques and tools for software clone detection have been proposed. This includes textual approaches and semantic approaches. Most of them are oriented to a specific computer language and they range from high precision to low precision, and from high recall to low recall (El-Matarawy et al., 2013). There are five approaches that have been used in code clone detection which are text-based approach, token-based approach, tree-based approach, metrics-based detection approach and program dependency graph-based (PDG-based) detection approach. The figures below explain of all the code clone detection approach:

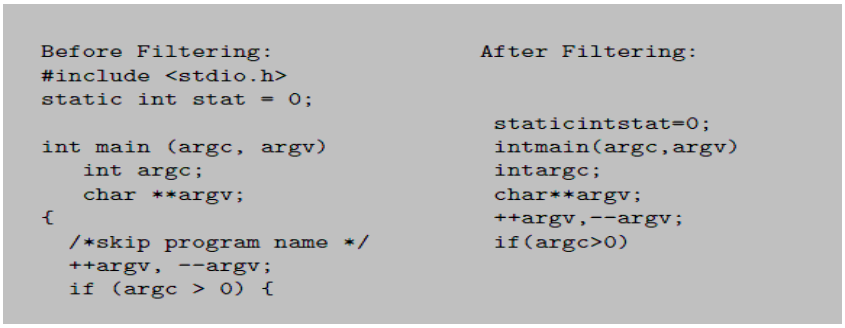


Figure 1.2: Example Output of Text-based Approach

The above figure shows text-based approach, two code fragments are compared with each other to find sequences of same strings. Once two or more code fragments are found to be similar in their maximum possible extent are returned as clone pair or clone class by the detection technique. Because of the purely text-based, detected clones do not correspond to structural elements of the language. A small or no normalization is performed on the source code before starting the actual comparison and most of the cases; the original source code is directly used in the clone detection process. However, to validate the following normalizations are applied on some approaches which are comments removal, whitespace removal and normalization of the code(Roy & Cordy, 2007).

In the token-based detection approach, the entire source system is transformed to a sequence of tokens. This sequence is then scanned for finding duplicated sub sequences of tokens and finally, the original code portions representing the duplicated sub sequences returned as clones. Compared to text-based approaches, a token-based approach is usually more robust against code changes such as formatting and spacing. Next, Tree-based approach is pared to a parse tree 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 Abstract Syntax Tree contains the complete information about the source code. The figure below shows the example of tree-based approach:

```
a[?] = x
```

occurs twice:

```
a[i] = x ;

a[i+1] = x ;
```

Figure 1.3: Example Output of tree-based approach

The above figure shows the argument to the first occurrence is lexical because it includes only a leaf and, perhaps, a unary node that identifies the type of the leaf. The argument to the second occurrence is, however, structural because it includes a binary based-tree node. Thus, it is clear that structural abstraction is more general than based-tree and hence, can find gapped clones by abstracting of a based-tree with the cost of much larger search space. Program Dependency Graph (PDG)-based approaches go one step further in obtaining a source code representation of high abstraction than other approaches by considering the semantic information of the source. PDG-based contains the control flow and data flow information of a program and hence carries semantic information.

Once a set of PDG-based are obtained from a subject program, isomorphic sub graph matching algorithm is applied for finding similar sub graphs which are returned as clones. Another approach of clone detection is metric-based approach. Metrics-based approaches gather different metrics for code fragments and compare these metrics vectors instead of comparing code directly. There are several clone detection techniques that use various software metrics for detecting similar code. First, a set of software metrics called fingerprinting functions are calculated for one or more syntactic units such as a class, a function, or a method or even statement and then the metrics values are compared to find clones over these syntactic units.

A clone detector must try to find pieces of code of high similarity in a system's source text. The main problem is that it is not known beforehand which

code fragments can be found multiple times. The detector thus essentially has to compare every possible fragment with every other possible fragment. Such comparison is very expensive from a computational point of view and thus, several measures are taken to reduce the domain of comparison before performing the actual comparison. Moreover, after finding the potential cloned fragments, further analysis and tool support is required to detect actual clones (Roy, Cordy, & Koschke, 2009).

1.3 RESEARCH OBJECTIVES

The main objectives to be achieved on this research are the following:

i. To propose a method in enhancing the generic code clone detection model.
ii. To implement the proposed method of enhancing the generic code clone detection model.
iii. To evaluate the result based on code clone type and run time performance of three applications.

1.4 RESEARCH QUESTIONS

There are several things that should be considered in achieving the objectives in this research:

i. What is the appropriate access modifier method to detect the type of code clone?
ii. How to implement the proposed access modifier method in generic code clone detection?
iii. How to evaluate the code clone detection result?

1.5 RESEARCH SIGNIFICANCES

There are a few of significances of this research:

i. The analysis of access modifier are useful as a medium to evaluate a huge of data and as a baseline reference for future research.
ii. The applied method of access modifier in detection of code clone can assist the expert in gaining the new program without duplication of code fragment.
iii. The applied method of access modifier in detection of code clone also can increase the confidence of the developer to create their own program without duplication of code fragments.
iv. Monitoring and removal of code clones are important in software development.

1.6 RESEARCH SCOPES

The boundary of the research as follows:

i. This research is focusing on workability of access modifier in detection of code clone.
ii. The datasets of the code is real codes that obtained from the existed code in library, internet sources or develop the code from programming tools for use in detection of code clone.
iii. Determine of applied method of access modifier in detection of code clone.
iv. The evaluation of the proposed method of access modifier in detection of the code clone.

1.7 RESEARCH ORGANIZATION

This research consists of five chapters. Chapter One which is introduction, that give briefly explanation to the readers about the research topic, related issues and objectives of this research. Next, chapter Two which is literature review, this part will explain in details about the selected topic, analyse the related work and make the comparison. Besides, this part also will explain the method that is suitable to be included in the research. Chapter Three which is methodology that will discuss about the research methodology, parameter of datasets, selected method, development technique and tools that used in this research. Then, we continue to the Chapter Four where the prototype is developed. From the prototype, we evaluate the result based on the code clone type and run time performance of the three application. Next, we move to Chapter Five which is the conclusion of the research. In conclusion, we conclude all the objective either it achieve or not achieve.

CHAPTER 2

LITERATURE REVIEW

2.1 INTRODUCTION

It is very common in computer programming to copy part of the program from one place and paste it in another place and then adapt it to fit in the new place. This happens for a variety of reasons. As a result, software systems often contain sections of code that are very similar, called code clones. Sometimes code clones are created for legitimate reasons, but other times they are not and they deteriorate the quality of the code. One of the main drawbacks of code clones is that the developer should modify multiple copies of the same pieces of code if a change is needed in a piece of code that has been cloned. Often this does not happen with good quality because the programmer forgets where they duplicated the code and leaves some clones unchanged. Fortunately, several techniques for detecting code clones have been proposed to help the programmer find code clones and locate the locations of duplicate code (El-Matarawy et al., 2013).

Code cloning is found to be a more serious problem in industrial software systems. In presence of clones, the normal functioning of the system may not be affected, but without countermeasures by the maintenance team, further development may become prohibitively expensive. Clones are believed to have a negative impact on evolution. Code clones may adversely affect the software systems' quality,

especially their maintainability and comprehensibility. For example, cloning increases the probability of update anomalies. If a bug is found in a code fragment, all of its similar cloned fragments should be detected to fix the bug in question. Moreover, too much cloning increases the system size and often indicates design problems such as missing inheritance or missing procedural abstraction. Although the cost of maintaining clones over a system's lifetime has not been estimated yet, it is at least agreed that the financial impact on maintenance is very high. The costs of changes carried out after delivery is estimated at 40% - 70% of the total costs during a system's lifetime (Latoza, 2005).

Existing research shows that a significant amount of code of a software system is cloned code and this amount may vary depending on the domain and origin of the software system. For instance, Baker has found that on large systems between 13% - 20% of source code can be cloned code. Lague et al. have studied only function clones and reported that between 6.4% - 7.5% of code is cloned code whereas Baxter et al. have reported that 12.7% of code being clones of a software system. Mayrand et al. have also estimated that normal industrial source code contains 5% – 20% of duplicated code. Kapser and Godfrey have experienced that as much as 10% –15% of source code of large system is cloned. For an object-oriented COBOL system, the rate of duplicated code is found even much higher, about 50% (Roy & Cordy, 2007).

Most previous work on code-clone detection has focused on finding identical clones, or clones that could be made identical via consistent transformations of identifiers and literals. However, code segments that are similar but not identical occur often in practice, and finding such non-identical clones can be as important as finding identical code segments. For example, while automated code compaction may require finding identical clones, studies of the evolution of a codebase over time require finding clones that vary in their similarity. One of the central issues with finding non-identical clones is assessing when two pieces of code are close enough to be considered "similar". Because this is likely to depend on the context in which the clone-detection clone detection tool is used, we believe that such tools should

provide a quantitative measure of clone similarity, leaving the ultimate decision of classification to the user of the tool (Smith & Horwitz, 2009).

2.2 IMPACT OF CODE CLONE

Code clone happened when the developer build a large software programme. While it is beneficial to practise cloning, code clones can have severe impacts on the quality, reusability and maintainability of a software system (Roy & Cordy, 2007). Several of impacts of code clone are listed as follows:

2.2.1 Increased Probability of Bug Propagation

On the off chance that a code portion contains a bug and that fragment is reused by adapting and gluing without or with minor adjustments, the bug of the first section may stay in all the glued fragments in the framework and accordingly, the likelihood of bug proliferation may increment fundamentally in the framework. Expanded of presenting another bug in numerous cases, just the structure of the copied piece is reused with the designer's obligation of adjusting the code to the present need. This procedure can be blunder inclined and may present new bugs in the framework (Kapser & Godfrey, 2008).

2.2.2 Increased Probability of Bad Design

Cloning may present terrible configuration, absence of good legacy structure or deliberation. Hence, it gets to be hard to reuse part of the usage in future tasks. It likewise severely effects on the viability of the product (Roy & Cordy, 2007; Roy et al., 2009).

2.2.3 Increased Difficulty in System improvement

As a result of copied code in the framework, one needs extra time and regard for comprehend the current cloned code and worries to be adjusted, and in this way,

it gets to be hard to include new functionalities in the framework, or even to change existing ones (Morshed, Rahman, & Ahmed, 2012).

2.2.4 Increased maintenance cost

If a cloned code segment is found to be contained a bug, all of its similar counterparts should be investigated for correcting the bug in question as there is no guarantee that this bug has been already eliminated from other similar parts at the time of reusing or during maintenance. Moreover, when maintaining or enhancing a piece of code, duplication multiplies the work to be done (Kapser & Godfrey, 2008).

2.2.5 Increased resource requirements

Code duplication introduces higher development rate of the framework size. Same time framework measure might not make a huge issue for a portion domains, others might require expensive equipment overhaul for a programming overhaul. Accumulation times will build in that's only the tip of the iceberg code need to a chance to be translated which need An adverse impact on the edit-compile-test cycle. The overall effect of cloning has been described by Johnson as a form of software aging or “hardening of the arteries” where even small changes on the architectural level become very difficult to achieve in the actual code (Kapser & Godfrey, 2008; Morshed et al., 2012; Smith & Horwitz, 2009).

2.3 CODE CLONE DEFINITION

Code clone definition is refer to the characteristics in code clone which is code clone have four types of detection. There are Type I, Type II, Type III and Type IV. Furthermore, the granularity in code clones detection which are clone pair and clone class.

Code clone Type I is define as identical code fragments except for variations in whitespaces, layout and comments. Refer to this journal, (Latoza, 2005) Type II of code clone is Structurally identical fragments except for variations in identifiers,

literals, types, whitespaces, layout and comments. Next, the code clone Type III is Copy and paste the code fragments with further modifications. Statements can be changed, added or removed in addition to variations in identifiers, literals, types, layout and comments (Prem, 2013). From this journal (Approach, n.d.), Type IV is defined that two or more code fragments that perform the same computation but implemented through different syntactic variants. These types of clones not only define an increasing level of subtlety from Type I through Type IV but also the analytical complexity and sophistication in detecting such clones increases from Type I through Type IV with Type IV being the highest. The detection of Type IV clones is the hardest even after having a great deal of background knowledge about the program construction and software design. This increasing level of analytical complexity from Type I through Type IV does not vary whether the process is automatic or not(Roy & Cordy, 2007).

The granularity of code clone which are clone pair and clone class. Clone pair is defined as a pair of code fragments is called a clone if there exists a clone-relation between them while clone class is the maximal set of code fragments in which any two of the code fragments hold a clone-relation (Roy & Cordy, 2007)

2.4 CODE CLONE DETECTION APPROACHES

Various clone detection techniques are presented in the literature. While a few of them are commercial, most of them are for research purposes aiming at assisting the development and maintenance processes (Van Rysselberghe & Demeyer, 2003).

2.4.1 Text-based Detection Approach

There are a few clone location strategies that depend on immaculate content based techniques. In this approach, the objective source system is considered as grouping of lines or strings. Two code pieces are contrasted with each other with discover groupings of 44 same strings. When two or more code pieces are observed

to be comparative in their most extreme conceivable degree are returned as clone match or clone class by the recognition system. Because of the purely text -based approach, detected clones do not correspond to structural elements of the language (Roy & Cordy, 2008). Below are the characteristics in code fragments that needed to evaluate the detection of code clone using text -based technique:

a) Comments Removal: Ignores all kinds of comments in the source code depending on the language of interest.
b) Whitespace Removal: Removes tabs, and new line and other blanks spaces.
c) Normalization: Some basic normalization can be applied on the source code

2.4.2 Token-based Detection Approach

In the token-based detection approach, the entire source system is transformed to a sequence of tokens. This sequence is then scanned for finding duplicated sub sequences of tokens and finally, the original code portions representing the duplicated subsequence returned as clones. Compared to text-based approaches, a token-based approach is usually more robust against code changes such as formatting and spacing (Roy et al., 2009).

2.4.3 Tree-based Detection Approach

In the tree-based approach a program is pared to a parse tree 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 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 (Baxter, Yahin, Moura, Sant'Anna, & Bier, 1998; Jia, Binkley, Harman, Krinke, & Matsushita, 2009; Roy et al., 2009).

2.4.4 Metrics-based Detection Approach

Metrics-based approaches gather different metrics for code fragments and compare these metrics vectors instead of comparing code directly. There are several clone detection techniques that use various software metrics for detecting similar code. First, a set of software metrics called fingerprinting functions are calculated for one or more syntactic units such as a class, a function, or a method or even statement and then the metrics values are compared to find clones over these syntactic units (Roy & Cordy, 2008; Van Rysselberghe & Demeyer, 2003).

2.4.5 Program Dependency Graph-based Detection Approach

Program Dependency Graph (PDG)-based approaches representation a source code with a high abstraction than other approaches by considering the semantic information of the source. PDG-based approach contains the control flow and data flow information of a program and hence carries semantic information. Once a set of PDG-based approaches are obtained from a subject program, isomorphic sub graph matching algorithm is applied for finding similar sub graphs which are returned as clones (Roy et al., 2009; Smith & Horwitz, 2009).

2.5 ACCESS MODIFIER

Access modifiers are keywords in object-oriented languages that set the accessibility of classes, methods, and other members. Access modifiers are a specific part of programming language syntax used to facilitate the encapsulation of components. If every member of every class and object were accessible to every other class and object then understanding, debugging, and maintaining programs would be an almost impossible task. The contracts presented by classes could not be relied on because any piece of code could directly access a field and change it in such a way as to violate the contract. One of the strengths of object-oriented programming is its support for encapsulation and data hiding. To achieve these we need a way to control who has access to what members of a class or interface, and

even to the class or interface itself. This control is specified with access modifiers on class, interface, and member declarations.

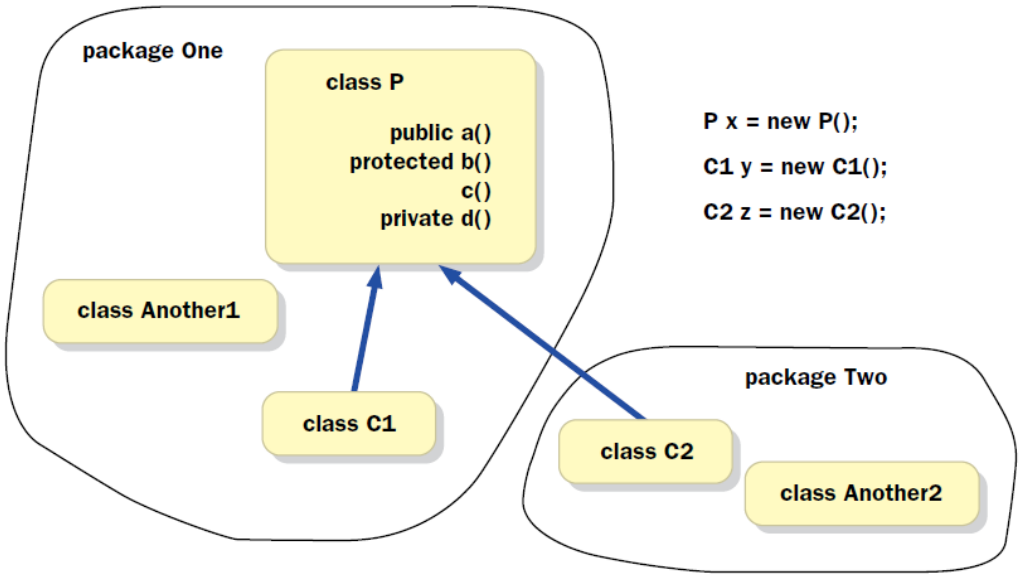


Figure 2.1: A situation demonstrating Java visibility modifiers

2.5.1 Public Access Modifier

A public class is publicly accessible. Anyone can declare references to objects of the classor access its public members. Without a modifier a class is only accessible within its own package (Arnold et al., 2005). Changing them can be impossible after that code relies on public or protected functionality. Package and private access are part of your implementation, hidden from outsiders. Below is the example of public access modifier:

```
1.  //save by A.java
2.
3.  package pack;
4.  public class A{
5.  public void msg(){System.out.println("Hello");}
6.  }

1.  //save by B.java
2.
3.  package mypack;
4.  import pack.*;
5.
6.  class B{
7.    public static void main(String args[]){
8.     A obj = new A();
9.     obj.msg();
10.   }
11. }
```

```
Output:Hello
```

Figure 2.2: Example of public access modifier (“Access Modifiers □,” n.d.)

2.5.2 Protected Access Modifier

Protected means it can be accessed by classes that extend that class, but that is loose language. More precisely, beyond being accessible within the class itself and to code within the same package, a protected member can also be accessed from a class through object references that are of at least the same type as the classthat is, references of the class's type or one its subtypes. An example will make this easier to understand (Arnold et al., 2005). Below is the example of protected access modifier:

```
1. //save by A.java
2.
3. package pack;
4. public class A{
5. protected void msg(){System.out.println("Hello");}
6. }
```

```
1. //save by B.java
2.
3. package mypack;
4. import pack.*;
5.
6. class B extends A{
7.   public static void main(String args[]){
8.    B obj = new B();
9.    obj.msg();
10.  }
11.}
```

```
Output:Hello
```

Figure 2.3: Example of Protected access modifier

2.5.3 No Access Modifier

No modifier is called as default bydefault. The default modifier is accessible only within package. Below is the example of no access modifier:

```
1.  //save by A.java
2.
3.  package pack;
4.  class A{
5.    void msg(){System.out.println("Hello");}
6.  }

1.  //save by B.java
2.
3.  package mypack;
4.  import pack.*;
5.
6.  class B{
7.    public static void main(String args[]){
8.     A obj = new A();//Compile Time Error
9.     obj.msg();//Compile Time Error
10.   }
11. }
```

Figure 2.4: Example of No access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

2.5.4 Private Access Modifier

Private Members declared private are accessible only in the class itself. apply only to members not to the classes or interfaces themselves. For a member to be accessible from a section of code in some class, the member's class must first be accessible from that code (Arnold et al., 2005). Below is the example of private access method.

```
1.  class A{
2.  private int data=40;
3.  private void msg(){System.out.println("Hello java");}
4.  }
5.
6.  public class Simple{
7.   public static void main(String args[]){
8.     A obj=new A();
9.     System.out.println(obj.data);//Compile Time Error
10.    obj.msg();//Compile Time Error
11.    }
12. }
```

Figure 2.5: Example of Private access modifier

Based on the above figure, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

2.5.5 The Differences of characteristics of Access Modifier Methods

There are several difference characteristics of using the methods of access modifier. Table 2.1 below shows the advantages and disadvantages of all of access modifier methods.

Table 2.1: The difference of characteristics of Access Modifier Methods

Access Modifier	Within class	Within package	Outside package by subclass only	Outside package
1. Public	Yes	Yes	Yes	Yes
2. Protected	Yes	Yes	Yes	No
3. No modifier	Yes	Yes	No	No
4. Private	Yes	No	No	No

2.6 RELATED WORK

Code clone detection model is used detect code clones by using the proposed access modifier rule and weightage to have the significant result. Therefore, model was proposed to have unified code clone detection and results. There are three code clone models used in code clone research domain. The models are generic clone model, generic pipeline model and unified clone model.

2.6.1 Generic Clone Model

The generic clone model is a model that defines the clones that exist in a program. The model has a division of concerns on clone detection, description and management using layers. The main function of this model is to describe the clones. Furthermore, the advantage of this model is can reduces the effort in the implementation of tools supporting these activities. The figure of overview of the generic clone model is shown below (Giesecke, 2007).

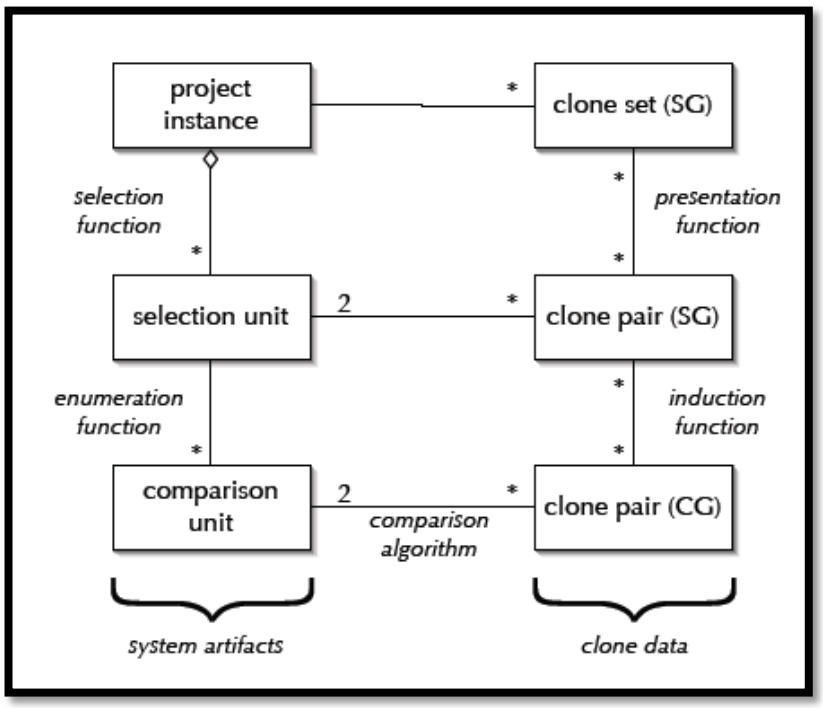


Figure 2.6: The overview of generic clone model (Giesecke, 2007)

There are two type of elements linked to the model which are elements correspond to system artifacts and artefacts representing the part of the clone data that is generated by a clone detection algorithm based on the system artifacts. Project instance is the highest level representation of the model. An instance is structured into selection units and comparison unit through a selection function and an enumeration function. This instance is known as clone data. The clone pairs exist in two granulities that are the selected units and the comparison units. Clone pairs are grouped into clone sets by a presentation function. The clone sets used here are sets with a distinguished reference element to reduce redundancy in a clone set.

The implementation of this model is done as plugin to eclipse. Eclipse is an integrated development environment (IDE) that is used for development standalone programs and web applications. In addition, this model has a clear separation of clone detection process definition using layers. Moreover, this model is more focused on management of code clone that was driven by operational aspects of code clone detection and removal. It is only available only as a plugin for IDE rather than a separated code clone detection tool.

2.6.2 Generic Pipeline Model

A Generic Pipeline Model is a combination of processes to detect code clone with all the necessary steps in a code clone detection process (Thesis et al., 2015). There are five processes involved in this generic pipeline model which is shown in the figure below.

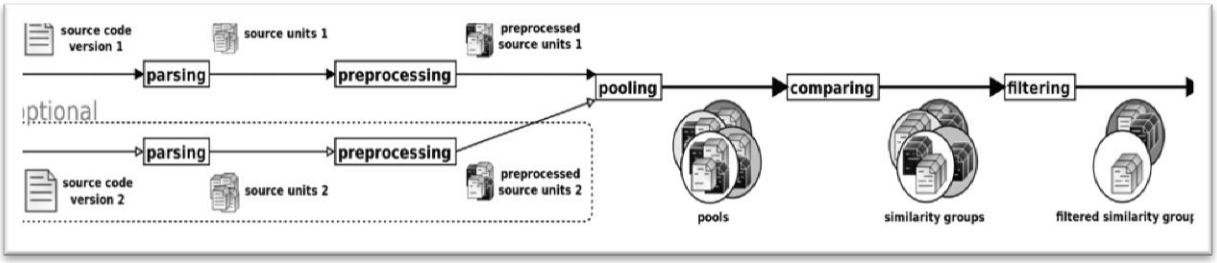


Figure 2.7: An overview of process in Generic Pipeline Model

The first process is parsing process and its transforms source code into source units. This process revolves in transforming source codes into source units. Representations of source units uses sub trees of an Abstract Syntax Tree. The input of this source file and the output of this process is the source units. Next, the source units are used as input for the second process of the generic pipeline model. Pre-processing process is to normalize the source units and to add additional annotations to the source units. Normalization turns the source units into a regular form and makes different source units more similar. It uses Abstract Syntax Tree as input and pre-processed Abstract Syntax Tree as output. It is implemented using several-cascaded processor. The output of this model is the pre-processed source files. The pre-processed source files then proceed to the third process which is pooling process.

Pooling process is the process of grouping pre-processed Abstract Syntax Tree source units into sets of groups according to defined characteristics based on criteria set by the user. The output of this model process is pools. Then comparing process is come after the pooling process. Comparing process is a recursive comparison process of source units in all pools using a division rules. The output of this process is clone similarity groups. These groups are then used as input for the final input which is the filtering process. The function of filtering process is to remove irrelevant clone candidate sets from the result set.

The use of the generic pipeline model is created by Java Code Clone Detector. It is a code clone detection tool designed and developed to detect code clones in Java.

2.6.3 Unified Clone Model

Unified Clone Model is a generic model that can represent all the results of all code clone tools (Harder, 2013). Below is the figure shows the flow of the Unified Clone Model.

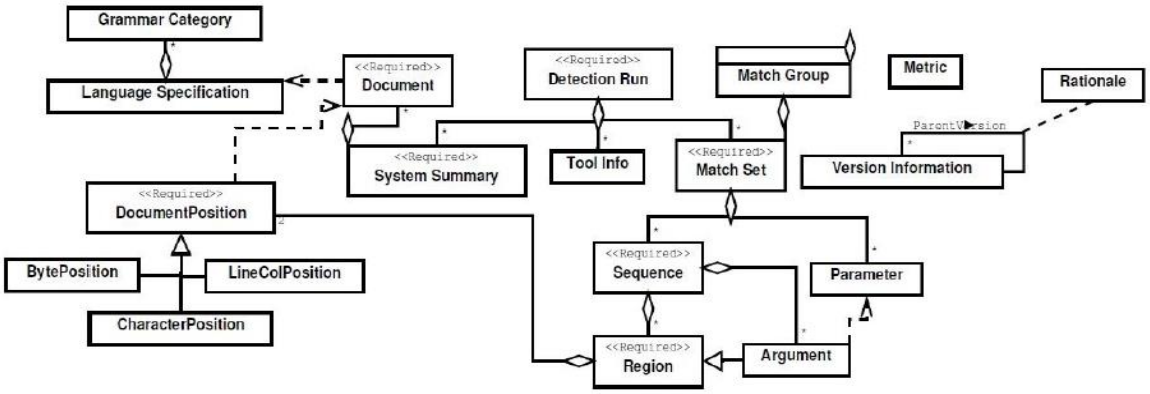


Figure 2.8: An overview of the flow of Unified Clone Model

The figure 2.8 above is still in design phase. It was designed through different clone representations of existing tool. As a concept analysis, uses case from eleven applications has been used. The outcome of the analysis has been divided into four groups which are detection for clone detection techniques. The disadvantage of this model is the model is still in design phase and it is lacks of a proper file format for data representation.

2.6.4 Advantages and Disadvantages of Models

Every model has its own advantages and disadvantages. Table 2.2 shows the Strength Weakness Opportunity Threat (SWOT) analysis of the models.

Table 2.2: SWOT analysis of the models

Feature	Generic Clone Model	Generic Pipeline Model	Unified Clone Model
Strength	This model has clear separation of clone detection process definition using layers which makes description of the clones possible.	This model consists of step by step process to detect clones in Java applications. It allows customization for the user to manipulate the model.	This model is created through the different clone representations of existing tools.
Weakness	Generic clone model not allow manipulation on it layers to extend the effectiveness of this model.	The extension of this model is limited because of the manipulation on the pre-defined sets and rules in the model.	This model is still in design phase and lacks of proper file format for the data representation.
Opportunity	The description of the generic clone model can be improved.	The clone type detection and the process can be improved to get the better code clone detection result.	The realization of the model using user defined process.
Threat	The implementation of the changes of the model is impossible as its nature of being a plugin.	The application used for evaluation will have different results compared to existing work.	Different tools might cause variation to the end results.

Based on the SWOT analysis above, the basic weakness of the model is the extendibility of the existing models. The generic clone model is not allowed the manipulation on its pre-defined layers to extend the effectiveness of its model while the generic pipeline model only allows the manipulation within its process so its limit the extendibility of the model to enhance the code clone detection. The realization of the model is another major weakness of the generic pipeline model. It is very different with the struggles in realizing the prototype or tools for the unified clone model.

2.7 SUMMARY

The conclusion of Chapter 2 is we have review the code clone area which is what is the disadvantages of code clone, the definition of code clone, and the all types of code clone. Then, we review about what access modifier is and how to practice in the research with describe the properties of the every access modifier. Moreover, we do the review of the models of clone include their advantages and the disadvantages of every models. From all the methods of access modifier, we only chose one method to be used in the code clone detection model which is protected access modifier. We moved to next chapter which is Chapter 3, the methodology in details of developing the model design.

CHAPTER 3

METHODOLOGY

3.1 INTRODUCTION

In this chapter, it represents all the necessary information. The chapter outlines the details about the research undertaken to address the question posed in chapter one. It explores the research question in more depth and discusses what method are the most appropriate, given the aims and nature of the research. Consequently, the content of this chapter are methodology and dataset of the method used in enhancing generic code clone detection model through the protected access modifier rule and weightage. In this chapter, the explanation of the methodologies is explained.

Dataset is discussing briefly about the parameter that we are going to use in this research. Besides, the technique that we are going to use in improving code clone detection is protected access modifier. This technique already briefly discussed in chapter two. Furthermore, the development tools is describe about the tools that we used during this research and justifying the importance of hardware and software chosen. Then, the Gantt chart is draw using appropriate tools.

3.2 RESEARCH METHODOLOGY

Figure 3.1 presents the overview of enhancing generic code clone detection model through the protected access modifier rule and weightage. The methodology is divided into three phases which are the literature review and analysis, design and develop the model and evaluation phase.

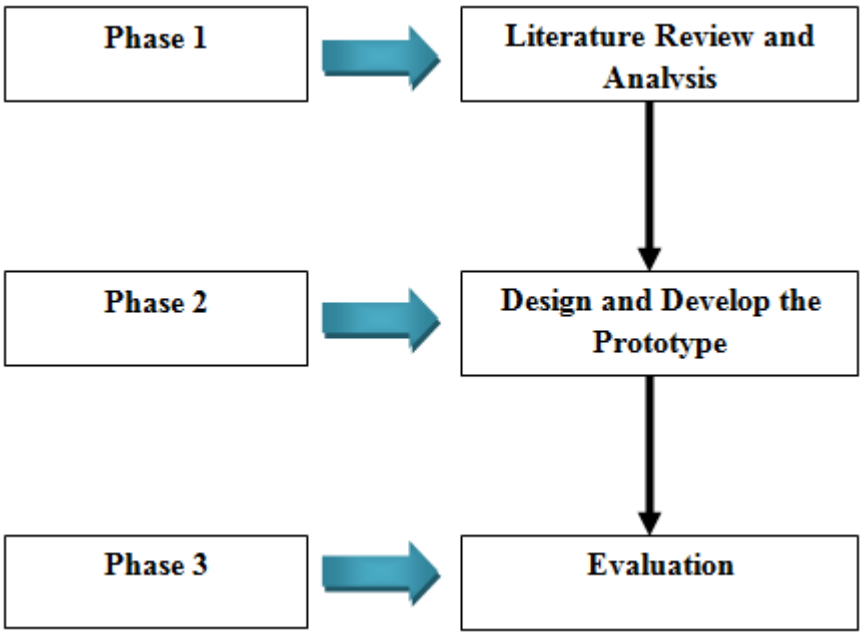


Figure 3.1: Research Methodology

3.2.1 Literature Analysis

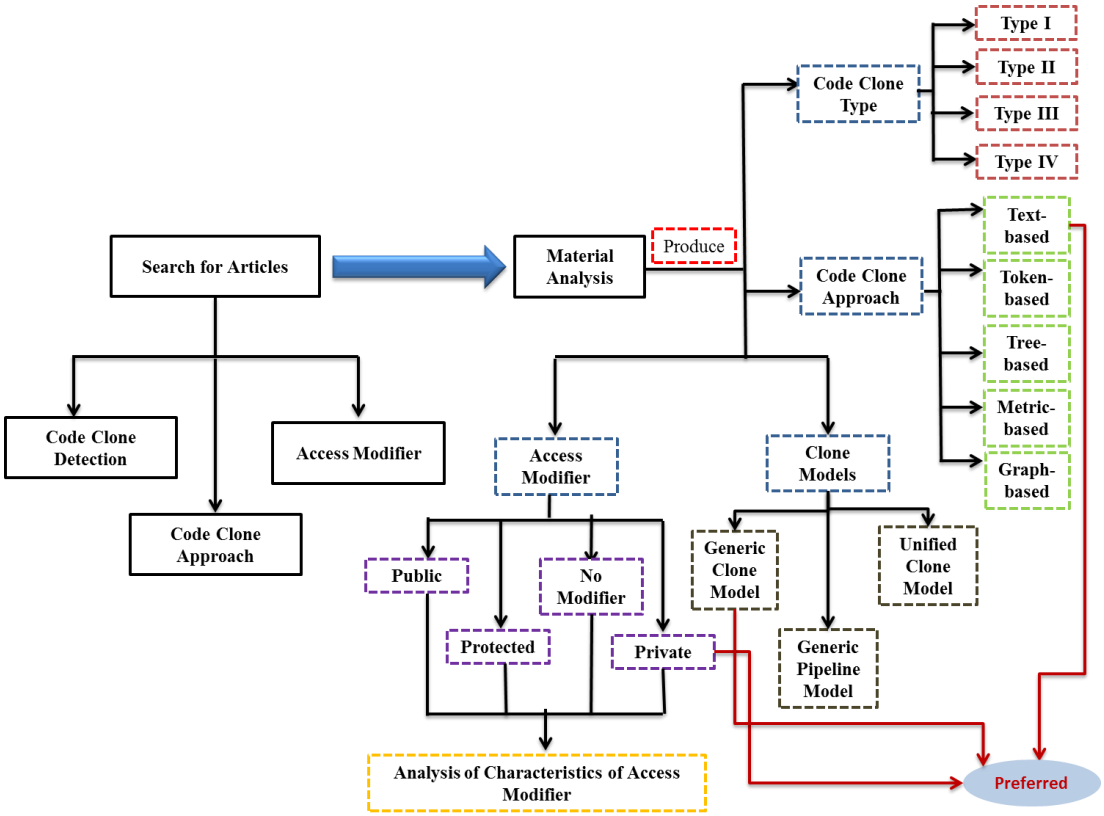


Figure 3.2: Literature Review and Analysis

The figure 3.2 above is the summary of literature review. During this phase, a study on code clone and access modifier field is conducted. Consequently, all the related literature papers and journals are gathered and reviewed in order to analyse the issues and challenges in code clone detection. We have concluded the definition of code clone, the approach of code clone and the whole of methods of access modifier. However, we are focusing more to code clone detection. All of them are difficult to diagnose but access modifier is the most challenged. Therefore, enhancing the generic code clone detection model through modifier access is selected to be studied. The clone model to be used in code clone detection is identified.

Furthermore, data related to the code clone detection is obtained from a relevant source which is can act as a real data. After selected the field of studied, the related technique are appointed to be implement in code clone detection. There are a few relevant access modifier are analysed such as public access modifier, protected access modifier, no access modifier and private modifier. However, protected access modifier would be the best methods to be used for enhancing the code clone detection due to the ability in identify the ambiguous data. Hereby, the problem statements, objectives, questions, significant and scope are identified in this phase.

3.2.2 Design and Develop Model

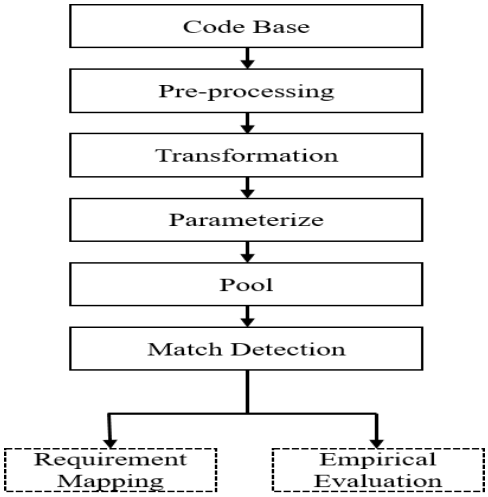


Figure 3.3: Design of the Generic Code Clone Model

From the figure 3.3 above, this phase consists of four main activities which are requirement design and analysis, model design, model implementation and model testing. During requirement design and analysis, data is composed from related references and be analysed. The data is the further analysed based on the protected access modifier rule and weightage. Apart from it, functional and non-functional requirements for the model also are determined during this activity. After getting and analysing the requirement needed, the system design is begin.

The rule and weightage of the protected access modifier is applied in the match detection process. This is the final process in the Generic Code Clone Detection Model. The objective of this process is to detect the code clone. The input of this process is the pairs and groups of source units based on three categories. The match detection process uses a hybrid detection technique of exact matching with Euclidean distance. As mentioned before, there are three pools obtained from the previous process. The match detection starts by finding the exact clone or better known as Type I clone; and near exact clones or better known as Type II clone. Therefore, there are two stage of exact matching being used to detect Type I and Type II clones.

The first stage is the exact matching technique is used in detecting the same average ratio value of both source units in the first pool. The compared functions that have the same average ratio value of both source units are detected as Type I. The second stage is the exact matching technique is used in detecting the same average ratio source units value is difference so the clones that are detected through this stage are known as Type II. The remaining average ratio source units value from first and second will be combined for the next step of this process.

As for the remaining average ratio source units or classifie as header and body value, Euclidean distance is applied. Assume there are two source units which are *A* and *B*. Therefore, the Euclidean distance, *ED*, between *A* and *B* calculated as:

$EDAB = \sqrt{(headerA - headerB)2 + (bodyA - bodyB)2}$

where,

EDAB is Euclidean distance of Function A and Function B

headerA = average ratio *header* of *A*

bodyA = average ratio *body* of *A*

headerB = average ratio *header* of *B*

bodyB = average ratio *body* of *B*

The calculation of the Euclidean distance is applied among the remaining average ratio *header* and *body* values in the third process. Once the calculation is done, it is the function is then grouped to Type III and Type IV based on the distance obtained. Type III clone are taken from the range of 85% - 100% while the remaining is defined as Type IV.

Consequently, the model is designed. Planning is the first design process, in this phase, we should plan the design of the interface and how it can interact with the user. Then analyses the user experience factors before develop the conceptual design. If have any changes, refinement process will occur. The figure 3.4 shows the design of the model while figure 3.5 shows the code will be insert in the tools and figure 3.6 shows the report that will be execute from NetBeans IDE 8.1:

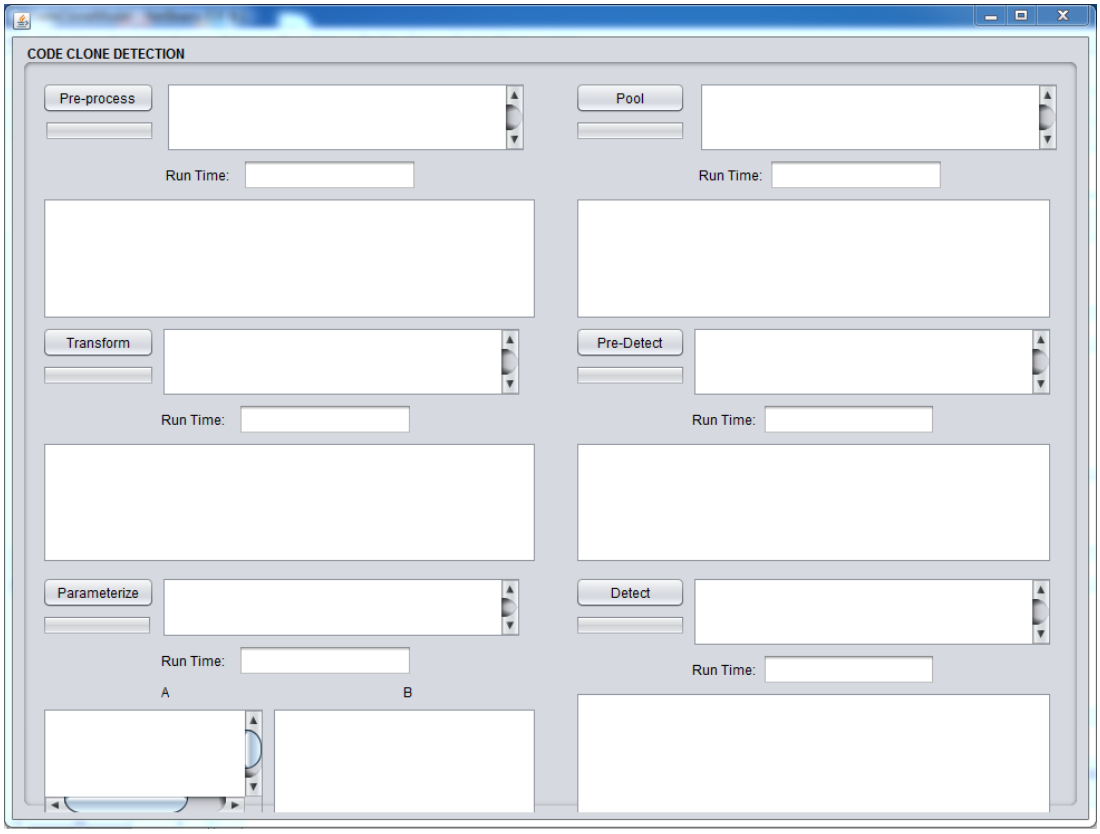


Figure 3.4: The interface design of the generic code clone model

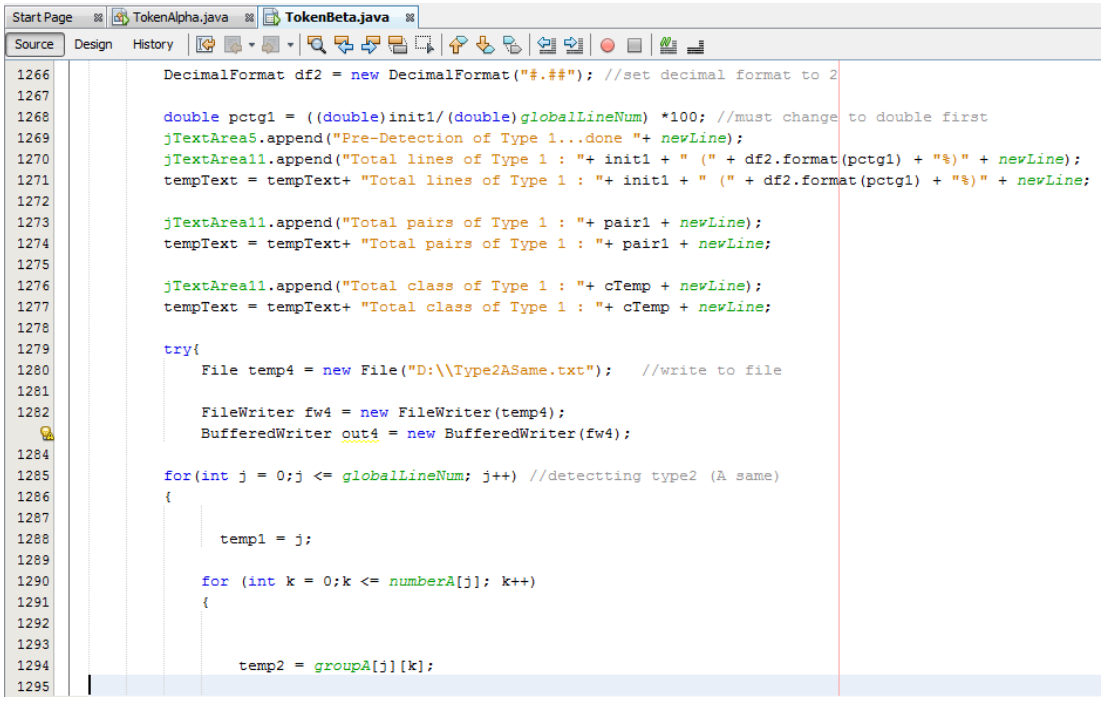


Figure 3.5: Example of code built in java

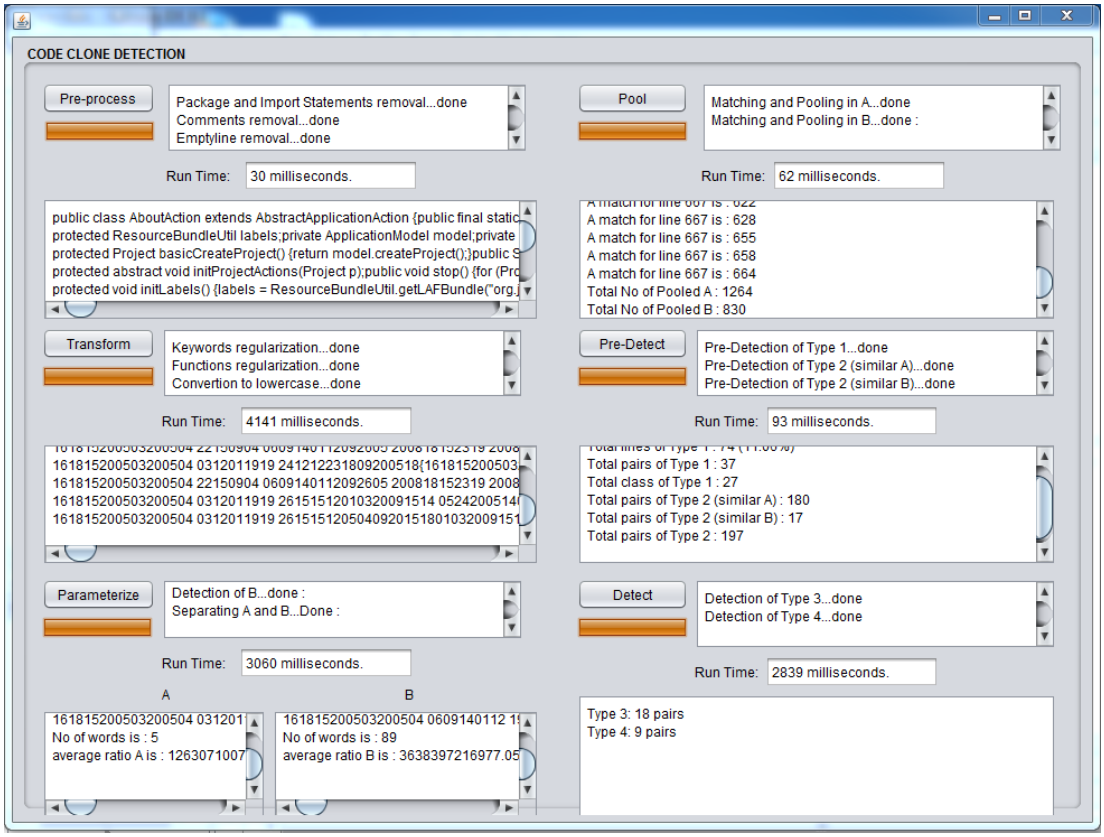


Figure 3.6: The sample of code clone detection report

Then, the interface of model will be developed. Next activity is the model implementation. The protected access modifier rule and weightage will be conduct during this phase. There are considering several steps to be define which linguistic variables, construct membership function are, construct knowledge of the rule and evaluate the result. After we proposed the method will be uses in the Generic Code Clone Detection Model, we suggest the rule and the weightage to evaluate the better result.

Besides, during this phase, Java code will be acts as a programming language to coding the models using NetBeans IDE 8.1 software. Lastly, model testing is conducted to test the system functionality whether the model achieve the main goal of this research. All this steps are will be implementing for the next semester.

3.2.3 Evaluation

Evaluation phase is the last phase in this research methodology. This phase is conducted based on the verification and validation process. Fundamentals (2011) stated that, verification is the process of evaluating work product of the system development to ensure we are on right track of creating the final product while validation is the process of evaluating the final product to check whether the software is satisfies the specific requirement. On the other hand, verification is conducted by comparing two different datasets. The datasets is divided into two, which are real data and synthetic data. Real data is obtained from the related references while synthetic data is gained from the expert opinion. Then, the results of these two datasets will be compared.

For validation, we are mapping the requirement to model to evaluate the result based on code clone type and run time performance of three applications. The evaluation will be carried out by design the model and explanation of the finished model.

3.3 Data Set

The data set for this project is taken from the three tools which are JHotDraw 7.0.6, SableCC 3.7 and ANTLR 3.0.1. JHotDraw tool is used to the code fragment that needs to use the graphic user interface application. For SableCC and ANTLR tools, they will be used for parses generators (Ishio, Date, Miyake, & Inoue, 2008). Table 3.1 below shows the length of code and the total of files belong to the three tools:

Table 3.1: The Characteristics of the Tools

Name	Version	Length of Code	Total of File
JhotDraw	7.0.6	90166	309
SableCC	3.7	35388	196
ANTLR	3.0.1	59687	522

3.4 Hardware and Software

This section covers the hardware and software requirement needed to develop and design the model of the code clone detection. The hardware tools used should be convenient with the development of the model.

3.4.1 Hardware Development

The hardware requirement and their purposed for this project is shown in Table 3.2.

Table 3.2: Hardware Development

Hardware	Purpose
Acer Aspire 4752Z	Device to develop the prototype
Printer	To print sheets and documents

3.4.2 Software Development

The software requirement and their purposed for this project is shown in Table 3.3.

Table 3.3: Software Development

Software	Purpose
Windows 7	Platform of operating system
Microsoft Word 2010	Prepare proposal and documentations
Microsoft Visio 2010	Create Gantt Chart
Microsoft Power Point 2010	Design and draw the diagrams
NetBeans IDE 8.1	Develop the model
Mendeley Desktop	Manage reference and citation

3.5 Gantt Chart

The Gantt chart shows the estimate duration from the start of the project until the end of the project (refer to Appendix A1).

3.6 SUMMARY

The conclusion of Chapter 3 is we have reached the methodology to develop the model of the research with determined the methods of access modifier that use in the code clone detection. The protected access modifier is chose to be used in enhancing the code clone detection model. Therefore, we have to develop the model through follow the model design in development part of Gantt chart.

CHAPTER 4

EVALUATION

4.1 INTRODUCTION

In this chapter, it presents all the evaluations of the research by apply the preferred method. The chapter outlines the output data about the research undertaken to address the objectives posed in chapter one. It explores the research objectives in more depth. Consequently, the content of this chapter are evaluation of the datasets that used in enhancing generic code clone detection model through protected access modifier rule and weightage. In this chapter, the evaluation of the output is explained.

4.2 MODEL EVALUATION

This section shows the code clone detection result by enhancing the generic code clone detection model through protected access modifier rule and weightage.

4.3 CLONE PAIR DETECTION

Table 4.1 shows the overall result of the detected clone by enhancing generic code clone model through protected access modifier rule and weightage for JHotDraw 7.0.6, SableCC 3.7 and ANTLR 3.01.

Table 4.1: Result of the detected code clone

Application	Type I	Type II	Type III	Type IV
JHotDraw 7.0.6	37	197	18	9
SableCC 3.7	3	0	0	2
ANTLR 3.0.1	87	212	58	44

From the table above, we can conclude that all of the code clone type is detected in the chosen application. In JHotDraw application, there is 37 match detection of type I, 197 match detection of type II, 18 match detection of type III and 9 match detection of type IV. For SableCC application, there is 3 match detection of type I while it does not have type II and type III of the code clone then it has 2 match detection of type 4. In addition, ANTLR have 87 match detection of type I, 212 match detection for type II, 58 match detection for type III and 44 match detection for type IV. ANTLR is the most have the significant of all the type of code clone as ANTLR have 522 java file than JHotDraw application only have 309 java file while SableCC have the least java file which is only have 196 java file.

4.4 OVERALL RUNTIME PERFORMANCE

Table 4.2 and figure 4.1 shows the overall run time performance for data evaluation of this research.

Table 4.2: The evaluation of the overall run time performance of code clone detection model

Application Process	JHotDraw	SableCC	ANTLR
Pre-process (ms)	30	204	109
Transform (ms)	4141	81461	12336
Parameterize (ms)	3060	609	5651
Pool (ms)	62	0	73
Pre-detect (ms)	93	218	141
Detect (ms)	2839	16	6162

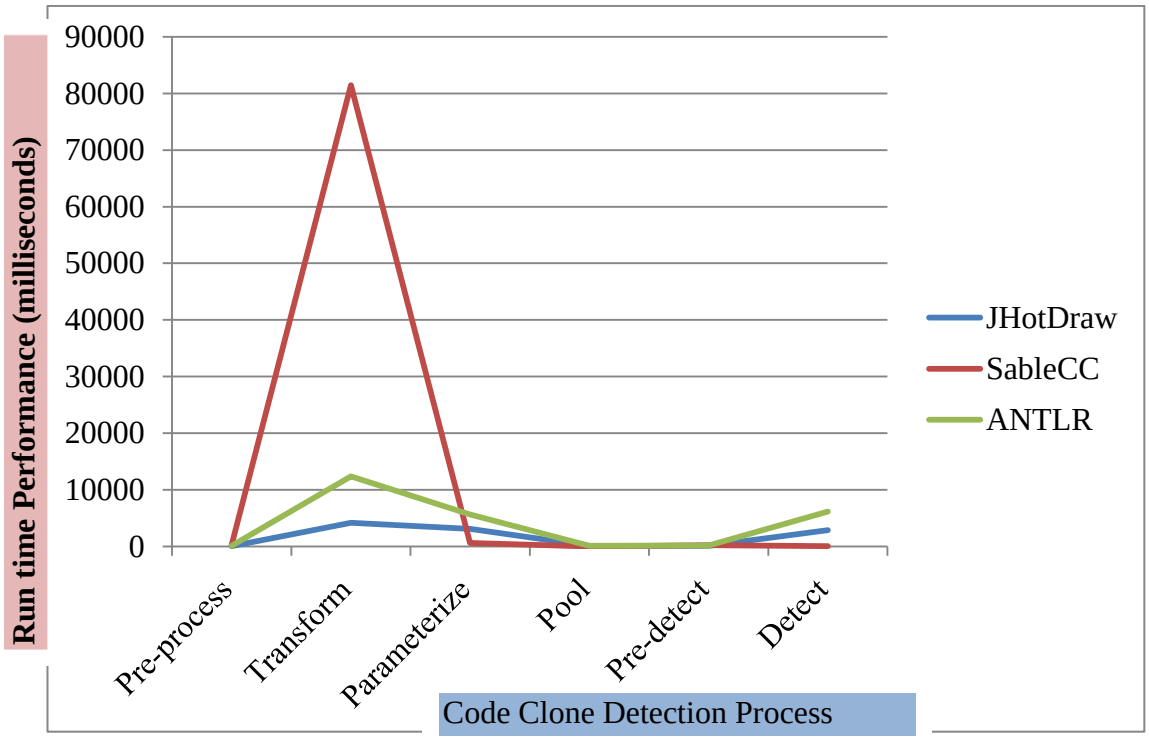


Figure 4.1: Line graphs for shows the overall run time performance of Generic Code Clone detection model

Based on the table 4.2 and the figure 4.1, the transform process has the highest runtime performance. The match detection which are combination of detect and pre-detect process has the second highest runtime while the parameterize process has the third highest runtime performance. The pool process which is categorization

process has the lowest runtime performance while the pre-processing process has the second lowest runtime performance.

Based on the comparison diagrammatically shown in Table 4.2 and Figure 4.1, the two highest runtime performance has been recorded by the transformation and parameterization process. It is essential that the transformation process to have a high runtime due to the transformation of the source codes into numerical form. There are a lot of source code that needs to be transformed. Therefore, a large Java applications that have a lot of line of codes such as J2sdk1.4.0-javax-swing takes a lentgh amount of time to be transformed. It essential for the match detection process to have a high runtime performance due to the task of the process in detecting all the code clone types.

4.5 SUMMARY

Chapter 4 is a discussion on evaluation of the result in enhancing generic code clone detection model through protected access modifier rule and weightage where the result is taken from the prototype. The discussion of the result is explained based on the result taken.

CHAPTER 5

CONCLUSION

5.1 CONCLUSION

In conclusion, protected access modifier rule and weightage is purposely use for enhancing the generic code clone detection model in order to analyse its efficiency classifying of the code clone type. This study also displays the analysis of accuracy so that we can know how accurate this algorithm to enhance the generic code clone detection model.

In this thesis, all the three main objectives stated in the early of the project development are achieved successfully as mention below.

i. To propose a method in enhancing the generic code clone detection model. This objective has been achieved and discuss at Chapter Three since we approve that framework over there.

ii. To implement the proposed method of enhancing the generic code clone detection model. This objective has been achieved and discuss at chapter Four since we approve that framework over there.

iii. To evaluate the result based on code clone type and run time performance of the three applications. As we can see after experiment data was handled, we can conclude that the total of code clone type and the run time performance have the significant result to be observed.

5.2 LESSON LEARNT

Throughout the project period, I learnt a lot of things. In terms of project planning, I believe that having a proper project milestone is a crucial criterion for any successful project. Milestone should be realistic and achievable on time. Delay on milestone will cause the whole project to be done later than expected. Thus, it is essential to always stick to the milestone and continuously check the next coming milestone to get done on time.

Secondly, getting the right idea on what to be done is also important. Never assume ideas or opinion without any proofs. Be critical and inventive when dealing with ideas. This is because any research always requires a formula or algorithm to prove whether it is right or wrong. Research without any critical analysis will ruin the research result as a whole.

Lastly, I learnt that consistency and self-explorative are two compulsory behaviour to get the research done on time. One, being consistent will ensure that I will never delay my work, and always stick with the milestone that was scheduled. Being self-explorative helps me to understand my research subject faster.

5.3 RESEARCH LIMITATIONS

While completing this research, there are lots of limitations that need to be handled in order to have a significant data and to achieve all the research objectives. Below are the limitations that we need to handle in this research:

i. Limited Time

To have a significant output and to achieve the all of the objectives, the long period of time is needed as we need to finish every chapter by define the methods and tools to have the significant output.

ii. Limited knowledge

This research need a lot of knowledge about the definition of code clone, the impact of the code clone, the approach of the code clone and etc. We need a lot of reference based on the code clone methods and tools so that we can decide the methodology to achieve the good result.

5.4 FUTURE WORK

The future work is focus on:

i. Improve the code clone detection process and performance can be done through improvement of the pre-processing process in supporting code clone detection in other structural and procedural programming language so generic code clone detection can support code clone detection in other programming languages too.

ii. Build a more dynamic view of the code clone detection result through visualization methods and the utilization of parallel algorithms in improving the runtime performance.

5.5 SUMMARY

Chapter 5 is a discussion on conclusion of the project research in enhancing generic code clone detection model through protected access modifier rule and weightage and its future work.

REFERENCES

Approach, A. N. E. W. (n.d.). Code Clone Detection. Retrieved from http://mondego.ics.uci.edu/projects/clonedetection/

Arnold, B. K., Gosling, J., Holmes, D., Arnold, B. K., Gosling, J., & Holmes, D. (2005). *No Title*.

Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M., & Bier, L. (1998). Clone detection using abstract syntax trees. *Proceedings International Conference on Software Maintenance Cat No 98CB36272*, *98*, 368–377. http://doi.org/10.1109/ICSM.1998.738528

El-Matarawy, A., El-Ramly, M., & Bahgat, R. (2013). Parallel and Distributed Code Clone Detection using Sequential Pattern Mining. *International Journal of Computer Applications*, *62*(10), 975–8887. http://doi.org/10.5120/ijca2015906324

Giesecke, S. (2007). Generic modelling of code clones. *Duplication, Redundancy, and Similarity in Software*, (6301), 1–23. Retrieved from http://drops.dagstuhl.de/opus/volltexte/2007/960

Harder, J. (2013). The limits of clone model standardization. *2013 7th International Workshop on Software Clones, IWSC 2013 - Proceedings*, 10–11. http://doi.org/10.1109/IWSC.2013.6613034

Ishio, T., Date, H., Miyake, T., & Inoue, K. (2008). Mining coding patterns to detect crosscutting concerns in Java programs. *Proceedings - Working Conference on Reverse Engineering, WCRE*, 123–132. http://doi.org/10.1109/WCRE.2008.28

Jia, Y., Binkley, D., Harman, M., Krinke, J., & Matsushita, M. (2009). KClone: a proposed approach to fast precise code clone detection. *Third International Workshop on Detection of Software Clones (IWSC)*.

Kapser, C. J., & Godfrey, M. W. (2008). "cloning considered harmful" considered harmful: Patterns of cloning in software. *Empirical Software Engineering*, *13*(6), 645–692. http://doi.org/10.1007/s10664-008-9076-6

Latoza, T. (2005). A Literature Review of Clone Detection Analysis.

Morshed, M., Rahman, M., & Ahmed, S. (2012). A Literature Review of Code Clone Analysis to Improve Software Maintenance Process. *arXiv Preprint arXiv:1205.5615*. Retrieved from http://arxiv.org/abs/1205.5615

Prem, P. (2013). A Review on Code Clone Analysis and Code Clone Detection, *2*(12), 43–46.

Roy, C. K., & Cordy, J. R. (2007). A Survey on Software Clone Detection Research. *Queen's School of Computing TR*, *115*, 115. http://doi.org/10.1.1.62.7869

Roy, C. K., & Cordy, J. R. (2008). Scenario-based comparison of clone detection techniques. *IEEE International Conference on Program Comprehension*, 153–162. http://doi.org/10.1109/ICPC.2008.42

Roy, C. K., Cordy, J. R., & Koschke, R. (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, *74*(7), 470–495. http://doi.org/10.1016/j.scico.2009.02.007

Smith, R., & Horwitz, S. (2009). Detecting and Measuring Similarity in Code Clones. *Workshop Proceedings of the 13th European Conference on Software Maintenance and Reengineering*, 28–34.

Thesis, D. O. F., Shahrizal, A., Muhamad, B. I. N., Scheduling, R. P., Access, O., Supervisor, S. O. F., & Supervisor, N. O. F. (2015). " I hereby declare that I have read this thesis and in my opinion this thesis is sufficient in terms of scope and quality for the award of the degree of Doctor of Philosophy (Computer Science)." Signature Principal Supervisor : Prof . Dr . Safaai bin De,

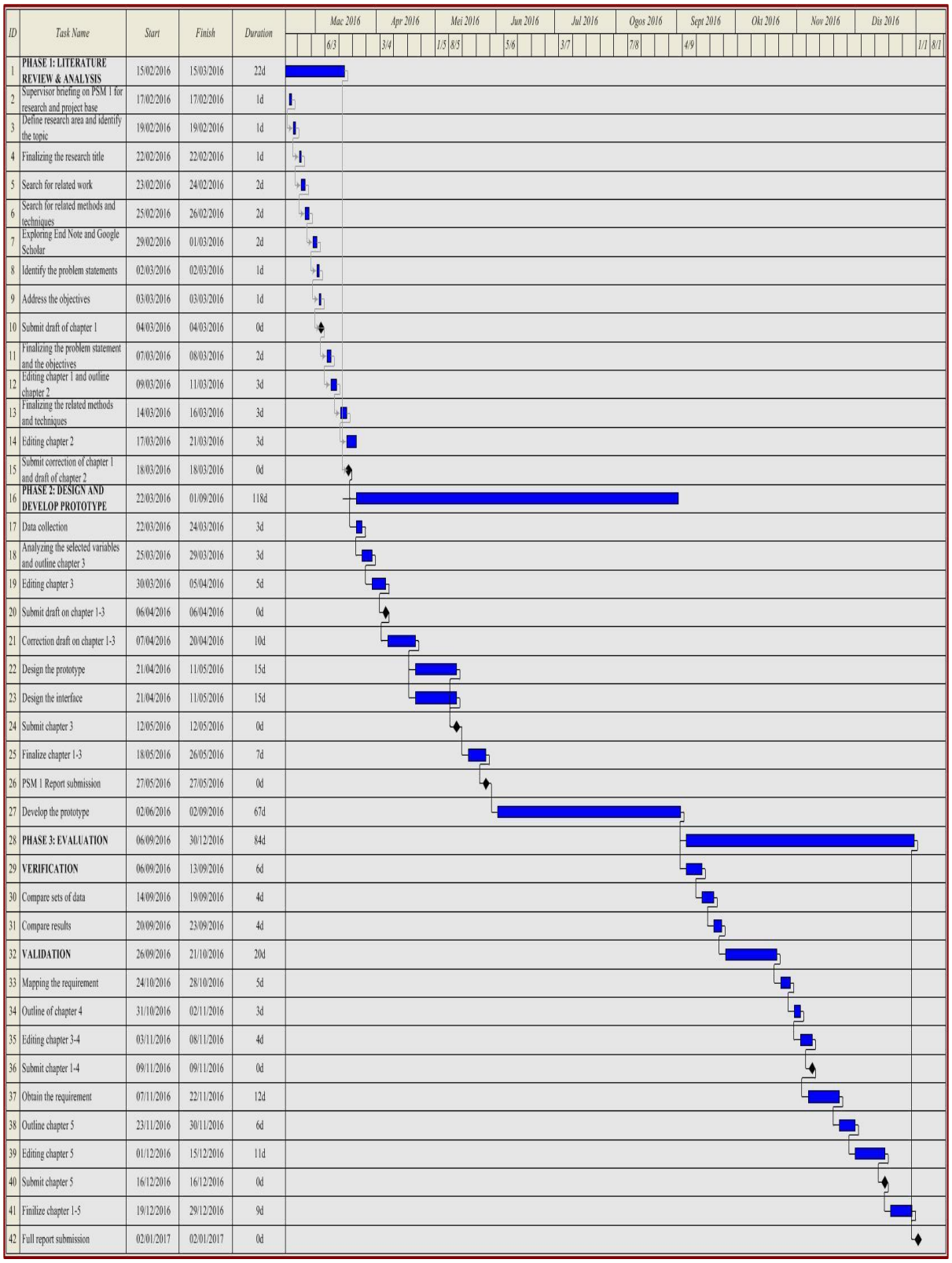
16(August).

Van Rysselberghe, F., & Demeyer, S. (2003). Evaluating Clone Detection Techniques. *Evolution of Large-Scale Industrial Software Applications (ELISA)*, (i), 1–12. http://doi.org/10.1109/ASE.2004.1342759

Yuan, Y., & Guo, Y. (2011). CMCD: Count matrix based code clone detection. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, 250–257. http://doi.org/10.1109/APSEC.2011.13

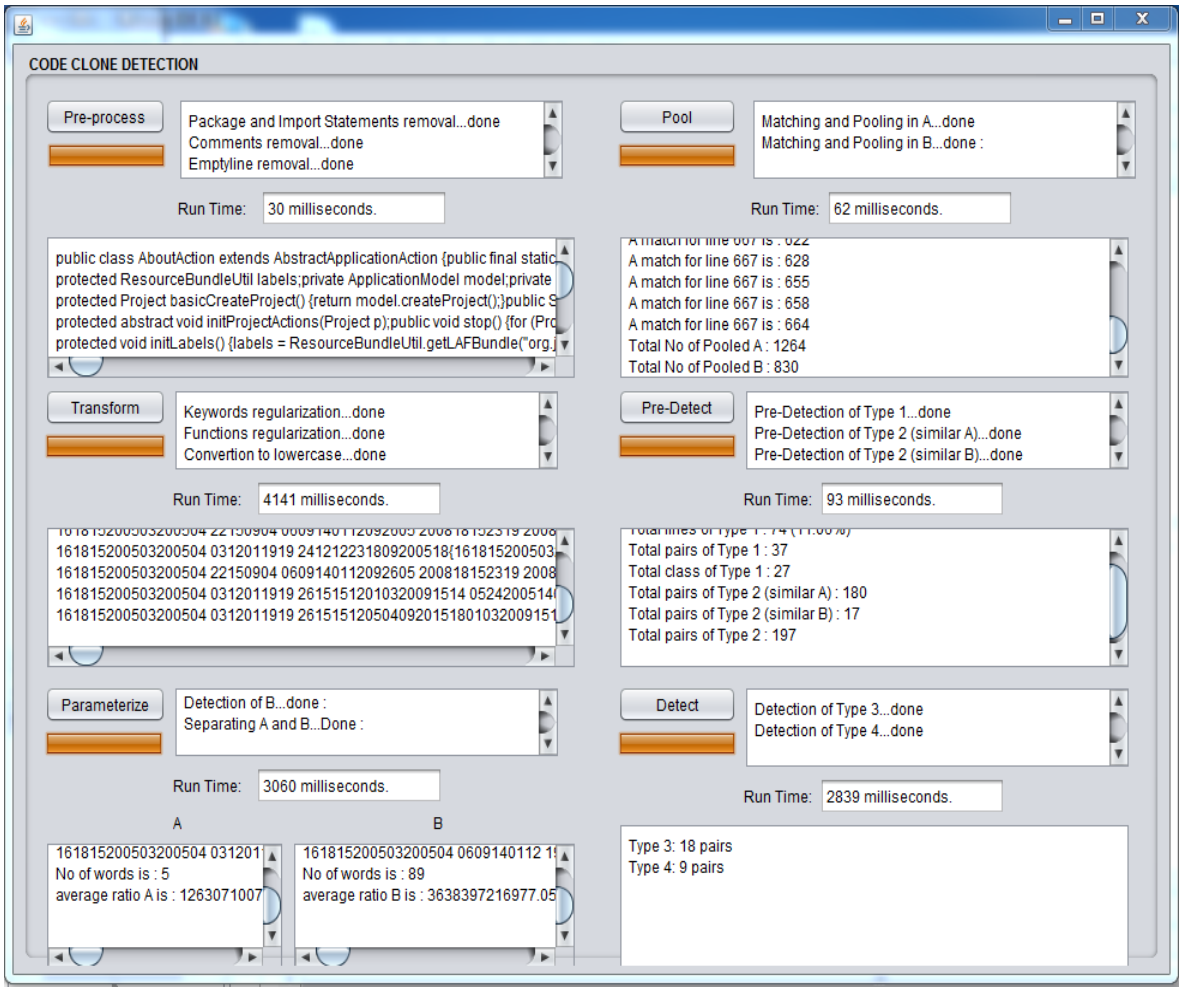
APPENDICES

A1: Project Gantt chart

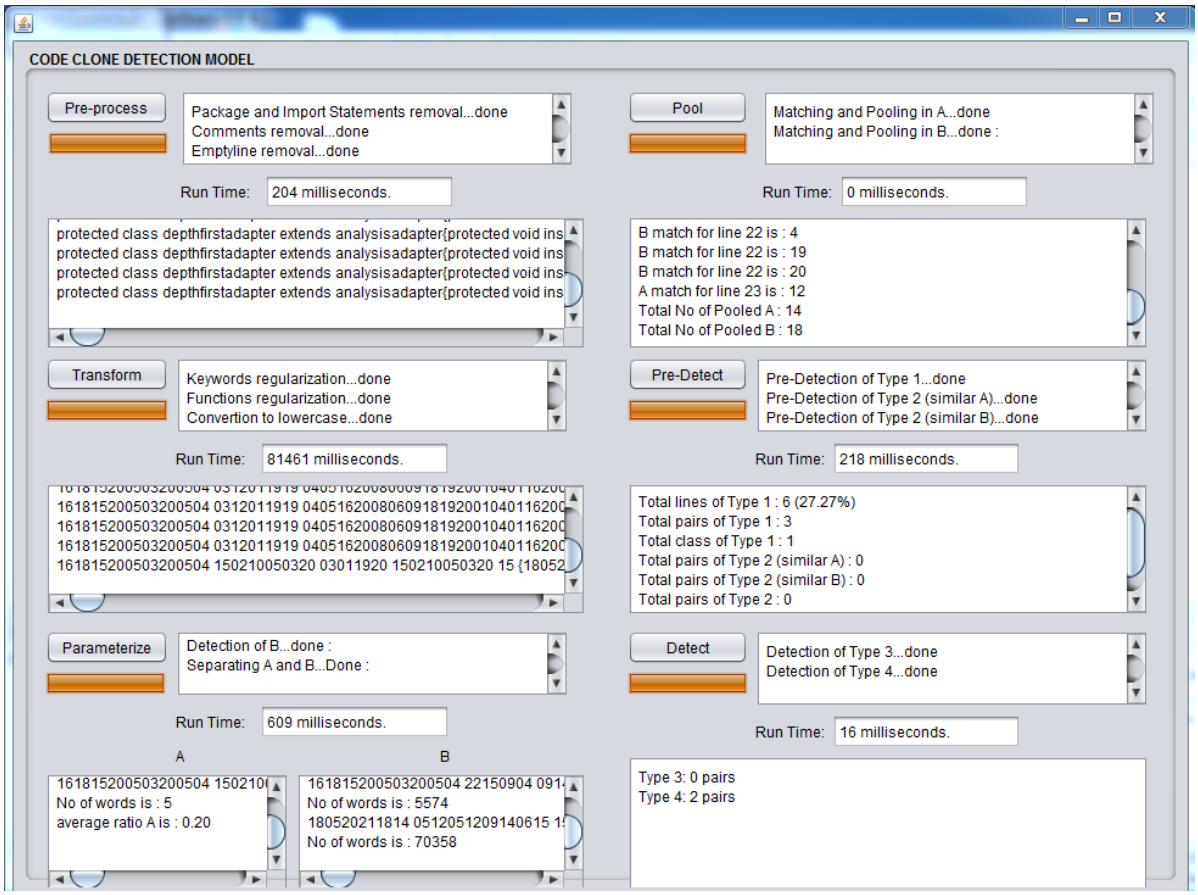


ID	Task Name	Start	Finish	Duration
1	**PHASE 1: LITERATURE REVIEW & ANALYSIS**	15/02/2016	15/03/2016	22d
2	Supervisor briefing on PSM 1 for research and project base	17/02/2016	17/02/2016	1d
3	Define research area and identify the topic	19/02/2016	19/02/2016	1d
4	Finalizing the research title	22/02/2016	22/02/2016	1d
5	Search for related work	23/02/2016	24/02/2016	2d
6	Search for related methods and techniques	25/02/2016	26/02/2016	2d
7	Exploring End Note and Google Scholar	29/02/2016	01/03/2016	2d
8	Identify the problem statements	02/03/2016	02/03/2016	1d
9	Address the objectives	03/03/2016	03/03/2016	1d
10	Submit draft of chapter 1	04/03/2016	04/03/2016	0d
11	Finalizing the problem statement and the objectives	07/03/2016	08/03/2016	2d
12	Editing chapter 1 and outline chapter 2	09/03/2016	11/03/2016	3d
13	Finalizing the related methods and techniques	14/03/2016	16/03/2016	3d
14	Editing chapter 2	17/03/2016	21/03/2016	3d
15	Submit correction of chapter 1 and draft of chapter 2	18/03/2016	18/03/2016	0d
16	**PHASE 2: DESIGN AND DEVELOP PROTOTYPE**	22/03/2016	01/09/2016	118d
17	Data collection	22/03/2016	24/03/2016	3d
18	Analyzing the selected variables and outline chapter 3	25/03/2016	29/03/2016	3d
19	Editing chapter 3	30/03/2016	05/04/2016	5d
20	Submit draft on chapter 1-3	06/04/2016	06/04/2016	0d
21	Correction draft on chapter 1-3	07/04/2016	20/04/2016	10d
22	Design the prototype	21/04/2016	11/05/2016	15d
23	Design the interface	21/04/2016	11/05/2016	15d
24	Submit chapter 3	12/05/2016	12/05/2016	0d
25	Finalize chapter 1-3	18/05/2016	26/05/2016	7d
26	PSM 1 Report submission	27/05/2016	27/05/2016	0d
27	Develop the prototype	02/06/2016	02/09/2016	67d
28	**PHASE 3: EVALUATION**	06/09/2016	30/12/2016	84d
29	**VERIFICATION**	06/09/2016	13/09/2016	6d
30	Compare sets of data	14/09/2016	19/09/2016	4d
31	Compare results	20/09/2016	23/09/2016	4d
32	**VALIDATION**	26/09/2016	21/10/2016	20d
33	Mapping the requirement	24/10/2016	28/10/2016	5d
34	Outline of chapter 4	31/10/2016	02/11/2016	3d
35	Editing chapter 3-4	03/11/2016	08/11/2016	4d
36	Submit chapter 1-4	09/11/2016	09/11/2016	0d
37	Obtain the requirement	07/11/2016	22/11/2016	12d
38	Outline chapter 5	23/11/2016	30/11/2016	6d
39	Editing chapter 5	01/12/2016	15/12/2016	11d
40	Submit chapter 5	16/12/2016	16/12/2016	0d
41	Finilize chapter 1-5	19/12/2016	29/12/2016	9d
42	Full report submission	02/01/2017	02/01/2017	0d

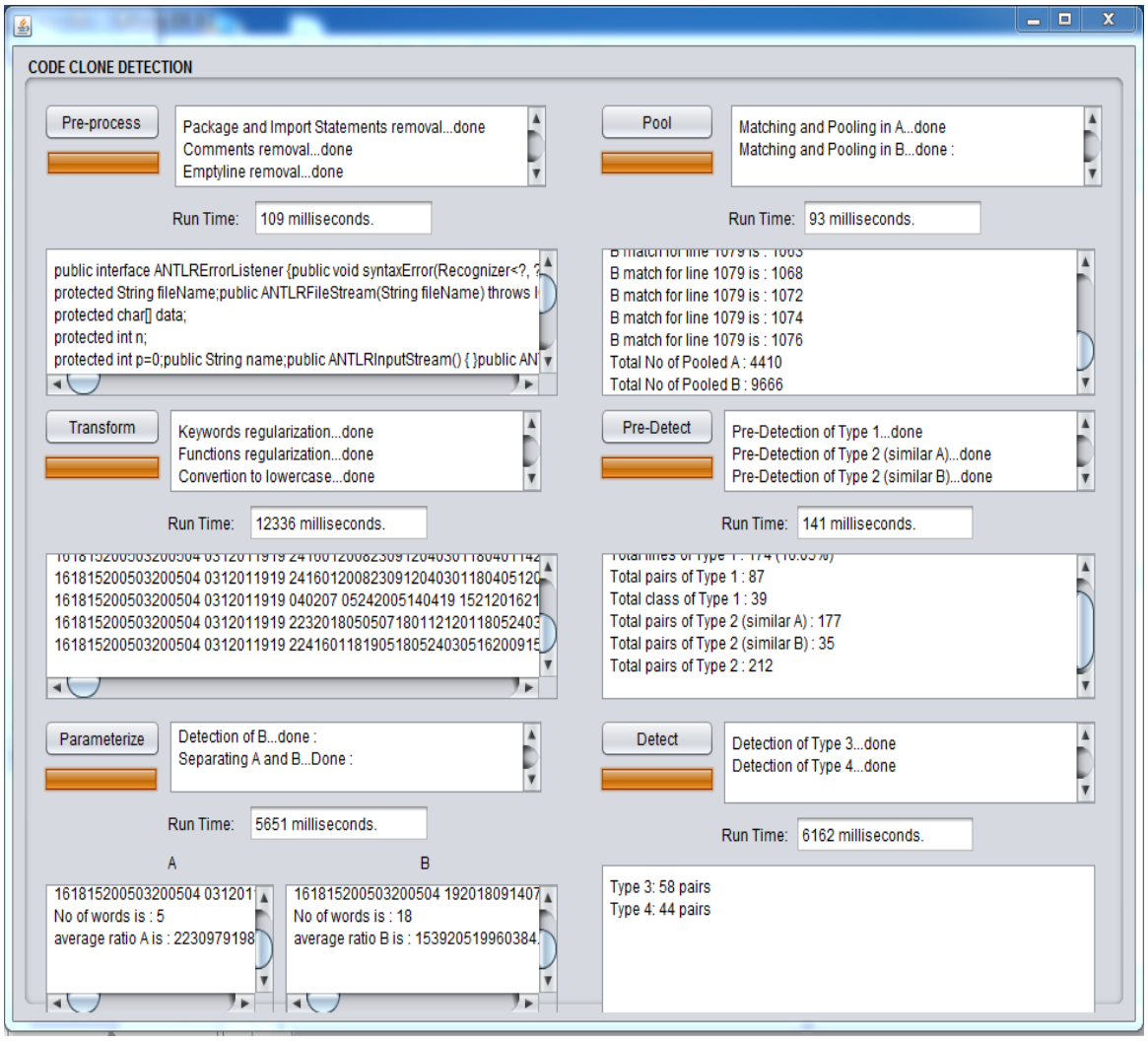
A2: Sample output of code clone detection JhotDraw application



A3: Sample output of code clone detection JhotDraw application



A4: Sample output of code clone detection JhotDraw application



A5: Codings

```
public class TokenAlpha extends javax.swing.JFrame {

    static String newLine = System.getProperty("line.separator");
////This will retrieve line separator dependent on OS.
    static String preSeconds; //for preprocessing
    static String transSeconds; //for transformation
    static float seconds = 0; // for preprocessing
    static float secondstemp; // for preprocessing
    static float seconds2 = 0; // for transformation
    static float secondstemp2; // for transformation

    static String pr1 = "Package and Import Statements not done"
+ newLine; //holds result pr
    static String pr2 = "Comments removal not done" + newLine;
//holds result pr
    static String pr3 = "Emptyline removal not done"; //holds
result pr

    static String tr1 = "Keywords regularization not done" +
newLine; //holds result tr
    static String tr2 = "Function regularization not done" +
newLine; //holds result tr
    static String tr3 = "Convertion to lowercase not done";
//holds result tr
    //create files array that can store 1000 files at max
    static File[] files = new File[1000];
    static boolean determiner = false; //to stop execution in
PartOneTokenizeJavaFile.main before selecting files in
FileChooser
    static String transText = ""; //string to hold transformation
text
    static String paraText = ""; //string to hold
parameterization text
    static String tempLine = ""; // to hold string for readline
to change to lowercase
    static String readline = "";
    static int countLine = 0;
    /**
     * Creates new form TokenBeta
     */
    public TokenAlpha() {
        initComponents();
    }

    /**
     * This method is called from within the constructor to
initialize the form.
```

```
        setDefaultCloseOperation
(javax.swing.WindowConstants.EXIT_ON_CLOSE);

        jPanel1.setBorder
(javax.swing.BorderFactory.createTitledBorder("Al Fahim V.1"));

        jButton1.setText("Pre-process");
        jButton1.addMouseListener(new java.awt.event.MouseAdapter
() {
            public void mousePressed(java.awt.event.MouseEvent
evt) {
                jButton1MousePressed(evt);
            }
        });

        jButton2.setText("Transform");
        jButton2.addMouseListener(new java.awt.event.MouseAdapter
() {
            public void mousePressed(java.awt.event.MouseEvent
evt) {
                jButton2MousePressed(evt);
            }
        });
        jButton2.addActionListener(new
java.awt.event.ActionListener() {
            public void actionPerformed
(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        jButton3.setText("Parameterize");
        jButton3.addMouseListener(new java.awt.event.MouseAdapter
() {
            public void mousePressed(java.awt.event.MouseEvent
evt) {
                jButton3MousePressed(evt);
            }
        });

        jButton4.setText("Detect");

        jButton5.setText("Visualize");

        jTextArea1.setColumns(20);
        jTextArea1.setRows(5);
        jScrollPane1.setViewportView(jTextArea1);
```

```
    private void jButton2MousePressed(java.awt.event.MouseEvent
evt) {//GEN-FIRST:event_jButton2MousePressed

        try {
            Thread.sleep(500); //sleep 500 milliseconds to wait
for progress bar
            jProgressBar2.setValue(50);
            Thread.sleep(1500); //sleep 1000 milliseconds to wait
for progress bar
            jProgressBar2.setValue(100);
            //1000 milliseconds is one second sleep.
        } catch(InterruptedException ex) {
            Thread.currentThread().interrupt();
        }

        jTextArea2.append(tr1); //display to textarea1
        jTextArea2.append(tr2);
        jTextArea2.append(tr3);
        jTextArea7.append(transText); //display latest code up
down on text area 6

        jTextField2.setText(transSeconds + " milliseconds.");
//display the running time for pre-processing

    }//GEN-LAST:event_jButton2MousePressed

    private void jButton1MousePressed(java.awt.event.MouseEvent
evt) {//GEN-FIRST:event_jButton1MousePressed

        //Make the jframe that hold the filechooser visible
        JFrame frame = new MainClass();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        frame.pack();
        frame.setVisible(true);
        determiner = true;

    }//GEN-LAST:event_jButton1MousePressed

    private void jButton2ActionPerformed
(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_jButton2ActionPerformed
        // TODO add your handling code here:
    }//GEN-LAST:event_jButton2ActionPerformed

    private void jButton3MousePressed(java.awt.event.MouseEvent
evt) {//GEN-FIRST:event_jButton3MousePressed
```

```
private void jButton1MousePressed(java.awt.event.MouseEvent evt) {//GEN-FIRST:event_jButton1MousePressed

    //Make the jframe that hold the filechooser visible
    JFrame frame = new MainClass();
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.pack();
    frame.setVisible(true);
    determiner = true;

}//GEN-LAST:event_jButton1MousePressed

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_jButton2ActionPerformed
    // TODO add your handling code here:
}//GEN-LAST:event_jButton2ActionPerformed

private void jButton3MousePressed(java.awt.event.MouseEvent evt) {//GEN-FIRST:event_jButton3MousePressed
    jTextArea8.append(paraText);
}//GEN-LAST:event_jButton3MousePressed

   //JFIlechooser main class
public class MainClass extends JFrame {

    public MainClass() {

    //Make the filechooser visible
    JFileChooser fileChooser = new JFileChooser();
    //show current directories
    fileChooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
    fileChooser.setDialogTitle("Choose a file");
    this.getContentPane().add(fileChooser);
    fileChooser.setVisible(true);

    //enable multi files selections
    fileChooser.setMultiSelectionEnabled(true);

    //Determine the approve, cancel, and error option action
    int result = fileChooser.showOpenDialog(null);
    switch (result) {
    case JFileChooser.APPROVE_OPTION:
    System.out.println("Approve (Open or Save) was clicked");

    //put selected files into files array
    files = fileChooser.getSelectedFiles();
    //System.out.println(Arrays.toString(files));
    break;
    case JFileChooser.CANCEL_OPTION:
```

```
FileWriter fw = new FileWriter(temp);
BufferedWriter out = new BufferedWriter(fw);


boolean eof = false;
do {
readline = in.readLine();

if (readline != null) {

 readline = readline.trim(); // remove leading and trailing whitespace
 //Preprocessing rules--start
 long preStartTime = System.currentTimeMillis();  //calculate current time for preprocessing

    if (readline.length() == 0) {
          System.out.println(readline + "cont1");   // which rule used
        pr3 = "Emptyline removal done";
        continue; // Remove emptyline
    }
    if (readline.startsWith("import") || readline.startsWith("package")) {
          System.out.println(readline + "cont2");
        pr1 = "Package and Import Statements removal done" + newLine;
        continue; // Remove packages and imports
    }
    if (readline.contains("//")) {
          System.out.println(readline + "cont3");
        pr2 = "Comments removal done" + newLine;
        continue; // Remove comments
    }
    if (readline.startsWith("*") || readline.startsWith("/ ") || readline.startsWith("//")) {
          System.out.println(readline + "cont4");
        pr2 = "Comments removal done" + newLine;
        continue;    // Remove comments
    }
    if (readline.startsWith("/*") && readline.endsWith("*/") || readline.startsWith("/*") || readline.endsWith("*")) {
          System.out.println(readline + "cont5");
        pr2 = "Comments removal done" + newLine;
        continue;  // Remove comments
    }
    if (readline.startsWith("//") || readline.contains("*")) {
          System.out.println(readline + "cont6");
        pr2 = "Comments removal done" + newLine;
        continue; // Remove comments
    }
```

```
    // Preprocessing rules ---end
long preEndTime = System.currentTimeMillis(); //calculate end time of preprocessing
// change to second    seconds = (endTime - startTime) / 1000F;

jTextArea6.append(readline + newLine);  //write to jtextfield6

secondstemp = (preEndTime - preStartTime); //hold temporary time for preprocessing
seconds = seconds + secondstemp; //adds all preprocessing time in the loop


    //   Transformation rules --- Start

long transStartTime = System.currentTimeMillis();  //calculate current time for transformation

        tempLine = readline.toLowerCase();  //change all to lower case
        readline = tempLine; //copy tempLine to readline
        tr3 = "Convertion to lowercase done";


    if (readline.contains("string ") ) {
          readline = readline.replace("String", "[S]");  // Remove String with [S]
        readline = readline.replace("string ", "[S] ");
        tr1 = "Keywords regularization done" + newLine;
    }
    if (readline.contains("int ")) // Remove int with [I]
    {
        readline = readline.replace("int ", "[I] ");
        tr1 = "Keywords regularization done" + newLine;
    }
    if (readline.contains("char ")) {
        readline = readline.replace("char ", "[C] ");  // Remove char with [C]
        tr1 = "Keywords regularization done" + newLine;
    }
    if (readline.contains("double ")) {
        readline = readline.replace("double ", "[D] ");  // Remove double with [D]
        tr1 = "Keywords regularization done" + newLine;
    }
    if (readline.contains("float ")) {
        readline = readline.replace("float ", "[F] ");  // Remove float with [F]
        tr1 = "Keywords regularization done" + newLine;
    }
    if (readline.contains("public ")) {
        readline = readline.replace("public ", "protected "); // Standardize access function to protected
        tr2 = "Functions regularization done" + newLine;
    }
    if (readline.contains("private ")) {
        readline = readline.replace("private ", "protected ");  // Standardize access function to protected
```

```
readline = readline.replace("a", "01");
readline = readline.replace("b", "02");
readline = readline.replace("c", "03");
readline = readline.replace("d", "04");
readline = readline.replace("e", "05");
readline = readline.replace("f", "06");
readline = readline.replace("g", "07");
readline = readline.replace("h", "08");
readline = readline.replace("i", "09");
readline = readline.replace("j", "10");
readline = readline.replace("k", "11");
readline = readline.replace("l", "12");
readline = readline.replace("m", "12");
readline = readline.replace("n", "14");
readline = readline.replace("o", "15");
readline = readline.replace("p", "16");
readline = readline.replace("q", "17");
readline = readline.replace("r", "18");
readline = readline.replace("s", "19");
readline = readline.replace("t", "20");
readline = readline.replace("u", "21");
readline = readline.replace("v", "22");
readline = readline.replace("w", "23");
readline = readline.replace("x", "24");
readline = readline.replace("y", "25");
readline = readline.replace("z", "26");

tempLine = readline.replaceAll("\\s+", "");
readline = tempLine;

paraText = paraText + readline + newLine;
  for (int i = 0; i < readline.length(); i++) {


}

//System.out.println(countLine + "\t" + originalLine + "\t" + inputLine +"\n");
//System.out.println(countLine + "\t" + wordcount + "\n");


    out.write(readline);
    } else
    eof = true;
    } while (!eof);

    in.close();
```

```
        if (readline.contains("protected")) {

            out.write (newLine); // Add separator for line write
            out.write (newLine); // Add separator for line write
           }
     //
if (readline.startsWith("protected")) {

countLine = countLine + 1;

readline = readline.replace("=", "");
readline = readline.replace("()", "");
readline = readline.replace("{", "");
readline = readline.replace("}", "");
readline = readline.replace(",", "");
readline = readline.replace(".", "");
readline = readline.replace("_", "");
readline = readline.replace("!", "");
readline = readline.replace("<", "");
readline = readline.replace(">", "");
readline = readline.replace("&&", "");
readline = readline.replace("&", "");
readline = readline.replace("==", "");

readline = readline.replace("[S]", "31");
readline = readline.replace("[I]", "32");
readline = readline.replace("[C]", "33");
readline = readline.replace("[D]", "34");
readline = readline.replace("[F]", "35");
readline = readline.replace("[s]", "31");
readline = readline.replace("[i]", "32");
readline = readline.replace("[c]", "33");
readline = readline.replace("[d]", "34");
readline = readline.replace("[f]", "35");

readline = readline.replace("a", "01");
readline = readline.replace("b", "02");
readline = readline.replace("c", "03");
readline = readline.replace("d", "04");
readline = readline.replace("e", "05");
readline = readline.replace("f", "06");
readline = readline.replace("g", "07");
readline = readline.replace("h", "08");
readline = readline.replace("i", "09");
readline = readline.replace("j", "10");
readline = readline.replace("k", "11");
```

```
            jTextArea2.setColumns(20);
            jTextArea2.setRows(5);
            jScrollPane2.setViewportView(jTextArea2);

            jTextArea3.setColumns(20);
            jTextArea3.setRows(5);
            jScrollPane3.setViewportView(jTextArea3);

            jTextArea4.setColumns(20);
            jTextArea4.setRows(5);
            jScrollPane4.setViewportView(jTextArea4);

            jTextArea5.setColumns(20);
            jTextArea5.setRows(5);
            jScrollPane5.setViewportView(jTextArea5);

            jLabel1.setText("Run Time:");

            jLabel2.setText("Run Time:");

            jLabel7.setText("Run Time:");

            jLabel8.setText("Run Time:");

            jLabel10.setText("Run Time:");

            jTextArea6.setColumns(20);
            jTextArea6.setRows(5);
            jScrollPane6.setViewportView(jTextArea6);

            jTextArea7.setColumns(20);
            jTextArea7.setRows(5);
            jScrollPane7.setViewportView(jTextArea7);

            jTextArea8.setColumns(20);
            jTextArea8.setRows(5);
            jScrollPane8.setViewportView(jTextArea8);

            javax.swing.GroupLayout jPanel1Layout = new
    javax.swing.GroupLayout(jPanel1);
            jPanel1.setLayout(jPanel1Layout);
            jPanel1Layout.setHorizontalGroup(
                jPanel1Layout.createParallelGroup
    (javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(jPanel1Layout.createSequentialGroup()
                    .addContainerGap()
                    .addGroup(jPanel1Layout.createParallelGroup
```