ENHANCING GENERIC CODE CLONE DETECTION MODEL FOR C BASED APPLICATION

AINUN SYAHIRAH BINTI ADNAN

Bachelor of Computer Science (Software Engineering)

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : _____

Date of Birth : _____

Title : _____

_____

_____

Academic Session : _____

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)
_____	_____
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 (OPTIONAL)

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

Author's Name
Thesis Title

Reasons (i)

(ii)

(iii)

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 checked this thesis and in my opinion, this thesis is adequate in terms of scope and quality for award of degree of Bachelor of Computer Science (Software Engineering).

_____

(Supervisor's Signature)

Full Name : DR. AL- FAHIM BIN MUBARAK ALI

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 : AINUN SYAHIRAH BINTI ADNAN

ID Number : CB16011

Date :

ENHANCING GENERIC CODE CLONE DETECTION MODEL FOR C BASED APPLICATION

AINUN SYAHIRAH BINTI ADNAN

Thesis submitted in fulfilment of the requirements for the award of the degree of Bachelor of Computer Science (Software Engineering)

Faculty of Computer Systems & Software Engineering

UNIVERSITI MALAYSIA PAHANG

MAY 2019

ACKNOWLEDGEMENTS

First of all, I want to thank God for the thanksgiving for giving me the health of my body as long as I prepare this thesis. I also would like to thank all those who helped and support in making this thesis especially for my parents, family, supervisor, lectures and friends.

Therefore, I sincerely want to thank my supervisor, Dr. Al- Fahim Bin Mubarak Ali for giving me a lot of knowledge and guidance throughout I prepare this thesis. Without his helps I am sure that I cannot succeed to complete this thesis. Besides, I would like to thank to my course mates and my roommates that share with their knowledge and perception.

Lastly, I want to give my appreciation to my family because the warm encouragement that has been given to me. I also want to say thank you for those were involved directly or indirectly in this project.

ABSTRAK

Klon kod adalah istilah yang digunakan untuk menggambarkan kod yang digunakan dalam sistem berulang kali. Pada masa ini terdapat empat jenis klon kod, iaitu jenis-1, jenis-2, jenis-3 dan jenis-4, yang dapat dikesan oleh beberapa alat pengesan klon kod. Setakat kualiti sistem berkenaan, klon kod boleh menyebabkan sistem memakan lebih banyak memori untuk menjalankan fungsi, kerana banyak kod yang digunakan berulang kali. Klon kod juga mempengaruhi proses penyelenggaraan sistem. Sekiranya fragmen kod yang disalin mengandungi pepijat, semua kod dengan persamaan dengan fragmen kod yang disalin mestilah diperbaiki satu persatu. Ia mengambil masa yang lama untuk mengekalkan sistem. Aplikasi yang dibangunkan di Java dan C biasanya mempunyai kemungkinan besar klon kod disebabkan penggunaan bahasa-bahasa ini yang melampau dalam pembangunan aplikasi. Oleh itu, objektif utama penyelidikan ini adalah untuk memperbaiki model pengesanan klon kod untuk mengesan klon kod dalam bahasa pengaturcaraan C. Pelbagai model boleh didapati untuk mengesan kod klon yang merupakan model klon generik, model saluran paip generik, model klon bersatu dan model pengesanan klon kod generik. Pengesanan Klon Generik Kod (GCCD) adalah keadaan model seni yang mengesan klon kod sehingga menaip 4 dalam program Java. Proses model ini adalah pra-pemprosesan, pemprosesan, parameterisasi, pengkategorian dan pengesanan padanan. Tujuan penyelidikan ini adalah untuk meningkatkan prototaip untuk mengesan klon kod dalam bahasa pengaturcaraan C. Oleh itu, objektif utama penyelidikan ini adalah untuk meningkatkan prototaip model pengesanan klon generik kod untuk mengesan klon kod dalam bahasa pengaturcaraan C. Kajian ini memberi tumpuan kepada meningkatkan dua proses, iaitu pra-pemprosesan dan transformasi. Untuk menilai penambahbaikan yang dibuat dalam kajian ini, prototaip GCCD dipertingkatkan dan diuji dengan menggunakan set data penanda aras yang dipanggil dataset penanda aras Bellon. Hasil yang diharapkan dari kajian ini ialah prototaip GCCD dapat mengesan kloning bahasa pemrograman C.

ABSTRACT

Code clone is a term used to describe a code used in a system repeatedly. There are currently four types of code clones, namely type-1, type-2, type-3 and type-4, which can be detected by some code clone detection tools. As far as the quality of a system is concerned, the code clone can cause a system to consume more memory to perform a function, due to the many codes that are repeatedly used. The code clone also affects the system maintenance process. If the copied code fragment contains a bug, all code with similarities to the copied code fragment must be fixed one by one. It takes longer to maintain the system. Applications developed in Java and C usually has the largest occurrence of code clone due to the extreme usage of these languages in application development. Therefore, the main objective of this research is to improve the code clone detection model to detect the code clone in the language of C programming. Various models are available to detect a clone code which is a generic clone model, generic pipeline model, unified clone model and a generic code clone detection model. Generic Code Clone Detection (GCCD) is the state of the art model that detects code clone up to type 4 in Java programs. This model's process is pre-processing, processing, parameterization, categorization and match detection. The aim of this research is to improve the prototype for the detection of code clones in the C programming language. Therefore, the main objective of this research is to improve the prototype of the generic code clone detection model to detect the code clone in the language of C programming. This research focuses on improving two processes, namely pre-processing and transformation. In order to evaluate the improvements made in this research, the GCCD prototype is enhanced and tested using a benchmark data set called Bellon's benchmark dataset. The expected result of this research is that the GCCD prototype can detect the C programming language code clone.

TABLE OF CONTENT

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

GCCD	Generic Code Clone Detection
VB	Visual Basic
PDG	Program Dependence Graph
CP	Clone Pair
CC	Clone Class
AST	Abstract Syntax Tree
SWOT	Strength Weakness Opportunity Threat
LOC	Line Of Code

CHAPTER 1

INTRODUCTION

1.1 Introduction

In the process of developing a system, developers typically reuse the code they make into another line of code or other modules in the same system. This practice or behaviour makes it easier for them to speed development process. This method can be called code clone or duplicate code. Code clone is a process of reusing a code repeatedly. Discussion related to code clone has been done by several researcher, most of the current systems, the results showed a fraction that between 20% to 30% of module in system may be cloned (Baker, 1995).

There are 4 types which are **Type-1,** code portion or fragments are identical, except for spacing, layout and comment variations. In clone type 1 also known as exact clone. This is because the different fragments are exact copies of each other. **Type-2**, code portion or fragment are syntactically identical, except for literals, identifiers, types, comments, and layout and whitespace variations. This type is like Type-1 which is have similarity of code portion or fragment to each other, but have more addition for identifiers declared (constants, class, methods, name of variables and so on). **Type-3**, this type is some evolution of type-2, the different of this type is the fragment that are copied from another source code have some added statements, with removal of some statements or some modified statement. **Type-4**, two or more code fragment or portion that have similarity with each other and perform the same computation, but different syntactic variants, called by type-4.

This type is not mandatory that code fragment should be copied from somewhere or different programmer but have same functionality. Otherwise, the category of two code fragment or portion is under Type-4. Most of the current code clone detection tools only detect until Type-3. Meanwhile, the purpose of this study, which is the addition of a function for the prototype of GCCD is to detect the code clone for a C programming language that supports to detect the code clone up to Type-4.

Although removing a code clone has the risk of changing the software structure or framework, however, to find out a code clone in a software, it is an advantage to consider whether it is necessary to make a changes for the code clone. In visual basic, there already have their code clone detection, but the code clone detection tool on this application only detect until Type-3.

Generic code clone detection model (GCCD) is a model used to detect code clone in a system. There are some research that explains the code clone model, the generic code clone model is one of the last models developed by pre-existing research. Previously, a prototype has been developed using this model, GCCD model can detect code clone up to Type-4 compared to other models but it can only detect code clone in Java programming language.

Also, there are several types of models and approaches for detecting code clone. This approach is widely used by other researches in establishing or improving existing code clone detection methods. Below are five (5) approaches of detecting code clone (Al-Fahim, 2015):

i) String based comparison

This approach will detect code clone by comparing source code by text / string that in the line of code in the same fragment.

ii) Metric based comparison

This Approach works by comparing different metrics and gathering into one, and on the basis of similar value, the similarity will detected.

iii) Tree based comparison

In this approach, abstract syntax tree of a system is produced. Then, tree matching technique is applied to detect similar sub trees. When the result come out between two sub trees, the source code of similar sub trees is returned as clone-pair.

iv) Token based comparison

This approach are detecting the code that have token sequence obtained from division the line of source code. The unique of this approach is the characteristic is using hash function.

v) Graph based comparison

The graph based comparison are the approach that detecting code clone by converting the code into the graph version. After several of graph that already detect the similarity, the function of this approach will continue with a clone-pair.

In addition to the approach, the model for code clone detection has a role to unify the tools and also the approaches that will be describe at above. An approach is a way of detecting a code clone by comparing of something. The model is a more complex way of detecting code clone on the system.

There are several model that can be used for code clone detection. The three model that always use for code clone detection, it is generic clone model, generic pipeline clone model, unified clone model, and generic code clone detection model . This research will focusing on enhancing the generic code clone detection model.

Each model have their own unique function. The description for each model will be explain at below:

Generic Clone Model (Giesecke, 2007)

This model focuses more on the detection and deletion of code clone. This model separates between cloning detection, management and description with layers. The separation in this model is very clear between the processes of clone detection using code layer.

Generic Pipeline Model (Biegel & Diehl, 2010)

This model has five (5) process to detect the code clone. First process is parsing, this process transform the source code into source unit. Second step of process is pre-processing, it is use for normalize the source units that have been transform at the first process. Continue with third step of process that is pooling, this process is to group the pre-processed source units into group based on user define criteria. After finish the third step, the process entering the fourth step of process that are comparing process, this process recursively compares source unit in all pools using a divide and conquer strategy. And the last step of the process is filtering, this process is to remove irrelevant and non-relevant candidate sets. For irrelevant, the sets are from the result set and for non-relevant, the set are out of the result set.

Unified Clone Model (Kapser, Harder, & Baxter, 2012)

This model is a model that has a generic model that can detect all types of code clone detection tool. Different clone representation of existing tools is the purpose of this model is designed. There are four groups of analysis results in this model, detection for cloning trials and management, integration of additional data from other sources, replication of scientific studies and benchmarking of cloning detection techniques.

Generic Code Clone Detection Model (Al-Fahim, 2015)

Generic code clone detection model code is a prototype of a model that is still in development stage. The model consists of a combination of processes that perform *pre-processing, transformation, parameterization, categorization,* and *match detection process*. Each process in this model is more complex than the previous models. The models apply the existing code clone detection approaches as part of its process.

In this research, generic code clone detection is the main model that will be used for development in the detection of code clone. Although this model is new and less of research that does development for this model, this model is able to detect various code clones that exist in a system.

1.2 Problem Statement

In the field of programming, C language very close among programmer. It is very basic language that compulsory and need to know before we learn other language. Other than that, this language also easy to learn when someone have basic knowledge in programming. Dennis Ritchie was developed originally C between 1969 and 1973 at Bell Labs (Dennis Ritchie, 2018). C is a basic procedural dialect. It was intended to be assembled utilizing a moderately direct compiler, to give low-level access to memory, to give dialect develops that guide productively to machine guidelines, and to require insignificant run-time bolster. Notwithstanding its low-level capacities, the dialect was intended to support cross-stage programming. A gauges agreeable C program that is composed in light of compactness can be arranged for a wide assortment of PC stages and working frameworks with few changes to its source code. The dialect has turned out to be accessible on an extensive variety of stages, from installed smaller scale controllers to supercomputers.

There are interesting and coherent projects accessible on Basic Input Output, If else, Conditional Statement (Ternary Operator) based projects. These projects are beginning from exceptionally fundamental level to abnormal state. Each program contains source codes, yield and clarification of the rationale. A significant number of the software engineer do the duplicate on their code lines and create code clone rehearses on the framework they create. Be that as it may, duplicate as a programming practice has an awful implication since it will make the code clone or copy code that will influence the support stage (Miryung Kim et al., 2004). Therefore, the existing prototype of generic code clone detection model will try to overcome the problem. It aims to maximize maintenance on the codes by detect the code clone that exist in a source code. The current prototype, only able to detect the code clone for using Visual Basic .net platform application and Java.

1.3 Objectives

The purpose of this research is to make the improvement for the prototype of generic model of code clone in C language. Below are some of the objectives for this research to produce satisfactory results:

i) To enhance the rules in the pre-processing and transformation process in generic code clone detection model for C language clone detection.

ii) To develop the improvement by modifying the existing prototype using Microsoft Visual Basic application recommended by developer of GCCD.

iii) To evaluate the enhanced prototype of the generic code clone detection tool using the dataset that contain the C language source code.

1.4 Scopes

In order to accordance with the title of this research which is “A code clone detection model for C platform application”, this research has some scopes as a follows:

i) Focuses on code clone detection for C language application platform.

ii) Focuses on clone pair.

1.5 Thesis Organization

The first chapter of this thesis is introduction. In this chapter explained about this research, code clone definition, problem background of research, the current solution and the purpose solution for this problem, problem statements, and objective of this research that should be achieve at the end of this chapter.

In chapter two, describe about literature reviews that related work of this research and the concept. The explanation of a code clone with different terminologies will be discuss in this chapter. Then, the code clone detection process, approaches with tools and evaluation metric is elaborated. Structure of C language will be explain at the last of this chapter.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

In this chapter, the definition of code clone will be explained in closer detail with the detection and approximation process. Some of the main topics for code clone approach and code clone detection models were explained in the previous chapter. Conclusions will be found in the last part of this chapter.

2.2 Code Clone

Software engineers often use a code fragment in the software development process by copying and pasting from another project that is successfully running into the project. Such reused code fragments make any changes to the new code location. These same or similar reused fragments of code called code clone. (Dang et al., 2017).

Code clone is the usual research terminology. A number of different terminologies used to describe the clone code. The variation in the terminology occurs due to the varied similarity and tolerance level allowed for the code clone. (Bellon, Koschke, Antoniol, Krinke, & Merlo, 2007).

Another understanding in another word about code clone is plagiarism. It makes the maintenance phase harder and takes a long time. A code clone is a code in another source file location similar to the code used by the programmer. Copy and paste code reuse makes it more difficult to modify consistently. To modify

the code clone, the entire code block must be changed if there is a problem. The task of maintaining the system is also becoming big. (Morshed, Rahman, & Ahmed, 2012). An example of code clone is shown at figure 2.1.

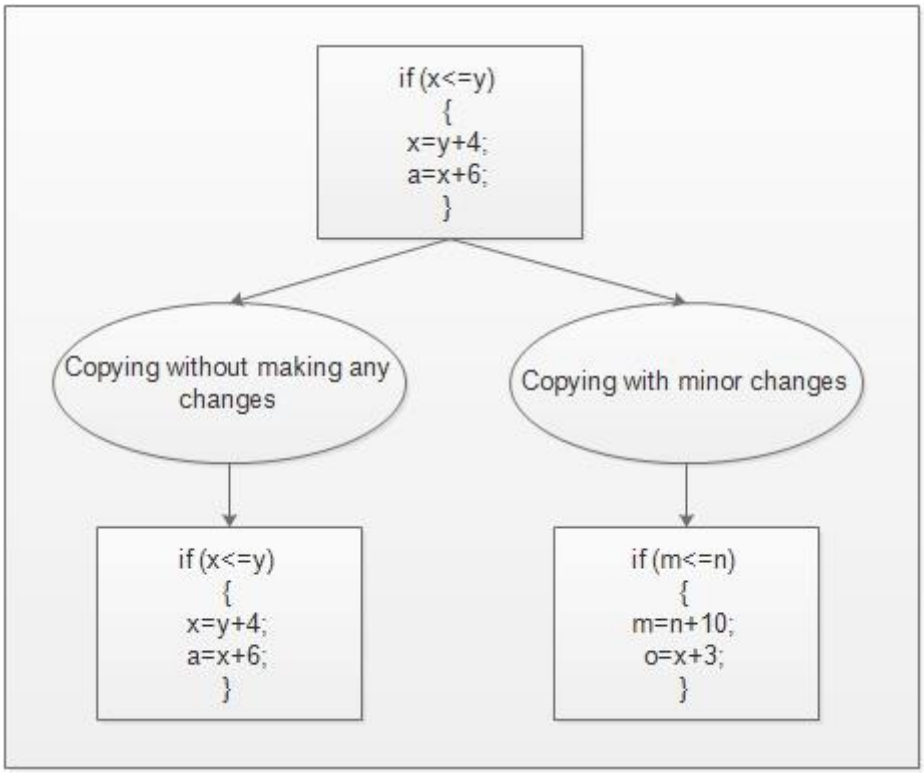


Figure 2.1 Code Clone

As illustrate in figure 2.1, code clone means the same code that is reused in another source in the same project, which will make the code read by the system longer. At the top of the figure the variable is x and y are used in another source and has the same variable. This usually happens in the program to avoid changing the new structure of the software development process. But it will make the system read more code line and take longer. This code clone copies without changes, which means that the variable and value are the same. And, copying the variable with minor changes also changes the value, but the head and body of the code is the same.

2.2.1 Advantage of Code Clone

The code clone has several advantages to detect clone code, some application needed to find a problem or bug. Otherwise the variety to find the code clone must be improved to use the application. Some of suggestion from developers to improve the application for code clone detection.

i. Find candidate of library

The usability of code fragment has been demonstrated by copying and reusing the code several times. This can be incorporated into the library and it can be officially reused.

ii. Getting new idea for system.

The possibility to obtain general ideas from other files containing dependencies of another functionality of the code itself. For example, the programmer has a memory management code line, the files contain a copy and data structure must be implemented with dynamic assigned space.

iii. Helps mining research aspects

It is also necessary to detect crosscutting concerns in the mining aspect. The cross- cutting code is typically duplicated across the entire application that could be identified with clone detection instruments.

iv. Finds same patterns

The same functional pattern of the cloned fragment can be found if all the cloned fragments of the same source are detected.

v. Detects bugs software

Some bug detect clone detection techniques in a vital role. The evidence can be found by comparing a bug with another software when one software system matches another.

vi. Detect copyright and plagiarism

Similar code with the same function is detected without any changes called plagiarism, for the use of code clone detection, plagiarism and copyright are useful.

vii. Contribute software evolution research

The dynamic nature of different clones in different versions of the system using clone detection techniques used in software development research.

viii. Contributes in code compacting

Clone detection technology can be used for compact devices by reducing source code size.

2.2.2 Disadvantages of Code Clone

In addition to the benefit of code clone, the quality, reusability and maintenance impacts are severe. The following are the disadvantages of detecting a programs code clone:

i. Increase possibility of bug propagation

Some bugs can already be copied into another project or code fragments in some code already copied by the programmer. Reusing the code without any changes to adapt the code to other code line will affect the original bug in the new project or code fragments that the programmer is working on. It will significantly increase the spread of bugs in the system.

ii. Bad design

Cloning a code will make the code structure a bad design. Lack of a good future of heritage or abstraction. The code part for another project becomes difficult to reuse in the future. Hard to understand the function code. And it will affect the maintenance phase.

iii. Difficulty for improvement or modification the system

The code clone can cause difficult in improvement or modification the system because need to take more attention to understand and additional time for the existing clone implementation.

iv. Increase maintenance cost, work and time

Some case in the maintenance phase have a bug that comes from the code clone. Many bug cases come from code clone that can cause maintenance to take more time to ensure that the system works well. There is no guarantee that this bug was eliminated from other similar parts during reuse or maintenance.

2.2.3 Reasons of Code Clone

The reason for reusing a system code is summarized in three reasons. The first reason for the use of code clones is development and maintenance. This is due to the reuse of code in the development of the system, it can also be called code reusability. For instance, a programmer builds a system that uses existing code in another system by changing a few functions, it can cause code clone.

As mentioned above, the code clone reason can be categorized into two, which are:

i. Simple reuse through copy and paste

Implementing the same code by copy- pasting used by the developer. This method is something that always happens in each system development because the code function can be reused in a new system.

ii. Design and functionality reuse

Some system functions and logic can be reused and repeatedly. This can happen in some subsystem applications. Linux is an operating system with a high level of duplication due to the frequent interface raised the same as the previous interface.

Then, a reason why code clone is used in developing the programmer's limitations and shortcomings. A programmer is someone who develops a system and can also be included in it. I can conclude I two reasons. This is:

i. Difficult to understand of large system

Because of the large system, developers have problems understanding the system. Developers prefer to use the existing code and copy and paste the code into the system it created.

ii. Lack of knowledge in development process

Some developers face problems, such as lack of understanding, when developing a system, to assemble a code to work in accordance with its function. In this situation, the developer will usually look for a code with the same function to solve the problem. This can cause the system code clone.

In development, a programmer must have a problem in several existing programming languages. Programming language is important in development and maintenance and has an important code clone role. Below is the reason for the programming language code clone.

i. Lack of programming language usage

Some programming languages currently have different mechanisms for facilitating program development. However, some programming languages do not have a mechanism source and require the repetition of the programming language.

ii. Time saving for writing a code

For programmers, writing a new code in a line takes longer. For them, copy- paste is easier to save time in developing the program than writing long and complex codes.

2.3 Process of Code Clone Detection

Most of the code clone detection process commonly used by most researchers as shown at figure 2.2. Besides the process, there are several approaches to detecting code clone in a software. Detection approaches can be distinguished based on information that can be obtained from the approach itself (Roy & Cordy, 2007). This approach is a process that is included in the code clone detection tool (Bellon et al., 2007). There are several processes that must be done before and during the detection of code clone. Some researchers have previously done some research and produced ways and approaches in detecting code clone. Instead of several combinations of research produced by the researchers, a generic process was obtained. Most of the code clone detection tools adopt some or all of

the processes mentioned in the process of generic code clone detection. Figure 2.2 will be explained about the detailed detection process of generic code cone.

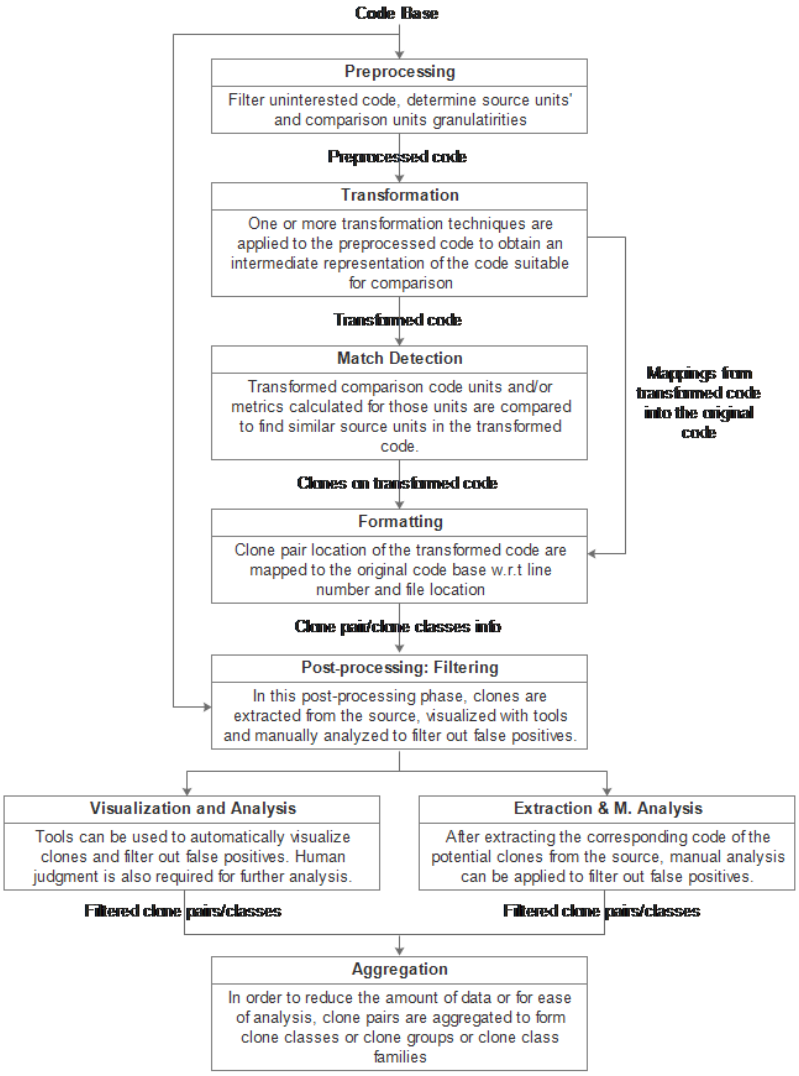


Figure 2.2 Clone Detection Process

2.3.1 Pre-processing

This process is the first phase in the code clone detection. In this pre-processing phase there are three processes and each process have its own goal. Which is:

i. Delete nonessential parts.

In this phase, all the uninteresting and unimportant parts will be remove to avoid the as a code clone. E.g. comment, empty line.

ii. Determine source units

After the pre-process, some source code is partitioned into the source unit. This source unit is not appropriate; therefore, cannot be combined beyond the unit limit of the source. There are several granularities for the source unit such as classes, functions, methods, blocks and sequence of source code lines.

iii. Define the comparison unit

The source unit will be partitioned into smaller units to get more details. It is based on the comparison function of a method. Then, the small unit that has been partitioned, will be divided into several lines or tokens that are intended for comparison. The unit order is another important aspect in determining the unit of comparison.

2.3.2 Transformation

This process is the main point to transform the comparison units with previous one process to another representation or from original code or source code to extract comparable properties. Table 2.1 show the approaches that might be used for transformation by (Roy & Cordy, 2007).

Table 2.1 Transformation Approaches

Transformation Approach	**Description**
Pretty printing of source code	Source code is transformed into normal form.
Removal of comments	The comments are removed from the source code.
Removal of whitespace	Almost all approach applies this approach. This approach is to remove the whitespaces from source code.
Tokenization	Each line of the source is divided into tokens. These tokens correspond to a lexical rule of the involved programming language. The token lines are then formed into token sequences for the detection purposes.
Parsing	The entire source codes of the software are parsed into abstract syntax tree. The source unit and comparison units are represented in the form of sub tree of the parse tree.
Generating Program Dependence Graph (PDG)	Usually used in semantic based approaches where source units or comparison units are in the form of sub graphs of these program dependency graph.
Normalizing identifiers	Usually applied in most of the approaches where the identifiers of the source code

	are replaced by a single token in such normalizations.
Transformation of program elements	Apart from normalization of the identifiers, a few transformation rules might be used on the elements of the source code.
Calculate metrics values	This usually happens in metric based approach where several metric are calculated from raw or transformed PDG and abstract syntax tree source codes to be used in detecting clone.

2.3.3 Match Detection

This step or process is to find coloration or similarity between transformation comparisons units using by comparison algorithm to detect the similarity. Comparison units owned by source units are aggregated to fixed granular clones. For irregular cloning granularity, aggregation is continued until the aggregate number is above the limit specified for the number of aggregate comparison units. This is to ensure that the largest group comparison unit is found. The production of this process is a matching list associated with the modified code that contains the candidate clone pair. Each clone usually represents information like the corresponding debris location in the clone that is changed.

2.3.4 Formatting

This process is to convert or change from the list of clone pairs obtained in conjunction with the modified source code to the clone list of the original source code. Usually each clone pair location is converted into a line number in the original source file. The general form will be group of list or nested tuple. This something with the comment in the code or inside the semicolon in code. That nested tuple it will find with the original code or founder commented it.

2.3.5 Post Processing and Aggregation

This phase is to find the false positive clone to filtered code out by:

i. Manual Analysis

Take the row data of the code clone and analysed manually to find the similarity from the original code after extraction. This phase can find or define the positive false after filtered out.

ii. Visualization

This method uses to visualize the obtained clone pair list. The list it can traced using this method and can analysis use this visualization tool. This visualize is used to describe the list of pairs of clones acquired. This list can be used to illustrate clones. Visualization tools can speed up the manual analysis process in removing false positives.

Aggregation is the process of reducing the amount of data or performing certain analyses. These clones are aggregated in clusters, classes, clones, clones, or clones. Usually this process is ignored in most of the code clone detection process.

2.4 Code Clone Detection Approaches

This part discuss about the five major approaches used to detect code clone. The component inside the approaches are string based, metric based, token based, tree based and graph based comparison approaches.

To guiding the comparison it will detection approach use the string base and more clear or understand how it work. The string base detection through this comparison approach is done by comparing source code. They have two part of source code by string base to compared approach. The source code are compared each with an "AND" statement for the same text / string. If one or more similar source code sections are found, the result will return as a cloning pair. This

technique can’t detect structural cloning because the detection does not focus on the program's structural elements. Normally user or programmer use this in code to use the function of code, they can’t normalization process, it is because the detection approach usually directly use in program source code to detect code clone (Roy & Cordy, 2007). However this approach by line to line to filtered, there are still works by combining with other techniques to detect the code clone.

In this version of the matrix-based comparison approach (Roy & Cordy, 2007) put on a string-based compression throttle. The difference in comparison of it through two phases between string based comparison and the metrics based comparison it uses different metres from fragments or parts of the code and compare them to each other (Roy & Cordy, 2007). The matrix difference matrix with a string-based comparison ratio that only compares the line to the base line of the source code. Some work uses distances in metric vectors instead of line coding to compare meanings (Göde & Koschke, 2009). The use of tree syntactic and parser trees through a tree based comparative approach. In the programming language there is represented as a representation when the source code is transformed into a tree structure (Rattan, Bhatia, & Singh, 2013; Roy & Cordy, 2007). This approach uses a sub-partition of tree abstract syntax program based on hash function and then compares subtests in partition the same through some techniques such as tree matching (Jiang, Misherghi, & Su, 2007) or dynamic programming (Duala-Ekoko & Robillard, 2007, 2008). Most variable names and literal values of sources are discarded in this approach but there is still a clone detection technique that can still be used to find the same code (Roy & Cordy, 2007). Normally a metric-based comparison is used to truncate a type 3 code clone (Rattan et al., 2013) this comparison approach exists.

A token-based comparative approach is done by dividing the line of code into a sequence of business. In this approach, the whole source code is changed to a token of the token. This can be done using lexical analysis (Rattan et al., 2013). The tokens do uniquely characterization using the hash function (Göde &

Koschke, 2009). The detection was done to find out duplication at which the original source code mark was originally represented as a code clone (Roy & Cordy, 2007). Approach to tree suffix or suffix based on token makes the main work is on the comparison finder (Kamiya, Kusumoto, & Inoue, 2002b) is an example of a clone detection tool that uses tokens comparison.

On the other hand, using a graphical-based technique to detect the code clone is a comparison-based approach. Program dependence graph (PDG) is used to represent the reliance of source data flow source code(Rattan et al., 2013). The PDG technique improves the source code representation abstraction compared to other approaches as it considers semantic information source code (Komondoor & Horwitz, 2001; Krinke, 2001; Liu, Chen, Han, & Yu, 2006). Usually the PDG obtained from the software is used with algorithms corresponding to sub-isomorphic graphs in search of the same subgraphs; it is called a clone (Roy & Cordy, 2007).

Each approach has its own strengths and weaknesses. The string-based approach is limited to the exact clone and in certain cases clones are almost right. Clones precisely refer to clone type 1 and clones almost exactly refer to clone type-2. The string detection strength lies in the work involved directly in manipulating source code and string based languages. The matrix-based approach relies heavily on a set of critical criteria extracted through an enhanced version of a string-based approach. Advanced transformation methods used in tokens-based detectors such as ccfinder (Kamiya, Kusumoto, & Inoue, 2002a) or the engineering techniques may improve the clone detection code but potential remains unknown. Most of the code clone detectors use a tree-based approach because the strength of this approach lies in detecting the same clone, the same clone and the corresponding clone. Specific clones refer to type-1 clones, the same clone refers to clone type-2 and similar clones referring to type-3 clones. Scalability is another major advantage of a tree-based approach compared to other

approaches. Scalability refers to cod detection in applications and tree-based approaches can detect code clones in large size applications.

Correction of the approaches that have been made to the clone of the code clone, a number of clone detection system approaches combined with certain techniques to improve the results of the code clone detection. There is also a combinatorial approach extended to be a code clone detection tool. Examples of techniques combined with main approaches include comparisons of substrings, scatter plots, point plots, feature vector clustering, the longest one, the longest substrings and hashing. Table 2.2 shows approaches with accepted techniques.

Table 2.2 Approaches with Adopted Technique

Approaches	**Adopted Techniques**	**Tools / First Author**
String based comparison	Substring Comparison and Scatter Plot	Duploc (Ducasse et al., 1999)
	Substring Comparison	Simian (Harris, 2003)
	Dot Plot / Scatter Plot	DuDe (Wettel and Marinescu, 2005)
	Longest Common Subsequences	Nicad (Cordy & Roy, 2011)
Metric based comparison	Abstract Syntax Tree and Metrics	CLAN (Mayrand, Leblanc, & Merlo, 1996)
	Abstract Syntax Tree with Metrics, Feature Vector Clustering and Dynamic Programming	(Kontogiannis, Demori, Merlo, Galler, & Bernstein, 1996)
	String with Metrics and Fingerprinting	(Perumal, Kanmani, & Kodhai, 2010)
	Abstract Syntax Tree with Metrics and Dynamic Programming	(Lavoie, Eilers-Smith, & Merlo, 2010)

	Set of matrices with light weight hybrid (LWH) approach	CloneManager (Kodhai & Kanmani, 2014)
Tree based comparison	Tree Parse with Dynamic Programming and Longest Common Substrings	(Yang, 1991)
	Abstract Syntax Tree with Hashing and Dynamic Programming	CloneDr (Baxter, Yahin, Moura, Sant'Anna, & Bier, 1998)
	Parse Tree with Dynamic Programming and Longest Common Substring	Sim (Gitchell and Tran, 1999)
	Abstract Syntax Tree with Metrics and Feature Vector Clustering	ClemanX (Nguyen, Nguyen, Pham, Al-Kofahi, & Nguyen, 2009)
	Abstract Syntax Tree with Metrics and Levenshtein Distance	CSeR (Jacob *et al.*, 2010)
Token based comparison	Suffix Tree	Dup (Baker, 1995) CCFinder (Kamiya et al., 2002a) D-CCFinder (Livieri, Higo, Mazushita, & Inoue, 2007) iClones (Göde & Koschke, 2009)
	Abstract Syntax Tree	Java Code Clone Detector (JCCD)
	Suffix Array	RTF (Basit, Rajapakse, & Jarzabek, 2005)
	Hashing	FCFinder (Sasaki, Yamamoto, Hayase, & Inoue, 2010)
Graph based comparison	Program Slicing	Scorpio (Higo & Ueda, 2007)

		PDG-DUP (Komondoor & Horwitz, 2001)
	n-length patch matching	Duplix (Krinke, 2001)

2.5 Clone Metrics

This part will discuss about the various detection approaches and models with their corresponding tool, it will provide to search the previous researchers. The challenging to compare these tools since the application approaches and tool are more towards picking the right approaches and tool for a particular purpose on interest. The researchers for the tools evaluation purpose, there are several common parameters will used.

This method need the higher precision to detect the code clone and detection tool that should be able to define the lesser false positives. Precision refers to the accuracy of detected code clones, the ability in detecting code clones with hidden cloning relationship. They not all the code is clone and visible textually and some editing activities can disguise the copied fragments with the original source code, therefore it is important for the code clone detection tool to detect the hidden code clones. For high action used the tool to detect a code clone are can find the hidden clone relationships. Adaptability refers to the tool being independent of programming language. Since there is a lot of programming languages available in the market, therefore it is important for the code clone detection tool to be adaptable to different kind of programming languages.

The tool also should be capable of adapting to multiple programming languages since the programming style involves multiple programming at the same time. Scalability refers to code clone coverage detected in the system. A good code clone detection tool it can detect a clone of code from a large system.

This tool should also shoed be able to control legacy and intricate systems without limitations of computer memory.

Performances of runtime to execute code clone detection in a larger system need to be fast. It also refers to the time taken to execute a process in a code clone detection tool. The code clone detection should not only detect code clone with high accuracy detection; but also, must be able to do it at minimal detection times. For the apart from mentioned metrics, there are two major clone metrics or granularity level used in reporting clones which are clone pair (CP) and clone class (CC). In this major clone they have similarity relation that occur between the code fragments. The similarity relation between cloned fragments, or also known as equivalence relation, refers two or more code fragments that have the same sequences; in which the sequences may refer to the source code itself or the source units.

2.6 Related Work

The code clone detection model has also been proposed to detect the clone of the code. As mentioned in the previous section, the proposed code clone detection approach results in different clone tracking code results. Therefore, the model is proposed to have a unified code clone detection and decision (Harder, 2013). There are three code clone models used in the code clone research domain. Generic clone model (Giesecke, 2007), generic pipeline models (Biegel & Diehl, 2010) and unified clone model (Kapser et al., 2012).

2.6.1 Generic Clone Model

The generic clone model is a model that illustrates the clone that exists in a program (Giesecke, 2007). This model contains separation of clone's detection, explanation and management using layers. The main use of this model is to illustrate clones. This reduces efforts in the implementation of tools that support these activities. Figure 2.3 shows an overview of the generic clone model.

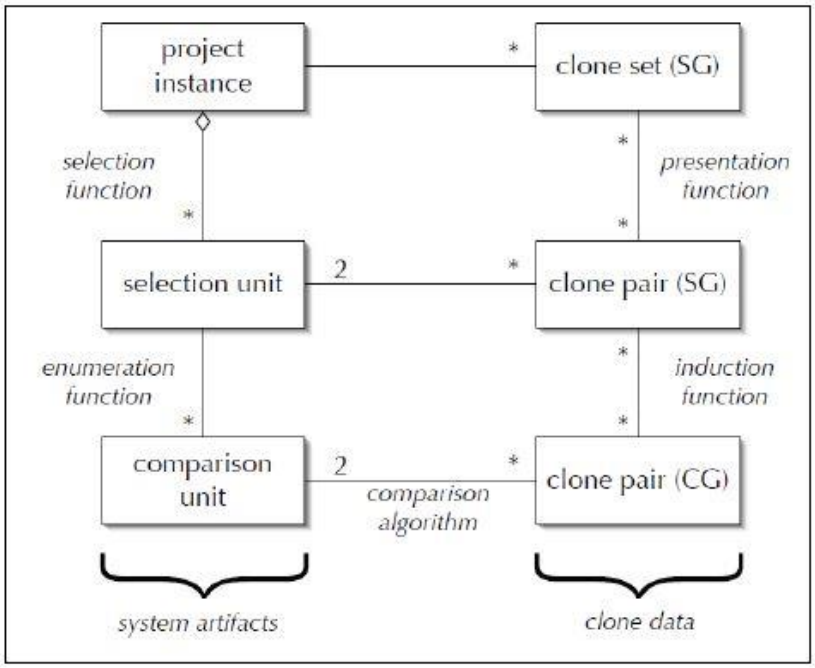


Figure 2.3 Overview of Generic Model Clone (Giesecke, 2007)

Based on the system artefact, there are two types of elements associated with the corresponding model, an artefact representing the piece of cloning data generated by a cloning detection algorithm based on system artefacts (Giesecke, 2007). The highest level of this model is a sample project. An example is organized into a selection unit and a comparator unit through the selection and enumeration functions. Then, this example is known as cloning data. At the top level, clone data is encapsulated in clone sets containing clone pairs. The cloning pair is present in two granularities of the selected unit and its reference unit. Then, clone pairs are grouped into clone sets by presentation function. To reduce redundancy in cloned sets, the clone set used is from a collection with leading reference elements.

2.6.2 Generic Pipeline Model

The generic pipeline model is a model for detecting code clones; it is a combination of processes with all the steps required in the code clone detection process. There are five steps in this generic pipeline mode (Biegel & Diehl, 2010). In the picture below will show the diagram view of this model.

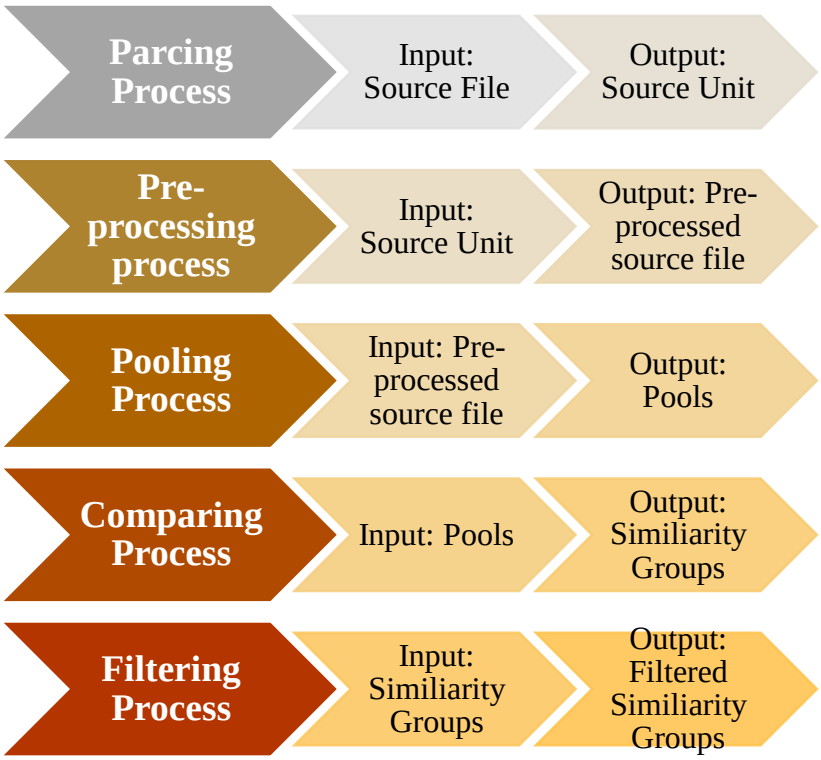


Figure 2.4 Generic Pipeline Model (Biegel & Diehl, 2010)

In the first process is the process of parsing and change the source code into source units. In converting the source code to the source unit, this process will work. The sub tree of abstract syntax tree (AST) is the representation of the unit to be used. The source unit is an input of this source file as well as the output. The source unit is then used as input for the second process. The second process in this model is a pre-process that serves to normalize the source unit and adds

additional annotations to the source unit. Normalization will convert the source unit into a regular form that will make the different source units, will become more similar. AST will be used as input and also as output that has been processed before. Multiple level processors will implement the process. This model will generate an output which is a previously processed source file. In the third process, the source file will be processed and inserted into the next stage.

The third process in this model is a unification process that will group the AST source units that have been processed into groups of groups that match the characteristics defined by the criteria set by the user of this detector. Pool is the outcome of this process. Pool obtained from the previous process is the input to be included in the next process is the comparison process. This process will unify the results. This is the recursive process of the source unit in which all the pools use a divide and conquer strategy. The output of this process is the clone equation group. This group will be used as input for the last process that is the screening process. This process is the last process in the Generic Pipeline model. The purpose of this process is to remove a collection of irrelevant cloning candidates from the result set.

2.6.3 Unified Clone Model

A model that seeks to have a generic model that can represent the results of all cloning tools is the Unified Clone Model (Kapser et al., 2012). Figure 2.5 below will show the steps of unified clone model.

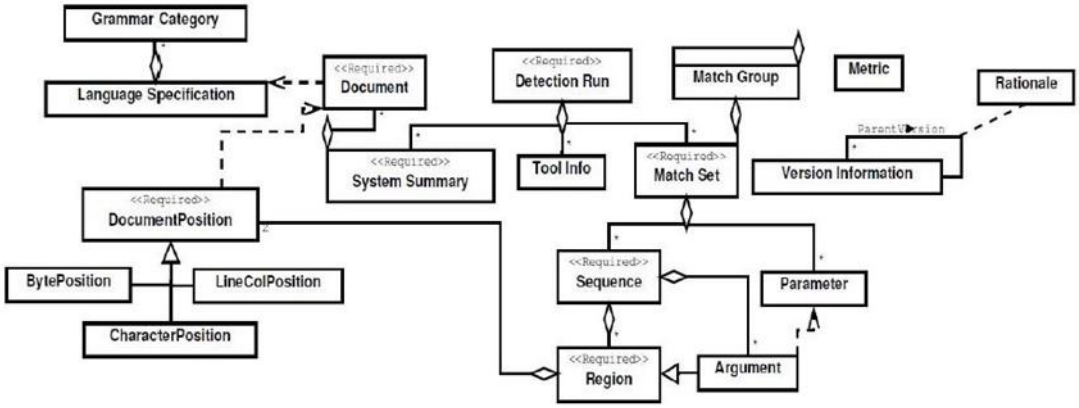


Figure 2.5 Unified Clone Model (Kasper et al., 2012)

This model is resistant design. This model is designed through a different clone representation of the current tool. As a concept analysis, it will use eleven applications to be the case that has been used. Then, the results of the analysis will be divided into four groups that are detection for cloning trials and management, integration of additional data from other sources, replication of scientific research and comparison of cloning detection techniques. The weakness of this model is still in design and does not have the right file format to present data (Harder, 2013).

2.6.4 Strength and Weakness of Model

This model has the advantage and disadvantages for the system. This part discuses about this situation what it can do and not. Table 2.3 shows the Strength Weakness Opportunity Threat (SWOT) analysis of the models.

Table 2.3 SWOT analysis on models

Feature	**Generic Clone Model** (Giesecke, 2007)	**Generic Pipeline Model** (Biegel & Diehl, 2010)	**Unified Clone Model** (Kapser et al., 2012)	**Generic Code Clone Detection Model** (Al-Fahim, 2015)
Strength	To make the description of the clones is possible, it will have to clear separation of clone detection process definition using layers.	It consists of step by step process to detect clones in Java applications. It allows customizatio n for the user to manipulate the model.	The model is designed through the different clone representation s of existing tools.	This model able to detect a code clone until Type- 4
Weakness	It does not allow manipulation on it layers to extend the effectiveness of this model.	The extension of this model is limited due to the manipulation on the pre-defined sets and rules in the model.	It is still in design phase and lacks a proper file format for data representation	It is still in improving *pre-processing* and *transformatio n* process.
Opportunit y	The description of the model on the clones	The clone type detection and the process	The realization of the model using user	Declare the part of unnecessary to be detect as

	could be improved.	can be enhanced to obtain a better code clone detection result.	defined process.	a code clone and convert the code into something that can be measure
Threat	The modification implementation of this model is impossible due to its nature of being a plugin.	The application used for evaluation will yield different results compared to existing work.	Different tools might cause variation to the end results	Difficult to implement plugin due to accuracy which is highly possible type of clone and language.

The researcher analyses the general weakness of the model which is the extension of the existing model using the SWOT analysis method. After analysis, the generic clone model does not allow for manipulation of predetermined layers, in order to extend the effectiveness of the model, while the generic pipeline model allows only the manipulation in the process that limits the expansion of the model to improve code clone detection. The realization of this model is a major weakness in this regard. It is very apparent with the struggles in realizing the prototype or tool for the unified clone model.

The generic pipeline model is the only code cloned detection model that detects code clone for Type-1. It consists of five processes used to detect code clone. Use a tree-based comparison approach that converts the source code to a source units that is a tree node and then uses a tree comparison to compare tree nodes. The categorization and filtering process in determining the cloned pool code and the result is manipulated by the user; therefore it may cause the final result of a particular clone to be removed due to user preferences. Although a tree-based comparison approach is well known for achieving good detection results

compared to token-based comparison approaches, the application of tree-based comparison approaches is very expensive in terms of hardware requirements and time. Therefore, a more robust source unit representation and coding cloning detection techniques are required with the prototypes required in detecting all types of code clones.

2.7 Discussion

For detect the code clone, there is five major of approaches, which is string based comparison approach, metric based comparison approach, token based comparison approach, tree based comparison approach and graph based comparison approach. Apart from these approaches, code clone detection models such as the generic clone model (Giesecke, 2007), the generic pipeline model (Biegel & Diehl, 2010), generic pipeline model (Kapser et al., 2012), and generic code clone detection (Al-Fahim, 2015) has been proposed in detecting code clone through multiple processes.

The contain of code clone detection model it have several process to determine the line or code clone will be located while the code clone detection approach is part of the process in the clone detection model of the code used to detect code clone. The use of a clone tracking approach affects the use of parsing and pre-processing on the source code in generating source units. An example can be seen with a Generic Pipeline Model. It uses a tree-based comparative approach; therefore the parsing process and pre-processing process in the Generic Pipeline Model meets the source code requirements that will be changed into a tree node.

The process need to consideration for proposing a code detection model is the model can realization of model and detect code clone by itself. Based on the generic pipeline model, there are five processes that are used in detecting code clone. The processes are parsing, pre-processing, pooling, comparing and filtering process. The parsing and pre-processing is related to source transformation while

the latter three processes is related in application of tree based comparison approach in detecting code clone.

The generic code clone detection has been chosen for the model that will be improve on this research. This model able to detect a code clone until type-4. Also, this model is more valuable to be improve than any model. The process that will be improve on this research is pre-processing and transformation process. Two of this process is to declaring the part of unnecessary to be detect as a code clone and convert the code into something that can be measure.

2.8 Summary

To conclude this chapter, the code clone terminology used in the code clone detection domain already shown in this chapter. Code clone reason benefit and disadvantage of code clone explained in this chapter. Determined code clone using detection and code clone detection approaches. The way code clone detection was discussed using algorithm or method. Five major approaches and three major code clone detection models exist. For approaches, metric- based comparison approach, token- based comparison approach, tree- based comparison approach and graph- based comparison approach. The generic pipeline model, the generic clone model and the unified clone model are also used for three key code clone detection models. Key tools related to approaches and models were also compared and discussed in this chapter. This research method will be shown in the next chapter.

CHAPTER 3

METHODOLOGY

3.1 Overview

This research method will be described in this chapter. In first two sections the operational and theoretical framework used in this research will explain. The research design and conclusion of the methodology to be used in this research will also be explained at the end of this chapter section.

3.2 Operational Framework

This research uses existing models of other researchers. The aim for this research is to improve the prototype of the generic code clone detection tool for detecting code clone in java programming has been developed by previous researchers. Besides that, this research has a scope for improving the prototype of generic code clone detection only for VB.Net application platform.

Figure 3.1 shows the operational framework diagrammatic view. The diagrammatic view of the operational framework consists of three phases, the review of the current function and the prototype rules, the design and evaluation of the proposal. In designing the proposed improvement, two processes will be improved, pre- processing and processing. Three applications will also be used as a dataset for the evaluation phase to be tested by the prototype, so the prototype can detect the C language code clone.

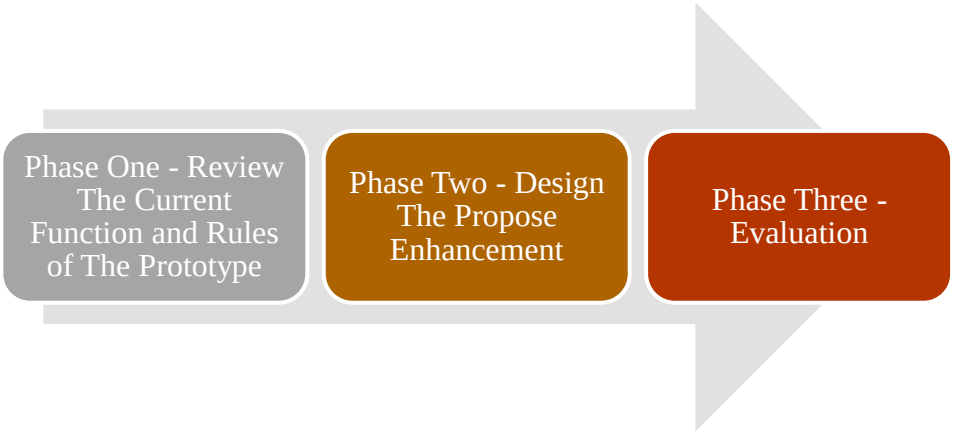


Figure 3.1 Operational Framework

3.2.1 Review the Current Function and Rules of the Prototype

The current function and rules of the prototype are reviewed at the first phase of the operational framework. The purpose of reviewing the current approaches is to gather and analyse the related domain information through the literature. In the process itself, code clone detection, concepts, detection approach and detection tools were discussed in chapter two.

3.2.2 Design the Propose Enhancement

After reviewing the current model approaches, the next phase is to design the proposed improvement. This phase aims to design and define GCCD processes. In this phase, certain requirements analysed in the previous phase will be used. The flow and definition of each process will then be determined in the model and the design will finally be obtained at this stage. In the next phase this design will be used. Details of the code clone detection model are explained and detailed in chapter 4. For this research, the GCCD will only improve the detection of the generic clone code. The process takes only two pre- processing and transformation processes to evaluate the model.

3.2.3 Evaluation

Following the third evaluation phase. The purpose of this evaluation phase is to evaluate and analyse the work proposed by evaluating the prototype. The results of the cloning detection from the prototype will be then be analysed and the analysis results will be summarized in the next step. The research design is described and presented in Section 3.3 and the evaluation results are also shown in chapter 4.

3.3 Research Design

This research aims to detect all clones such as Type-1, Type- 2, Type-3 and Type-4, and the detection category is included in definition of generic code clone detection as described in chapter two. Case studies are used to identify prototypes working capabilities. To find out how the prototypes work, one software practitioner will use researcher to check the performance of generic code clone detection. The development of prototype tools for generic code clone detection is therefore very important to validate the proposed work. There are several steps in the design of this research to validate the approach proposed. The research design is shown in Figure 3.2.

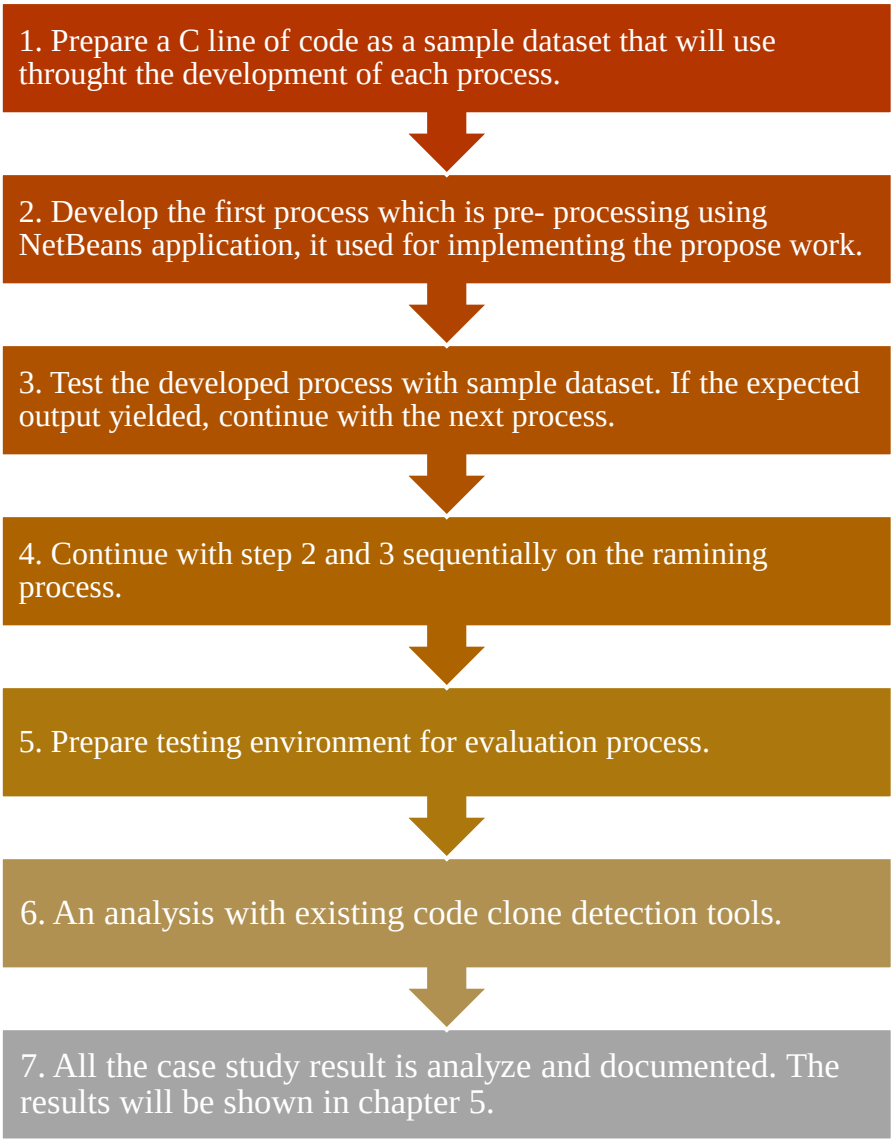


Figure 3.2 Step of Research Design

The first step in this research design is to prepare C source code as a sample data set that will be used during each process development. The dataset is downloaded without websites that provide the source code that can be downloaded free of charge.

The second step is to develop the first process using the NetBeans application. It is used to implement the proposed work. This step is an important step in this research. There are many code categories that are not used to code clone detection processes, such as empty lines, comments and others. The researcher will improve the current program to detect the code clone in .net programming languages.

After finishing step two, the researcher will test the developed process / program with the sample dataset in step three. Continue step four for the next process if the expected output occurred.

Step five to prepare the testing environment for the evaluation process. The program is already improved and ready to use to detect the code clone. The researcher should prepare the hardware and software environment and the data set to be tested by the generic code clone detection tool.

The six step is the case study. An analysis with existing code clone detection tools is working well to make the program clear. This assessment focuses on testing the prototype and comparing it to existing code clone detection tools.

The final step in this research design, step 7, is to analyse and document all the results of the case study. The results are presented in Chapter 5.

3.4 Dataset

Bellon's BenchMark datasets should have C source code. It has been chosen to set the data for this research. This dataset will be used to evaluate the prototype of the C application platform code clone.

The researcher collects the data set system for evaluating generic code clone detection from the open source code provided by the website. The collected dataset is available with the source code itself. The reason why open source code is used is to obtain available data and save time in determining the data to be used in this research.

3.5 Design the Propose Enhancement

In the generic increase in the code clone, the researcher will transfer the prototype from the past one from the detection of the Java code clone to the detection of the C programming language code clone. Some of the processes clearly explained in the following steps:

Figure 3.3 Propose Enhancements

3.5.1 Learn the Process of GCCD

The first step in the design of a code clone detector is to learn how and how to calculate or read and also what types of code clones this tool will read.

3.5.2 Improvement on Functionality and Rules for C Programming Language

After studying and understanding how the prototype of the clone detector tool previously intended for the java application platform, the researcher will add the functionality and rules of the existing prototype to detect the code clone in the VB.Net application platform.

3.5.3 Testing the Functionality of the Prototype Improved

The experiment will test the prototype to detect the VB.Net application platform code clone. If there are errors or problems in the construction of the code clone detection tool, some improvements will be made until the tool works well and produces the prototype until the aim of this research has been achieved.

3.6 Limitation and Assumption

The existing visual studio detection tools have some limitations in finding or detecting the type of code clone in programming. Code clone detection only detected in the visual studio until type 3. Although the purpose of this study can detect the type-4. Furthermore, the limitation of this research lies in the empirical assessment of the models.

Some research and also work on code clone detection has been done for a long time and probably no new things that could be used as research material. There are always some developments from different sources every year, and the improvement of this tool is constantly changing (Mubarak- Ali & Sulaiman, 2014). Several approaches and techniques have previously existed, only the generic pipeline model (Biegel & Diehl, 2010) is accessible and can also be used as references for building the generic prototype cloning detection tool.

The researcher's expectation and assumption of making generic code clone detection tools is:

3.6.1 Detecting all code clone type

As for research in the construction of this tool, the focus of this research is to be able to detect all types of code clones in order to produce tools that work better to detect code clones.

3.6.2 Tools easy to use and easy to understand by the user

Using this tool, it is expected that ease of use of this tool can be achieved to achieve satisfactory results and to make a system or program more qualified.

3.6.3 Improvement of various aspects

Researchers expect improvements from various aspects during the development of this prototype. As long as an improvement recommendation can be understood and expected.

3.7 Software Specification

For this research, some software will be used to improve the generic code clone detection tool. First, the NetBeans software is used to run and improve the generic code clone detection tool prototype. This software is used for editing the source code in java programming language. Visual 2010 basic software to open a dataset tested by generic code clone detection tools. The data set that will be tested can certainly be run in the visual base and the process will be recorded for future use by the researcher to know the performance after the code clone is detected using the generic code clone detection tool. C also use clone doctor to compare code clone detection tools. These tools are available for free download from clone doctor's website. Other comparative tools are code clone analysis. Code clone analysis already installed in the 2015 visual studio. These tools are provided by the visual studio to enable their developer to detect their code clone.

3.8 Hardware Specification

To run the prototype smoothly and achieve a good result, a central processing unit (CPU) is expected to have sufficient specifications for the processing of various tasks in the running of generic code clone detection tools (GCCD). This tool will require a lot of memory to use in the process of improving this tool. Researchers will use a memory size of 4(four) GB as a minimum requirement and the requirements for writing this thesis have been met. The current device used by researchers to improve and evaluate the GCCD meets the requirements. If computer specifications are required, the developer will use a

computer lab at the computer systems and Software Engineering College in Malaysia Pahang (Gambang) University. The faster the process is performed by the prototype of generic code clone detection instruments (GCCD) in the CPU.

3.9 Summary

A summary of chapter three at the end of this chapter describes the operational framework to be used as the first sub topic in this study. There are different process to meet the requirements for the development and use of GCCD. An explanation of the dataset to be used as an example of this tool's wok.

Following the design of the proposed improvement in the development or improvement of prototypes previously existing, the prototype was designed to detect the code clone in the java programming language, whereas the researcher developed or improved the programming language in C

The scope and assumptions continue with different descriptions and also have three points on the assumptions that researchers expect in the development or enhancement of genetic code clone detection.

For development, sufficient computer specification and software specification were required to conduct research and improve the prototype properly and properly. Good detection results can be achieved using qualified computer specifications.

In the next chapter, the focus will be on discussing Generic Code Clone Detection in more detail and understanding the function and the result.

CHAPTER 4

Improvement, Implementation, and Evaluation of GCCD

4.1 Overview

In this chapter, GCCD (Generic Code Clone Detection) will explain and illustrate the research work that the researcher is working to improve and implement. This chapter will explain the details and any implementation or enhancement of each model process.

4.2 Generic Code Clone Detection Model Improvement

There are 5 processes in generic code clone detection model which are pre- processing, transformation, parameterization, and match detection. The objective of this all process is to detect all type of code clone which type- 1, type-2, type-3 and type-4. Figure 4.1 above show the all process of GCCD.

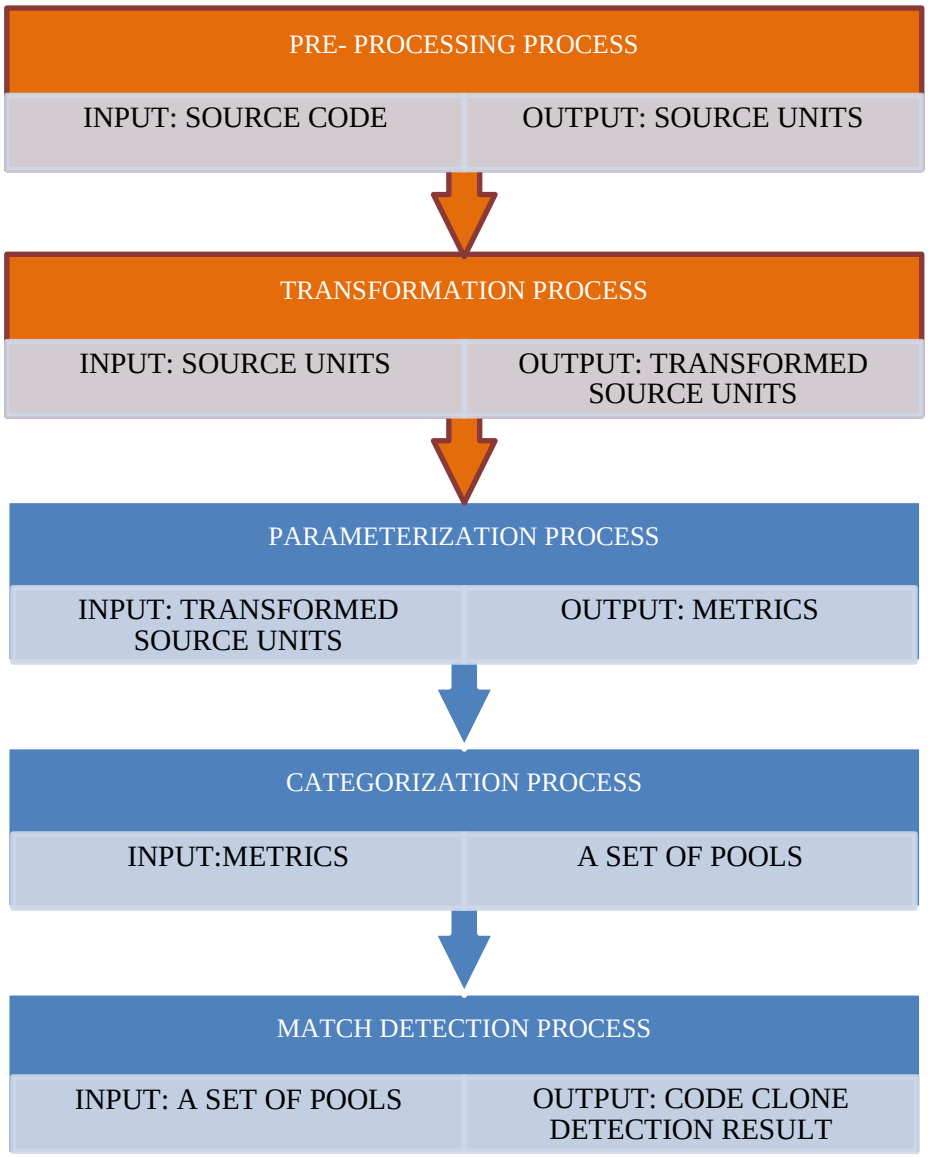


Figure 4.1 Process of Generic Code Clone Detection Model

All the function at Generic Code Clone Detection (GCCD) have been discussing in Chapter 2. This model almost similar with generic Pipeline Model that are followed by previous researcher before built this tool for detecting clone in C language. In this research focusing on improving 2 functions which are pre-processing and transformation process. The explanation about these two processes will be explain below.

4.2.1 Pre- processing Process

At the beginning of this process for Generic Code Clone Detection is pre-processing. The process of it is to produce a normalized source code or source unit. The aim of normalization is to turns the source code into a regular form and to makes the different code into similar. The figure 4.2 shows the diagrammatic of the pre- processing process.

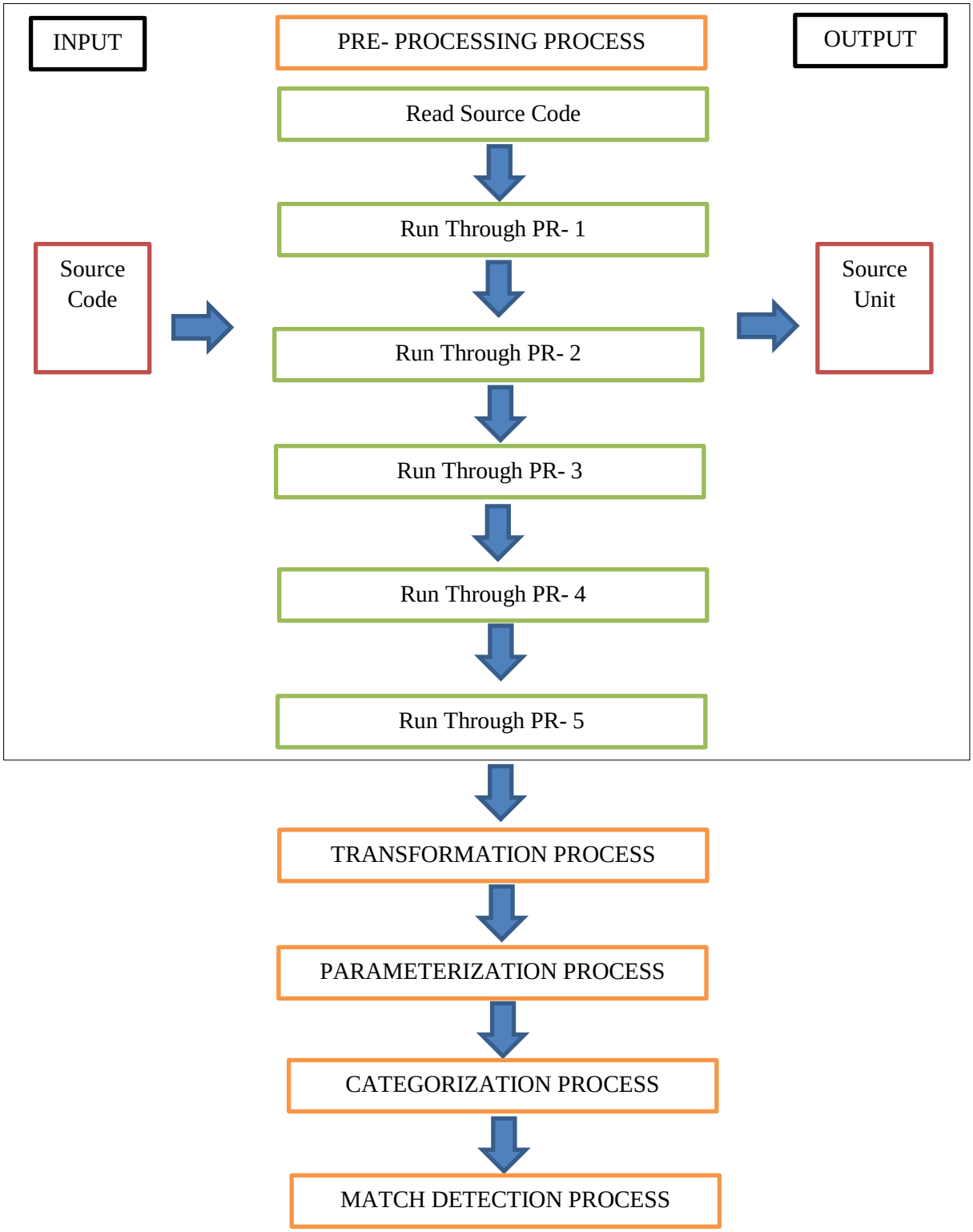


Figure 4.2 Pre- processing process flow

The input of this process is source code, it will transform into source code units after passed a few processes. There are fives rules in the pre- processing process to achieve the aim. For the first rule is ***PR- 1:*** *Remove namespace and using statements.* The function for this rule is to remove the namespace from the source code and using the statements in the source code.

Secondly, ***PR- 2:*** *Remove comments.* In this rule, it will remove all the comments inside the programming code. In the programming, writing some comments is to help programmer to read or maintenance the function for each line of code. Usually developer put a comment between the source codes. Therefore, this function is to remove the comments that have been written by programmer.

For the third of this process is ***PR- 3:*** *Remove empty lines.* Empty lines is detected if there in no content inside the line. It occurs because programmer not cleaning up their code after finish developing and maintaining the code. In this process, this rule is not importance in detecting the code clone.

Next is ***PR- 4:*** *Regularize function access keyword to void.* In the C language there are many function accesses such as main, void and int. The function of this rule is to regularize all the function accesses into a single function access which is *void.*

Lastly, for the fifth rule is ***PR- 5:*** *Regularize source codes to uppercase.* Usually programmer write a source code with their own style. As an example, some of programmer write 'SearchMenu' and another programmer write 'searchmenu'. As noticed that there are two style of writing for variable 'searchmenu'. For all variable that has same function, the code detection tools should detect this variable as the clone that can produce as a clone in the results. Therefore, to detect different variable that have same function is important to Generic Code Clone Detection tools to get a better result for code clone.

Every output that produced by every process will be used for next process until the code clone result is found. Normalized codes or known as a source unit which is still in the form of source code and represent the function of source code. There are a few parameters to make this process is:

- C application, C1
- Source File, [S_1, S_2, S_3, ….. , S_n]
- Source Code, [SC_1, SC_2, SC_3, … , SC_n]
- Source Unit, [SU_1, SU_2, SU_3, …., SU_n]
- Pre- processing Rule 1, *PR- 1*
- Pre- processing Rule 2, *PR- 2*
- Pre- processing Rule 3, *PR- 3*
- Pre- processing Rule 4, *PR- 4*
- Pre- processing Rule 5, *PR- 5*

For the first process, read the input which is the source code with code SC_1. The source code is from source file (S_1) that is in C application (C_1). Then continue with the five pre- processing rules from *PR- 1, PR- 2, PR- 3, PR- 4* and *PR- 5.* The purpose of the pre- processing rule is to remove the unnecessary component for detect the code clone. After done go through all the pre- processing rule phase, the source unit (SU_1) are produced. Once the process is done for S_1, the remaining process will continue for S_n in C_1. The output for this process will be used for the next phase of Generic Code Clone Detection tools process. The pseudocode of this process will be shown at figure 4.3.

C application, C1

Source File, [S_1, S_2, S_3, ….. , S_n]

Source Code, [SC_1, SC_2, SC_3, … , SC_n]

Source Unit, [SU_1, SU_2, SU_3, …., SU_n]

Pre- processing Rule 1, *PR- 1*

Pre- processing Rule 2, *PR- 2*

Pre- processing Rule 3, *PR- 3*

Pre- processing Rule 4, *PR- 4*

Pre- processing Rule 5, *PR- 5*

1. Read source file S_1 in C_1
2. For each S_1,
3. Check SC_1,
4. For each existing SC_1,
5. Apply PR-1
6. Apply PR-2
7. Apply PR-3
8. Apply PR-4
9. Apply PR-5
10. Repeat on the remaining source code [SC_2, SC_3, .. , SC_n] in S_1
11. Continue step 2 until step 10 on the remaining source files [S_2, S_3, … , S_n] in C_1

Figure 4.3 Pseudocode of Pre- processing process

4.2.1.1 Improvement in Pre- processing Process

Pre-processing is the first process on the generic code clone detection tool. The enhancement of this process is to add more rules for detect the unnecessary component to be detected for code clone detection and to produce a *source unit* that will be used for next process which is transformation process. As for a mention before, this research is to enhance the prototype of code clone program that already make by other researcher which are to detect code clone for java programming language. Table 4.1 is explanation of the rules added to enhance the prototype for C programming language code clone detection.

Table 4.1 Rules added for pre- processing process

Rules	**Unnecessary component added**	
Pre-processing Rule 1 (PR-1) - Remove namespace and using statements	- "namespace" - "using"	
Pre-processing Rule 2 (PR-2) - Remove comments	- " ' "	
Pre-processing Rule 3 (PR-3) - Remove empty lines		
Pre-processing Rule 4 (PR-4) - Regularize function access keyword to public	- "protected" - "friend" - "protectedfriend" - "private" - "shared" - "shadow" - "readonly" - "withevents"	
Pre-processing Rule 5 (PR-5) - Regularize source codes to lowercase	- A = a - B = b - C = c - D = d - E = e - F = f	- N = n - O = o - P = p - Q = q - R = r - S = s

	- G = g - H = h - I = i - J = j - K = k - L = l - M = m	- T = t - U = u - V = v - W = w - X = x - Y = y - Z = z

Besides that, there are categories that are unnecessary to detect the code clone for the programming language of C, some of the rules are already mentioned in the prototype but the rules before are used to detect the code clone for the programming language of Java. The above table shows the rules to be removed, changed or transformed.

4.2.2 Transformation

Transformation process is the second process in the generic code clone detection tool. This process is to transform into a number or measurable units. The source units already produced by the pre-processing process. After transforming the source units, the source units will be used to determine the parameters for the next process. Figure 4.4 shows the diagram of this process.

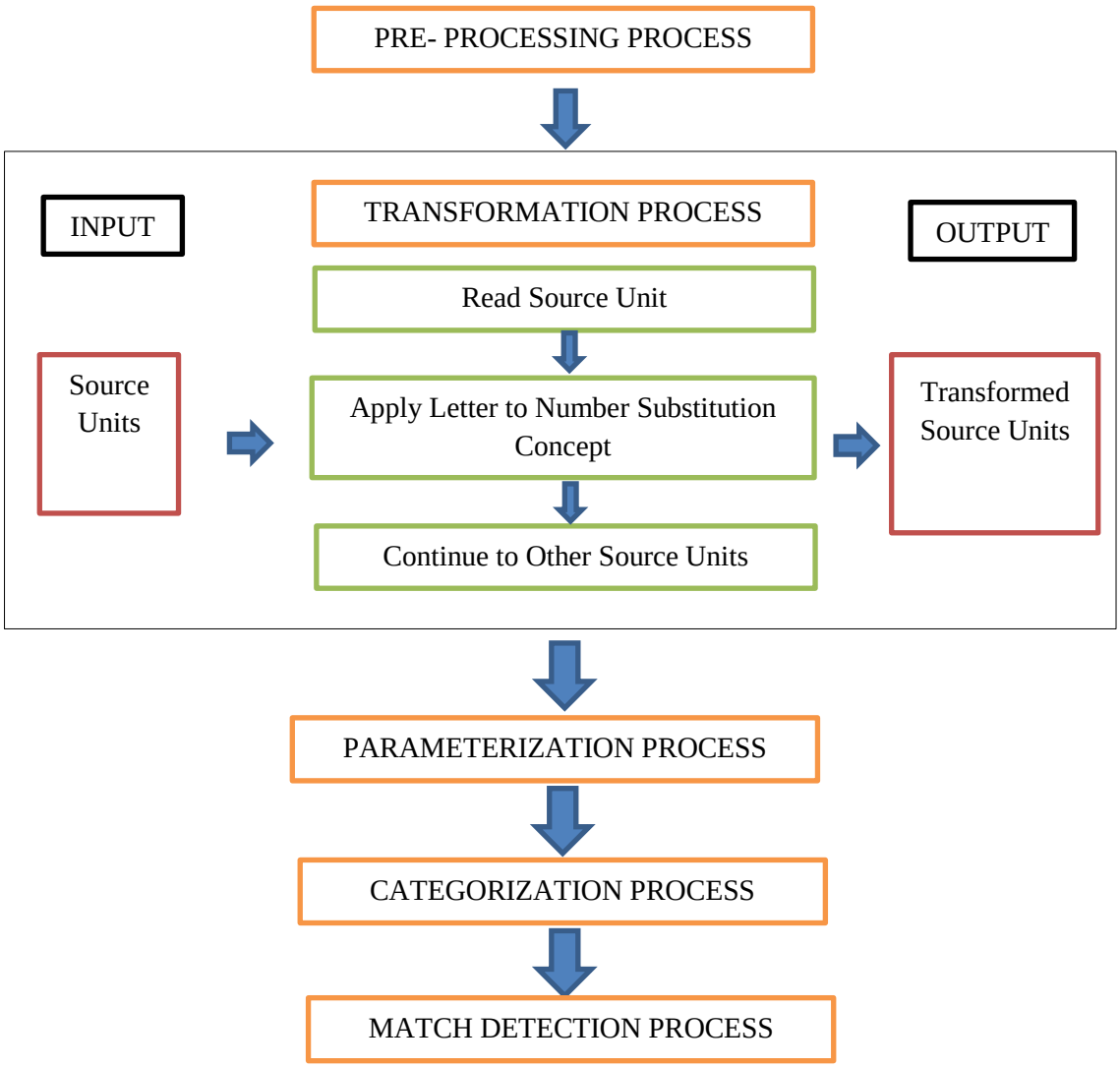


Figure 4.4 Transformation process flow

In the previous process, pre- processing process produced source units. The source unit will be use in this process. The aim of this transform source code into measurable units which is using the letter inside the source code and transform it into substitution concept that transform the letter into a measurable unit.

Table 4.2 Letter to numerical substitution concept value

Letter	Value	Letter	Value
a	01	n	14
b	02	o	15
c	03	p	16
d	04	q	17
e	05	r	18
f	06	s	19
g	07	t	20
h	08	u	21
i	09	v	22
j	10	w	23
k	11	x	24
l	12	y	25
m	13	z	26

The objective of this process is to transform the letter that are from source units into a numerical form. After finish transform the source units into numerical form, they are divided into two group which are *header (h)* and *body (b). Header* refer to the head or beginning of the source unit that already transformed that prior to the body. Meanwhile, *body* is the body of the source unit. In figure 4.5 below shown pseudocode of the transformation process.

Source unit, [SU_1, SU_2, SU_3,…,SU_n]

Transformed source unit, [TSU_1, TSU_2, TSU_3, … , TSU_n]

header [h_1, h_2, h_3, … , h_n]

body [b_1, b_2, b_3, … , b_n]

letter to number substitution, LN, [a=01, b=02, c=03, d=04, e=05, f=06, g=07, h=08, i=09, j=10, k=11, l=12, m=13, n=14, o=15, p=16, q=17, r=18, s=19, t=20, u=21, v=22, w=23, x=24, y=25, z=26]

1. Read a source unit SU_1,
2. For SU_1,
3. Apply LN
4. Save header of source unit SU_1 as h_1
5. Save body of source unit as b_1
6. Continue with step 2 units 5 on the remaining source units [SU_2, SU_3, … SU_n]

Figure 4.5 Transformation process pseudocode

4.3 The Generic Code Clone Detection Prototype

Other researchers are already conducting the GCCD prototype. This research enhances the GCCD tool to detect the code clone in the programming language of C. In addition, the generic code clone detection tools interface will be displayed in Figure 4.6. Improvement or improvement is not an interface effect. The enhancement adds only more rule that the prototype can detect.

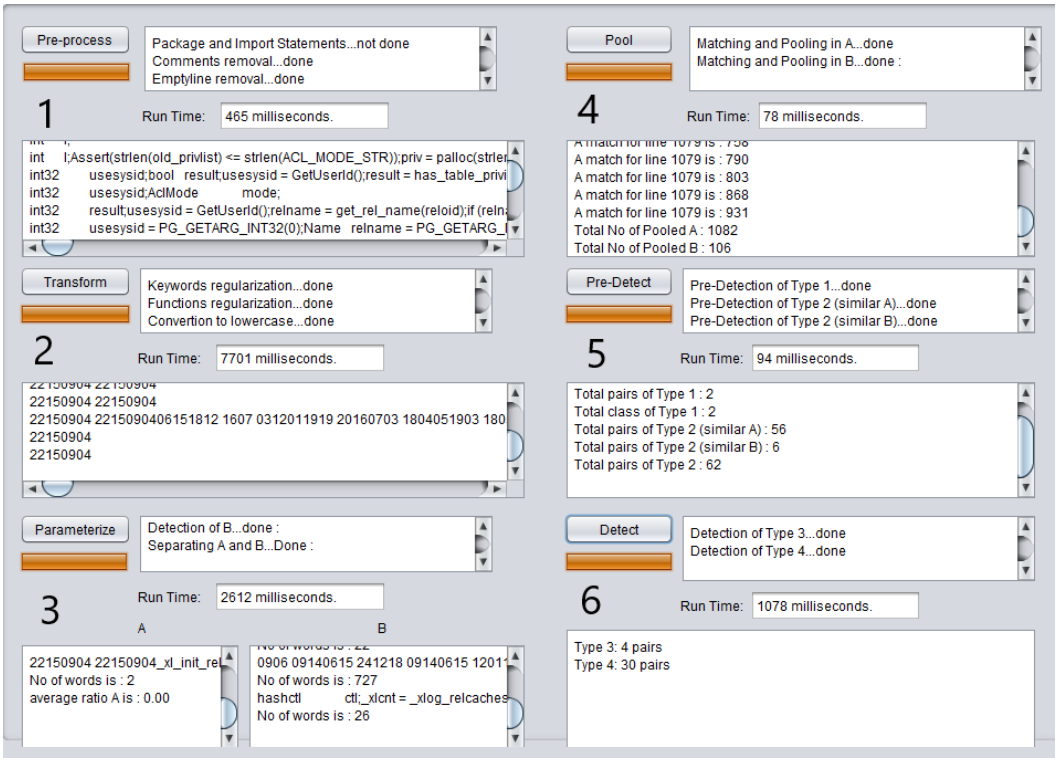


Figure 4.6 Generic Code Clone Detection prototype interface

Except for the process of categorization and match detection, it will be presented on the button for each process in generic code clone detection. The name will change to "Pool" for the categorization process as the output is the pools or groups of code clones. There is also a Pre - detect and Detect button to present the process of match detection. For Type-1 and Type-2 clones, the pre - detection processes the code clone. The Detect button for Type-3 and Type-4 clones will process the code clone. The display of the process and the result is separated, making it easier to compare the result with the existing model. The process time for each process will also be displayed for the Run Time label until the result is obtained.

In the figure 4.6 below start with pre- processing until match detection process as marked the flow from **1** until **6.** The explanation of the mapping prototype will be explained in the table 4.3.

Table 4.3 Mapping of Generic Code Clone Detection to prototype

GCCD Model	**The start of the process**	**Result of the process (Output)**
Pre-processing process	*Pre-process* button	**1**
Transformation process	*Transform* button	**2**
Parameterization process	*Parameterize* button	**3**
Categorization process	*Categorize* button	**4**
Match Detection process	*Pre-detect* and *Detect* button	**5** and **6**

As displayed above, there is a source file was executed using GCCD prototype. For number **1,** represented the pre- processing process. The textbox displayed the source code lines that are used for pre- processing process. The textbox showed all the function that available in the processed the source file.

Advantages

a. Filtering the unnecessary component to be detected as a code clone

b. Faster in detecting a code clone

c. Support all type of code clone

Disadvantages

a. Selection of file type

b. Result of code clone are not shown in a line of code

c. Still in development phase

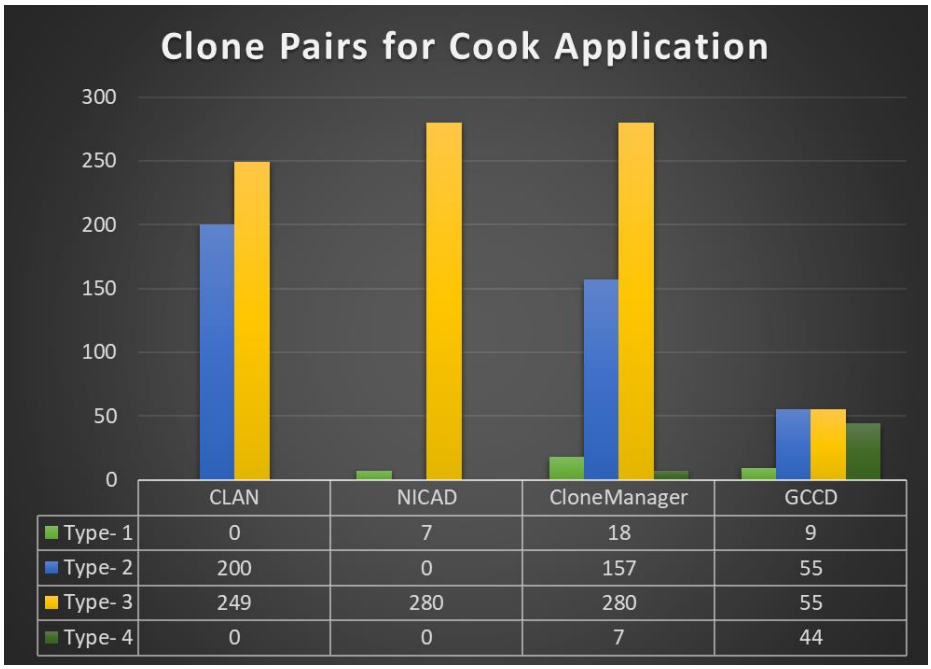
d. Can be used in three (3) programming language: Java, C#. Net and C

4.4 Comparison Result

As we discuss in chapter 3, four dataset from Bellon has been taken for the result of code clone using Generic Code Clone Detection tools. This research focusing on C application. Four dataset are available which are *cook, snns, wetlab* and *postgresql.*

4.4.1 Comparison Code Clone Detection tools for Cook application

Figure 4.7 shows the result for Cook application that researcher compares from another tools detection.



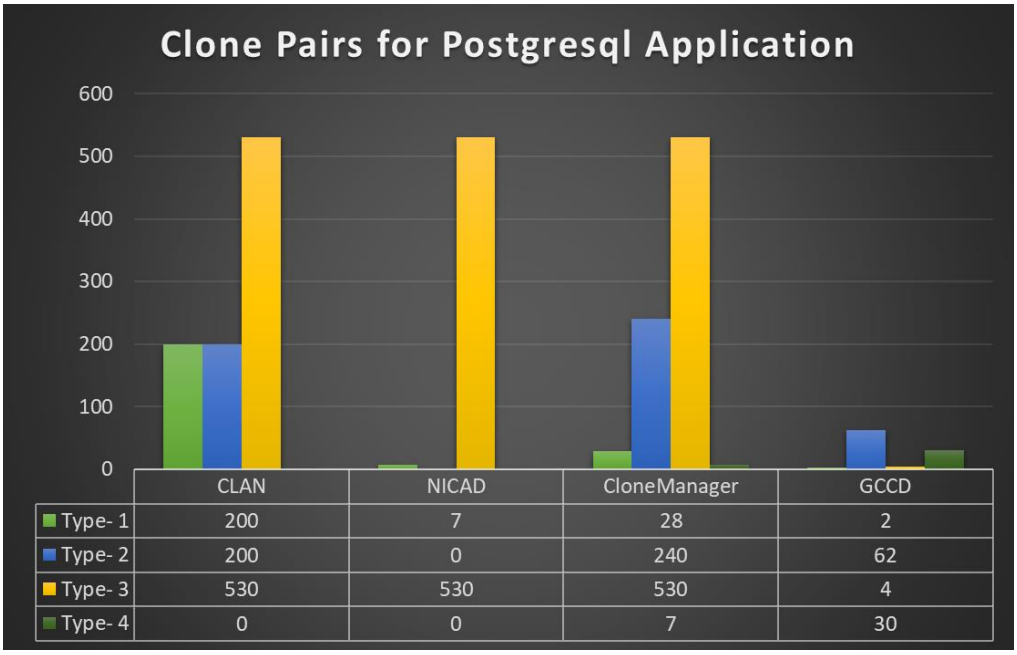
	CLAN	NICAD	CloneManager	GCCD
Type- 1	0	7	18	9
Type- 2	200	0	157	55
Type- 3	249	280	280	55
Type- 4	0	0	7	44

Figure 4.7 Total number of Clone Pair for Cook Application

From the figure above, the result from cook application. The researcher compares the result between three tools which are CLAN, NICAD and CloneManager. For Generic Code Clone Detection tool, the research can the detect the clone from Type- 1 until Type- 4. There are two tools that only can detected the code clone from Type-1 until type- 3 only. From the GCCD the total number of files scanning is 537 files.

4.4.2 Comparison Code Clone tools for Postgresql application

This figure 4.8 above shows the result for Postgresql application. The researcher compares GCCD tool between three tools which are CLAN, NICAD and CloneManager.



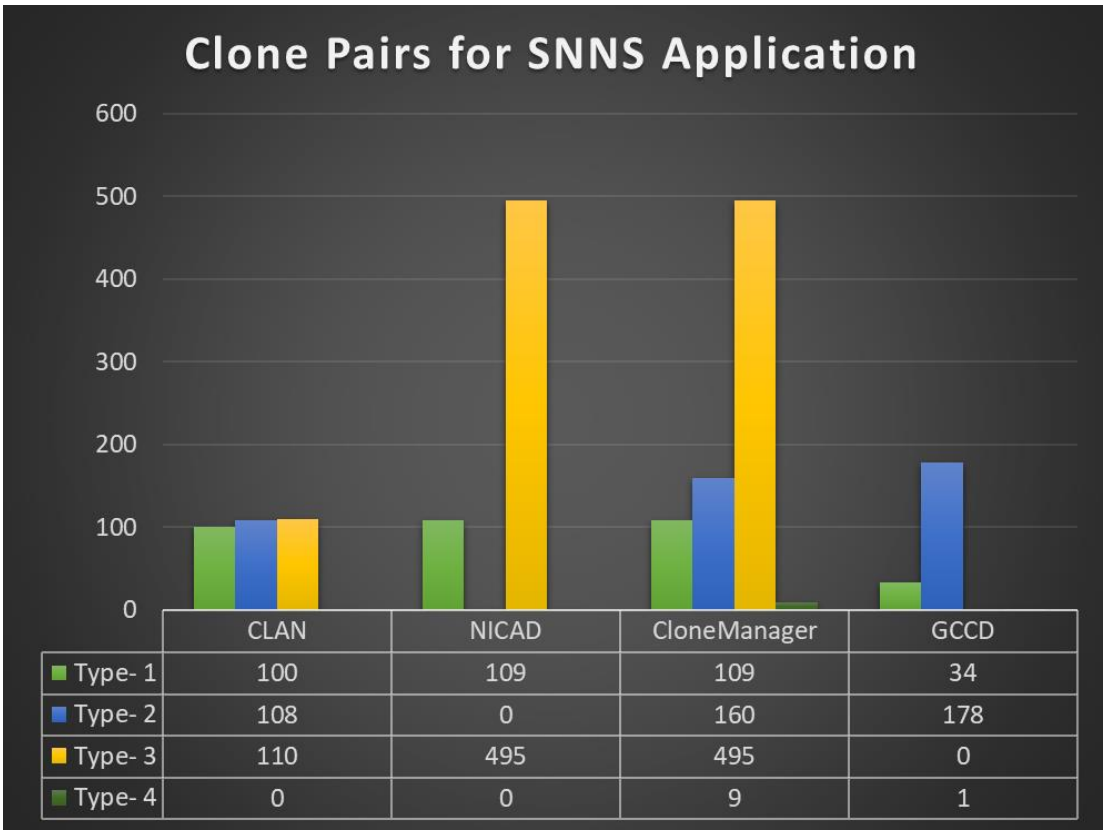
	CLAN	NICAD	CloneManager	GCCD
Type- 1	200	7	28	2
Type- 2	200	0	240	62
Type- 3	530	530	530	4
Type- 4	0	0	7	30

Figure 4.8 Total number of Clone Pair for Postsgresql Application

The result from the figure 4.8 displayed the total number of clone pair for Postsgresql application. The GCCD tools shows the number of clone pair can be detected from Type-1 until Type-4. For this application total files are this tool scanned is 747.

4.4.3 Comparison Code Clone tools for SNNS application

This figure 4.8 above shows the result for SNNS application. The researcher compares GCCD tool between three tools which are CLAN, NICAD and CloneManager.



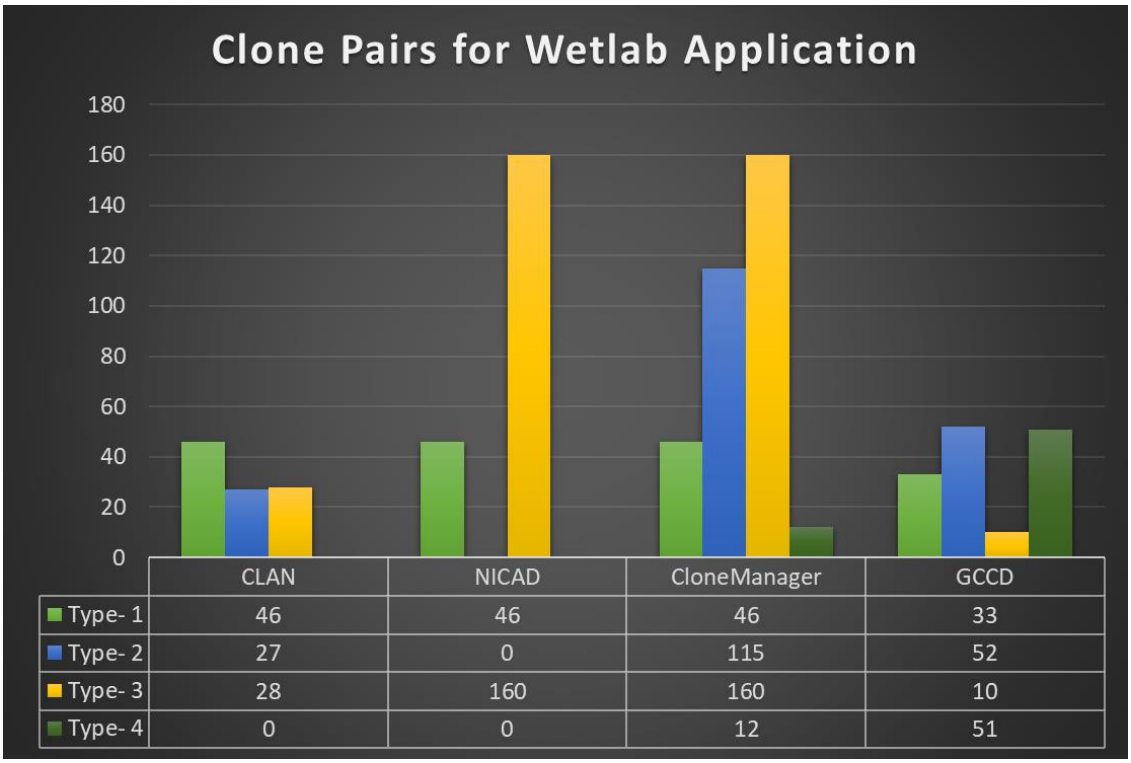
	CLAN	NICAD	CloneManager	GCCD
Type- 1	100	109	109	34
Type- 2	108	0	160	178
Type- 3	110	495	495	0
Type- 4	0	0	9	1

Figure 4.9 Total number of Clone Pair for SNNS Application

The figure 4.9 above shows the total number of Clone pair for SNNS application. The number of file scanning for this application is 557. From the result of the GCCD tools it detects a higher number of clones in Type- 2 which is 178. For CLAN and NICAD tools, these tool only can detect until Type- 3 clone pair.

4.4.4 Comparison Code Clone tools for Wetlab application

This figure 4.8 above shows the result for Wetlab application. The researcher compares GCCD tool between three tools which are CLAN, NICAD and CloneManager.



	CLAN	NICAD	CloneManager	GCCD
Type- 1	46	46	46	33
Type- 2	27	0	115	52
Type- 3	28	160	160	10
Type- 4	0	0	12	51

Figure 4.10 Total number of Clone Pair for Wetlab Application

The Figure 4.10 showed the total number of Clone Pair for Wetlab Application. Due to the result GCCD tools give a 33 number for Type-1 followed by 52 for Ttpe-2 and for Type- 3 has 10 clone pair. Lastly for Type-4 the number of clone pair is 51.

4.5 Summary

As a summary of this chapter, the improvement in the detection of generic code clones was made to fulfil this research's objective. There are five (5) processes to detect the code clone in the detection of generic code clones, the improvement is made only at two (2) processes at the beginning of the entire process which is pre-processing and transformation. Pre-processing process involves removing some unnecessary component to be detected as a code clone and converting all code into a lower case. The second process that made the improvement is the process of transformation. This process is aimed at converting the code into something that can be measured by changing the alphabet into a number. The GCCD can detect the code clone for the C application platform that is the main objective of this research as a result of this evaluation.

CHAPTER 5

5.1 Overview

In this chapter will explain in detail every objective of the research. In addition, the recommend suggestions of future work for improving the Generic Code Clone Detection tools.

5.2 Objective Revisited

This research aims at enhancing the existing GCCD (Generic Code Clone Detection) code detection tools. This code clone detection tool has the advantage of detecting all kinds of code clones (Type-1, Type-2, Type-3 and Type-4). The other tools that another research or company has already developed are detected only until type-3. The tools capable of detecting the code clone up to type-4 must make some payment. And this researcher's scope is to use the free source that the internet can provide. The generic code clone detection that is still under development process is to be added to this software, which is the rule of another programming language.

The first objective of the research is to enhance the rules in the process of pre - processing and transformation in the model of generic code clone detection for C code detection. Only 2 processes at the beginning to enhance from 5 processes in GCCD, because the rule to be added is only in these two processes, the rest of the process is the software's main function, that part of the process cannot be changed or altered to avoid software error or accidentally change the software's structure. Researchers need to conduct a research or study on the C programming structure to enhance the rules. This helps the researcher to find out what kind of uninterested things in the code clone detection process are not to be detected. For more details, this explanation is already discussed in Chapter 4.

Continue with the second objective, by modifying the existing prototype to develop the improvement. The prototype runs on the programming language of java. The researcher needs to learn a little bit of java programming language to avoid the mistake of changing or modifying the prototype. NetBeans is the software used by the researcher to modify the existing prototype. The prototype that recommends by developer for this software. In enhancing the prototype, there are some problems that researcher must ask the prototype developer's suggestion for the problem.

The last goal that the researcher needs to achieve is to evaluate the enhanced prototype of the generic code clone detection tool model using the data set using the code clone detection result. This goal is to run the prototype without any error. The prototype must be tested with the dataset to prove that the researcher is successful in adding the rule of C application. There are four datasets of the cook, snns, postgresql, and, wetlab for C application. The prototype runs smoothly without any error from all the dataset that tested with this prototype. From the result, this research fulfils the goal of detecting the C application code clone. The outcome of detection of code clones is discussed in Chapter 4

5.3 Recommendation for Future Work

A lot of programming language is currently being used by developers to develop the software or system. Another programming language can be added to improve the generic code clone detection prototype for future work. This prototype helps the developer detect the clone of code that may affect software or system performance. It is also easier for the software or system to maintain and evolve without the code clone inside the code.

Another possible improvement is to locate the code clone detail. The maintenance clone in which part of their code needs to be known. This enhancement will help the programmer decide to make a code change that is detected as a code clone. Otherwise, this improvement can make the generic code detection tool prototype more functional, better, and can be a reference for other code clone detection tools evolution.

The future work can be done in terms of interface by enhancing the interface for this prototype. Some option at the top can be added. Also, a single click button can automate the step for each process, but the result still shows the same as before.

REFERENCES

Al-Fahim. (2015). Generic Code Clone Detection Model for Java Applications, (August).

Baker, B. S. (1995). On Finding Duplication and Near-Duplication in Large Software Systems.

Basit, H. A., Rajapakse, D. C., & Jarzabek, S. (2005). Beyond templates: a study of clones in the STL and some general implications. *Proceedings of the 27th International Conference on Software Engineering*, 451–459. https://doi.org/10.1145/1062455.1062537

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)*, 368–377. https://doi.org/10.1109/ICSM.1998.738528

Bellon, S., Koschke, R., Antoniol, G., Krinke, J., & Merlo, E. (2007). Comparison and Evaluation of Clone Detection Tools, *33*(9), 577–591.

Biegel, B., & Diehl, S. (2010). Highly configurable and extensible code clone detection. In *Proceedings - Working Conference on Reverse Engineering, WCRE* (pp. 237–241). https://doi.org/10.1109/WCRE.2010.34

Cordy, J. R., & Roy, C. K. (2011). The NiCad clone detector. In *IEEE International Conference on Program Comprehension* (pp. 219–220). https://doi.org/10.1109/ICPC.2011.26

Dang, Y., Zhang, D., Ge, S., Huang, R., Chu, C., & Xie, T. (2017). Transferring code-clone detection and analysis to practice. *Proceedings - 2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track, ICSE-SEIP 2017*, 53–62. https://doi.org/10.1109/ICSE-SEIP.2017.6

Duala-Ekoko, E., & Robillard, M. P. (2007). Tracking code clones in evolving software. In *Proceedings - International Conference on Software Engineering* (pp. 158–167). https://doi.org/10.1109/ICSE.2007.90

Duala-Ekoko, E., & Robillard, M. P. (2008). Clonetracker: Tool Support for Code Clone Management. In *Proceedings of the 13th international conference on Software engineering - ICSE '08* (p. 843). https://doi.org/10.1145/1368088.1368218

Giesecke, S. (2007). Generic modelling of code clones. *Duplication, Redundancy, and Similarity in Software*, (06301), 1–23.

Göde, N., & Koschke, R. (2009). Incremental clone detection. In *Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR* (pp. 219–228). https://doi.org/10.1109/CSMR.2009.20

Harder, J. (2013). The Limits of Clone Model Standardization, 10–11.

Higo, Y., & Ueda, Y. (2007). Simultaneous Modification Support based on Code Clone Analysis, 262–269. https://doi.org/10.1109/ASPEC.2007.44

Jiang, L., Misherghi, G., & Su, Z. (2007). D ECKARD : Scalable and Accurate Tree-based Detection of Code Clones ∗, (0520320).

Kamiya, T., Kusumoto, S., & Inoue, K. (2002a). CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code CCFinder: A Multilinguistic Token-Base Code Clone Detection System for Large Scale Source Code. *Ieee Transactions on Software Engineering*, *28*(28), 654–670.

Kamiya, T., Kusumoto, S., & Inoue, K. (2002b). Review of CCFinder : A Multi-linguistic Token-Based Code Clone Detection System for Large Scale Source Code, *28*(7), 3.

Kapser, C. J., Harder, J., & Baxter, I. (2012). A common conceptual model for clone detection results. *2012 6th International Workshop on Software Clones, IWSC 2012 - Proceedings*, 72–73. https://doi.org/10.1109/IWSC.2012.6227870

Kodhai, E., & Kanmani, S. (2014). Method-level code clone detection through LWH (Light Weight Hybrid) approach. *Journal of Software Engineering Research and Development*, *2*(1), 12. https://doi.org/10.1186/s40411-014-0012-8

Komondoor, R., & Horwitz, S. (2001). Using Slicing to Identify Duplication in Source Code, 40–56. https://doi.org/10.1007/3-540-47764-0_3

Kontogiannis, K. A., Demori, R., Merlo, E., Galler, M., & Bernstein, M. (1996). Pattern matching for clone and concept detection. *Automated Software Engineering*, *3*(1–2), 77–108. https://doi.org/10.1007/BF00126960

Krinke, J. (2001). Identifying similar code with program dependence graphs. *Proceedings Eighth Working Conference on Reverse Engineering*, 301–309. https://doi.org/10.1109/WCRE.2001.957835

Lavoie, T., Eilers-Smith, M., & Merlo, E. (2010). Challenging cloning related problems with GPU-based algorithms. *Proceedings of the 4th International Workshop on Software Clones - IWSC '10*, 25–32. https://doi.org/10.1145/1808901.1808905

Liu, C., Chen, C., Han, J., & Yu, P. S. (2006). GPLAG: detection of software plagiarism by program dependence graph analysis. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 872–881. https://doi.org/10.1145/1150402.1150522

Livieri, S., Higo, Y., Mazushita, M., & Inoue, K. (2007). Very-Large Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder. *Proceedings of the 29th International Conference on Software Engineering (ICSE '07)*.

Mayrand, Leblanc, & Merlo. (1996). Experiment on the automatic detection of function clones in a software system using metrics. *Proceedings of International Conference on Software Maintenance ICSM-96*, 244–253. https://doi.org/10.1109/ICSM.1996.565012

Morshed, M., Rahman, M., & Ahmed, S. (2012). A Literature Review of Code Clone Analysis to Improve Software Maintenance Process. *ArXiv Preprint ArXiv:1205.5615*.

Nguyen, T. T., Nguyen, H. A., Pham, N. H., Al-Kofahi, J. M., & Nguyen, T. N. (2009). ClemanX: Incremental clone detection tool for evolving software. *2009 31st International Conference on Software Engineering - Companion Volume, ICSE 2009*, 437–438. https://doi.org/10.1109/ICSE-COMPANION.2009.5071050

Perumal, A., Kanmani, S., & Kodhai, E. (2010). Extracting the similarity in detected software clones using metrics. *2010 International Conference on Computer and Communication Technology, ICCCT-2010*, 575–579. https://doi.org/10.1109/ICCCT.2010.5640465

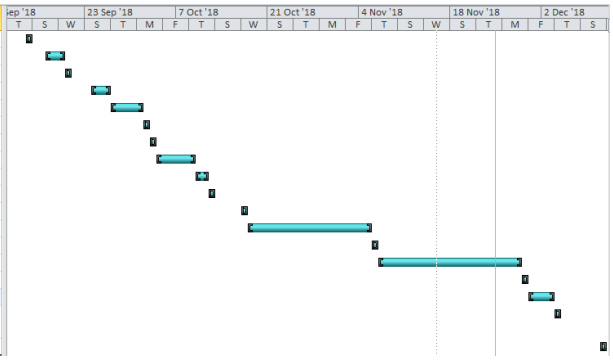
Rattan, D., Bhatia, R., & Singh, M. (2013). *Software clone detection: A systematic review*. *Information and Software Technology* (Vol. 55). Elsevier B.V. https://doi.org/10.1016/j.infsof.2013.01.008

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

Sasaki, Y., Yamamoto, T., Hayase, Y., & Inoue, K. (2010). Finding file clones in FreeBSD ports collection. *Proceedings - International Conference on Software Engineering*, 102–105. https://doi.org/10.1109/MSR.2010.5463293

APPENDIX A

Task Name	Duration	Start	Finish
Meeting with Supervisor	1 day	Fri 14/9/18	Fri 14/9/18
Planning Research Flow	3 days	Mon 17/9/18	Wed 19/9/18
Meeting with Supervisor	1 day	Thu 20/9/18	Thu 20/9/18
Documenting Chapter 1	3 days	Mon 24/9/18	Wed 26/9/18
Correction Chapter 1	3 days	Thu 27/9/18	Mon 1/10/18
Submission Chapter 1	1 day	Tue 2/10/18	Tue 2/10/18
Meeting with Supervisor	1 day	Wed 3/10/18	Wed 3/10/18
Documenting Chapter 2	4 days	Thu 4/10/18	Tue 9/10/18
Correction Chapter 2	2 days	Wed 10/10/18	Thu 11/10/18
Submission Chapter 2	1 day	Fri 12/10/18	Fri 12/10/18
Meeting with Supervisor	1 day	Wed 17/10/18	Wed 17/10/18
Research Design	13 days	Thu 18/10/18	Mon 5/11/18
Meeting with Supervisor	1 day	Tue 6/11/18	Tue 6/11/18
Correction Chapter 3	16 days	Wed 7/11/18	Wed 28/11/18
Submission Chapter 3	1 day	Thu 29/11/18	Thu 29/11/18
Final Correction	2 days	Fri 30/11/18	Mon 3/12/18
Submit Final Report PSM 1	1 day	Tue 4/12/18	Tue 4/12/18
PSM 1 Evaluation	1 day	Tue 11/12/18	Tue 11/12/18



Task Name	Duration	Start	Finish
Improvement, Implementation and Evaluation of GCCD	30 days	Tue 29/1/19	Mon 11/3/19
Generic Code Clone Detection Prototype	20 days	Tue 12/3/19	Mon 8/4/19
Analysis with Existing Code Clone Detection Tools	10 days	Tue 9/4/19	Mon 22/4/19
Meeting with Supervisor	1 day	Thu 25/4/19	Thu 25/4/19
Testing GCCD prototype	7 days	Fri 26/4/19	Mon 6/5/19
Documenting chapter 4 & 5	20 days	Tue 7/5/19	Mon 3/6/19
Meeting with Supervisor	1 day	Wed 5/6/19	Wed 5/6/19
Correction Chapter 4 & Chapter 5	15 days	Thu 6/6/19	Wed 26/6/19
Submit Documentation Chapter 4 & 5	1 day	Thu 27/6/19	Thu 27/6/19
Documenting Thesis Report	8 days	Mon 1/7/19	Wed 10/7/19
Submission Thesis Report	1 day	Thu 11/7/19	Thu 11/7/19