Implementing a 3-Way Approach of Clone Detection and Removal using Pattern & Clone Detector Tool

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

Master of Engineering
in
Software Engineering

Submitted By
Ginika Mahajan
(801031008)

Under the supervision of
Ms. Ashima Singh
Assistant Professor
CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2012

Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Implementing a 3-Way Approach of Clone Detection and Removal using Pattern & Clone Detector Tool*", in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Ms. Ashima Singh* and refers other researcher's work which are duly listed in the reference section.

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

(Gunika Mahajan)
801031008

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

(Ms. Ashima Singh)
Assistant Professor,
Computer Science and Engineering Department,
Thapar University,
Patiala.

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department,
Thapar University,
Patiala.

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

Acknowledgement

I express my gratitude and appreciation to all those who have helped me throughout the duration of my research work. It would not have been possible to complete this research without the guidance and the help of several individuals who in one way or another contributed and extended their valuable assistance in the preparation and completion of this study.

At this moment of accomplishment, first of all I would like to articulate thanks for support of my friends Arpita Sharma and Meena Bharti for their kind support and help. I pay homage to my guide, Ms. Ashima Singh for her personal support and great patience at all times. It gives me immense pleasure to pay my gratitude for her valuable advice, constructive criticism, and expensive discussion around my work.

I am also thankful to Dr. Maninder Singh, Head, Computer Science and Engineering Department, for his kind help and cooperation. Also I am thankful to Mr. Karun Verma and Mr. Sumit Miglani, PG Coordinator.

I would also like to thank all the staff members of Computer Science and Engineering Department for providing me all the facilities required for the completion of my thesis work.

Last but not the least; I want to acknowledge the contributions of my parents, for their constant motivation, inspirations and for supporting me spiritually throughout my life, and the one above all of us, the omnipresent God, for giving me the strength to complete this work.

(Gini ka Mahajan)
804031008

Abstract

Software Systems are evolving by adding new functions and modifying existing functions over time. Through the evolution, the structure of software is becoming more complex and so the understandability and maintainability of software systems is deteriorating day by day. These are not only important but one of the most expensive activities in software development.

The copying and duplication of source code has been studied in software engineering under several topic areas. Copy and paste programming is a common activity but it introduces a negative point to reuse by creating Clones. Detection of duplicate code fragments leads to efficiency on the software maintenance process and decreases maintenance cost. It is possible to outwit the hindrance of clones by applying a 3-way approach of detecting the clones at design and code level. The process is automated by developing a tool that requires no parsing yet is able to detect a significant amount of code duplication.

Since code clones are believed to increase the maintenance effort, several code clone detection techniques and tools have been proposed. This thesis proposes a 3-way approach of integrating Model Based Visual Analysis using UML, Pattern Based Semantic Analysis and Syntactically Code Analysis to detect Type1 and Type 2 clones using Pattern Clone Detector (P C Detector).

For the implementation of 3-way approach, I have developed a tool, named Pattern Clone Detector (PC Detector) that extracts code clones in C and C++ source files. After detecting the clones, PC Detector also identifies the pull up pattern residing in the code and thereby removes the clones. The proposed 3-way approach is compared with other clone detection techniques. The other studied clone detection techniques detect clones at code level but the proposed 3-way approach detects and removes cloning at design as well as code level.

Table of Contents

Certificate		**i**
Acknowledgement		**ii**
Abstract		**iii**
Table of Contents		**iv**
List of Figures		**vii**
List of Snapshots		**x**
Chapter 1	**Introduction**	**1-7**
1.1	Software Reuse	1
1.2	Software Cloning	2
1.3	Refactoring	4
1.4	Need of Software Cloning	5
1.5	Scope of Cloning	6
1.6	Objective of Thesis	6
1.7	Organisation of Thesis	7
Chapter 2	**Literature Survey**	**8-25**
2.1	Basic concepts of Clone Detection	8
	2.1.1 Cloning Terminologies	8
	2.1.2 Reason of Software Cloning	10

	2.1.3 Types of Clones	12
	2.1.4 Advantages of Clone Detection	14
	2.1.5 Software Clone Analysis	14
	2.1.6 Overview of Clone Detection Techniques	15
2.2	Design Model (UML) Based Clone Detection	16
	2.2.1 Design Model Based Development	16
	2.2.2 Definition of Model Clones	18
	2.2.3 Types of Model Clones	19
2.3	Clone Detection with Refactoring Patterns	19
	2.3.1 Refactoring	19
	2.3.2 Refactoring Patterns	20
	2.3.2.1 Extract Pattern	21
	2.3.2.2 Pull Up Pattern	21
	2.3.2.3 Template Pattern	22
	2.3.2.4 Strategy Pattern	23
2.4	Cloning and Software Maintenance	24
Chapter 3	**Problem Statement**	**26-27**
Chapter 4	**Proposed Model and Implementation**	**28-53**
4.1	Proposed Model	28
4.2	Model based Visual Analysis through UML	29
	4.2.1 Case Study: Hospital Management System(HMS)	30

	4.2.2 Use-Case Diagram	31
	4.2.3 Class Diagram	32
	4.2.4 Activity Diagram	33
4.3	Pattern-based Semantic Analysis	35
	4.3.1 Flow Chart of Pattern Detection Process	35
	4.3.2Algorithm of Pattern Detection Process	36
4.4	Syntactic Detection of Clones	36
	4.4.1 Flaw in Type 2 Clones	36
	4.4.2 Flow Chart of Clone Detection Process	38
	4.4.3 Algorithm of Clone Detection Process	39
4.5	PC Detector: Implementation and Working	39
	4.5.1 Pattern Detection Process	40
	4.5.2 Clone Detection Process	41
4.6	Tool Snapshots	44
	4.6.1 Start up Form of PC Detector	44
	4.6.2 Main Form of PC Detector	44
	4.6.3 Pull Up Pattern Module	45
	4.6.3.1 Input File	46
	4.6.3.2 Output File	48
	4.6.4 Clone Detection Module	49
	4.6.5 Single File Form	50

	4.6.5.1 Type 1 Clones	50
	4.6.5.2 Type 2 Clones	51
	4.6.6 Clone Detection in Two Files	53
Chapter 5	**Conclusion and Future Scope**	**54-55**
5.1	Conclusion	54
5.2	Future Scope	55
	References	**56-61**
	List of Publications	**62**

List of Figures

S. No.	Description	Page No.
Figure 1	Example of Cloning	2
Figure 2	Source Code with its Clones	4
Figure 3	Clone Relationship	9
Figure 4	Code Example showing Clone pair and Clone Class	10
Figure 5	Reasons of Cloning	11
Figure 6	Types of Clone	13
Figure 7	Simple copy/ paste or editing operations can create clones in models	17
Figure 8(a)	Unfinished modeling	18
Figure 8(b)	Finished clone free modeling	18
Figure 9	Advantage of Refactoring Patterns	20
Figure 10	Example of Extract Method	22
Figure 11	Example of Pull Up Method	22
Figure 12	Example of Template Pattern	23
Figure 13	Example of Strategy Pattern	24
Figure 14	A 3-way Approach for Clone Detection and Removal with PC Detector	29
Figure 15(a)	Use case diagram showing Unfinished model	31

Figure 15(b)	Use case diagram showing Finished model	31
Figure 16(a)	Class Diagram showing Unfinished model	32
Figure 16(b)	Class Diagram showing Finished model	32
Figure 17(a)	Activity diagram showing Unfinished model	33
Figure 17(b)	Activity diagram showing Finished model	34
Figure 18	Flowchart of Pattern Detection	35
Figure 19	Example explaining the flaw of Type 2 clones with respect of data types	37
Figure 20	Flowchart of Clone detection Type1 and Type 2	38
Figure 21	Example of Pull Up Pattern detection	40
Figure 22	Sample codes	41
Figure 23	Normalized code	41
Figure 24	Filtered Code	42
Figure 25	Example of Tokenization	42
Figure 26	Tokenized Code	43
Figure 27	Matrix Representing the clones present in two files	43

List of Snapshots

S. No.	Description	Page No.
1	Start up Form	44
2	Main Form	44
3(a)	Pull Up Pattern Input	45
3(b)	Pull Up Pattern Output	45
4	Clone Detection Module	49
5	Clone Detection in Single and in Two files	49
6	Single File Form	50
7	Type 1 Clone	50
8	Type 1 Clone Input Source Code	51
9	Type 1 Clone Source code converted to Normalized code	51
10	Type 1 Clone Normalized code converted to Filtered code	52
11	Type 1 Clone Filtered code converted into Tokens	52
12	Matrix Representation of Clones	53
13	Clone Detection in Two Files	53

Chapter 1

Introduction

This chapter introduces a description of the work presented in thesis. It gives a brief introduction of Software Reuse, Software Cloning and concepts, Refactoring Patterns, Need of Cloning, objective, scope and organization of thesis.

Cloning at design as well as at code level is seemed as an obstacle in software development activities and so it needed to be removed. Cloning has grown as an active area in software engineering research community yielding numerous techniques, various tools and other methods for clone detection and removal.

1.1 Software Reuse

Software reuse reduces software development and maintenance costs in the process of creating software systems. It is the likelihood a segment of source code that can be used again to add new functionalities with slight or no modification. Reusable modules and classes reduce implementation time, increase the likelihood that prior testing and use has eliminated bugs and localizes code modifications when a change in implementation is required. Basically, the reuse of a software artifact is its integration into another context. The purpose of reuse is to reduce cost, time, effort, and risk; and to increase productivity, quality, performance, and interoperability [18]. One form of reuse is to copy-paste the code which results in duplication of code i.e. clones.

Software Reuse is defined as the “the process of creating software systems from existing software systems” [9]. This notion is shared worldwide that it’s easier to modify the existing software than developing programs from the scratch and so reuse is emerging as a central theme within Software Engineering. The use of existing components is done basically with the activity of copy and paste.

Software Reuse is using existing software artifacts during the construction of a new software system. The types of artifacts that can be reused are not limited to source code fragments but

rather may include design structures, module-level implementation structures, specifications, documentation, transformations, and so on [9].

1.2 Software Cloning

The copying of code has been studied within software engineering mostly in the area of clone analysis. Software clones are regions of source code which are highly similar; these regions of similarity are called clones, clone classes, or clone pairs. While there are several reasons why two regions of code may be similar, the majority of the clone analysis literature attributes cloning activity to the intentional copying and duplication of code by programmers [5]; clones may also be attributable to automatically generated code, or the constraints imposed by the use of a particular framework or library [30]. In addition to these, some other issues, including programmers' behavior such as laziness and the tendency to repeat common solutions, technology limitations, code understandability and external business forces have influences on code cloning [26]. Cloning is the unnecessary duplication of data whether it is at design level or at coding level.

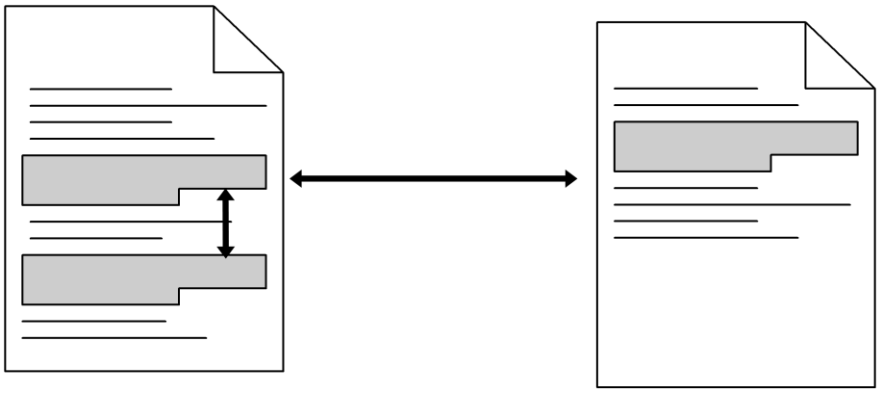


Figure 1: Example of Cloning

Cloning works at the cost of increasing lines of code without adding to overall productivity. Same software bugs and defects are replicated that reoccurs throughout the software at its evolving as well its maintenance phase. It results to excessive maintenance costs as well. So cut paste programming form of software reuse deceivingly raise the number of lines of code without expected reduction in maintenance costs associated with other forms of reuse. So, to refactor code clones, is a promising way to reduce the maintenance cost in future.

Software clones are important aspects in software evolution. If a system is to be evolved, its clones should be known in order to make consistent changes. Cloning is often a strategic means for evolution. For instance, copies can be made to create a playground for experimental feature evolution, where modifications are made in cloned code of a mature feature reducing the risk to break stable code. Once stable, the clone can replace its original. Often, cloning is the start of a new branch of evolution if the changes in the cloned code are not merged back to the main development branch [39]. Clone detection techniques play an important role in software evolution research where attributes of the same code entity are observed over multiple versions.

The reasons why programmers duplicate codes are manifold and include the following reasons [32]:

i. Making a copy of a code fragment is simpler and faster than writing the code from scratch. In addition, the fragment may already be tested so the introduction of a bug seems less likely.
ii. Evaluating the performance of a programmer by the amount of code he or she produces gives a natural incentive for copying code.
iii. Efficiency considerations may make the cost of a procedure call or method invocation seems too high a price. In industrial software development contexts, time pressure together with first and second points lead to plenty of opportunities for code duplication.

According to the definition of cloning, there can be different notions of similarity. They can be based on text, lexical or syntactic structure as shown in figure 2 or can be semantics, model based, or functionally. They can even be similar if they follow the same pattern, that is, the same building plan. Instances of design patterns and idioms are similar in that they follow a similar structure to implement a solution to a similar problem. Semantic similarity relates to the observable behavior. A piece of code, *A*, is semantically similar to another piece of code, *B*, if *B* subsumes the functionality of *A* [39].

This phenomenon occurs similarly in models, suggesting that model clones are as detrimental to model quality as they are to code quality. However, programming language code and visual models have significant differences that make it difficult to directly transfer notions and algorithms developed in the code clone arena to model clones. The structural clone analysis

extends the benefits of analysis based on simple clones in the areas of program understanding, maintenance, reuse, and refactoring [40].

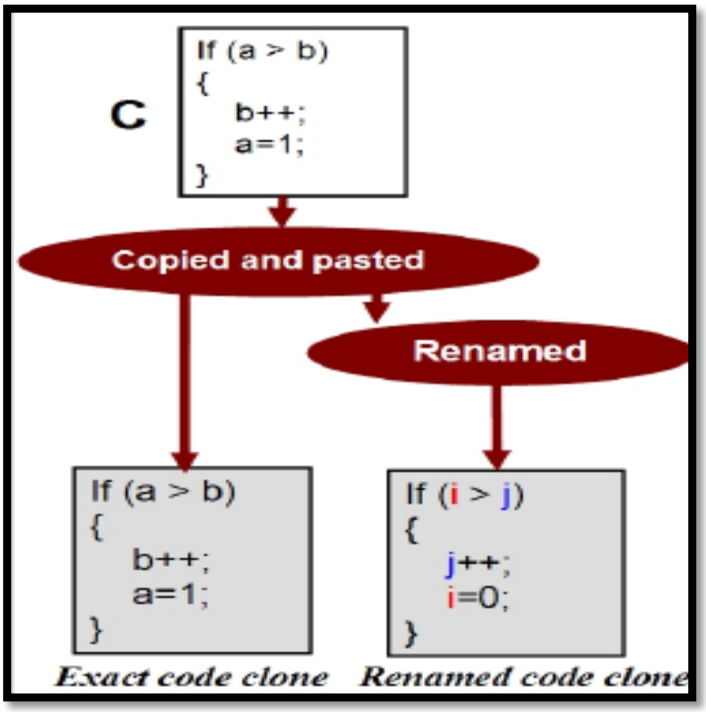


Figure 2: Source Code with its Clones

1.3 Refactoring

Refactoring has as many definitions as practitioners, but perhaps the most concise and certainly the most widely cited definition is as follows: “A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior.” [26]. A more practical-minded definition is, “a technique in which a software engineer applies well-defined source-level transformations with the goal of improving the code's structure and thus reducing subsequent costs of software evolution." [34].

The concept of refactoring (and also the word “refactoring” itself) was coined already several years ago, but its breakthrough came with the integration of refactoring into the software development process Extreme Programming [21]. Refactoring involves restructuring the system by removing duplication, improving communication, simplifying and adding flexibility but without changing the functionality of the program

Refactoring is used in the context of agile development and reengineering. Agile development processes such as Extreme Programming (XP) can rely on refactoring because of their short iterative development cycles [10]. For the same reason, refactoring does not fit very well in the linear waterfall or incremental spiral models of software engineering [11].

In particular, refactoring may not always improve software with respect to clones for two reasons. First, many code clones exist in the system for only a short time; extensive refactoring of such short-lived clones may not be worthwhile if they are likely diverge from one another very soon. Second, many clones, especially long-lived clones that have changed consistently with other elements in the same group, are not easily refactorable due to programming language limitations [23]. These insights show that refactoring will not help in dealing with some types of clones and open up opportunities for complementary clone maintenance tools that target these other classes of clones.

1.4 Need of Software Cloning

The copying and duplication of source code has been studied in software engineering under several topic areas. Copy and paste programming is a common activity but it introduces a negative point to reuse by creating clones. Cloning, the copying and duplicating of blocks of code, is the basic means of software reuse [25]. The most prominent research area within software engineering which studies the duplication of source code is clone analysis; other treatments of code copying and duplication include studies of programmer behavior, code plagiarism detection algorithms, the post-modern programming movement, as well as the development of some specialized programming languages. Clone analysis is, however, the largest area of research related to code duplication.

During the software development cycle, code cloning is easy and inexpensive (in both effort and money). However, this cloning practice can complicate software maintenance and it has been suggested that too much cloned code is a risk, albeit the practice itself is not generally considered harmful [37]. Not only it effects the maintenance phase but also leads to various problems like clones increases Resource Requirements, increases Defect Probability and also increases the Probability of Bad Design.

1.5 Scope of Cloning

Today various programming methodologies are being used in the software development process. The practice of copy and paste code is extensively acknowledged but is rarely explicitly accounted for in models of software development. The software industry seems to be embracing yet another change in the way it does business. Because of their emphasis on agility and time-to-market, many software shops have made the move to extreme programming and agile methods [38]. To implement these methods adherents embrace XP practices like pair programming, refactoring and collective code ownership to generate their products. These releases, which are working versions of the product, not prototypes, are used to demonstrate functions and features to stakeholders who help shape their form through refactoring and continuous integration. Programming methodology is accompanied with high degree of reuse. Refactoring is one of the main practices of Extreme Programming and thus refactoring is used in the cloning process. Implementing refactoring patterns and detecting clones helps in improving the code and removing the clones.

Code clones are considered harmful in software development, also provides hindrance in software evolution and maintenance phase. So various techniques are used to detect them and remove them from the software. One of the approaches is to try to eliminate them through refactoring. Various refactoring patterns are used to detect the clones and hence remove them. Other techniques are also used to eliminate them.

1.6 Objective of Thesis

Code clone detection could be useful in many ways e.g. decreasing the cost of software maintenance activities. Detection of duplicate code fragments increases understandability of software systems and may help system maintainers to increase code quality of the existing system.

Detection of duplicate code fragments leads to efficiency on the software maintenance process and decreases maintenance cost [7]. The aim of this thesis is to design and implement a tool for detecting patterns and clones and removing them.

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

i. To study and compare the existing software clone detection techniques and types of clones.
ii. To study refactoring patterns, UML modeling language and diagrams.
iii. To propose a clone detection approach to detect clones at design and code level.
iv. To design and implement a tool that can automate the process of clone detection and removal at design and code level.

1.7 Organization of Thesis

Chapter 1 gives a brief introduction of Software Reuse, Software Cloning and concepts, Refactoring Patterns.

Chapter 2 deals with basic terminology of Software Cloning, its advantages, types of clones, detection techniques, and UML model based analysis, refactoring, refactoring patterns and maintenance.

Chapter 3 identifies the problem of Software Cloning.

Chapter 4 discusses the proposed 3-way approach for Clone Detection and Removal and its implementation through Pattern and Clone Detector (PC detector) tool.

Chapter 5 concludes the work done along with future directions.

Chapter 2

Literature Survey

This chapter describes in detail the literature survey. It covers basic terminology of Software Cloning, its advantages, types of clones, detection techniques, model based analysis, refactoring, refactoring patterns and maintenance.

2.1 Basic Concepts of Clone Detection

Clones, as the name implies, are copied regions of code. However, unlike a biological clone, a software clone may or may not be exactly the same [25]. For a given clone relation, a pair of code portions is called clone pair if the clone relation holds between the portions. An equivalence class of clone relation is called clone class [34]. The code clone pair and class are shown in figure 3.

2.1.1 Cloning Terminologies

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

Definition 2: *Code Clone.* A code fragment CF2 is a clone of another code fragment CF1 if they are similar by some given definition of similarity, that is, f(CF1) = f(CF2) where f is the similarity function (see clone types below). Two fragments that are similar to each other form a clone pair (CF1, CF2), and when many fragments are similar, they form a clone class or clone group [4].

Definition 3: *Clone Types.* There are two main kinds of similarity between code fragments. Fragments can be similar based on the similarity of their program text, or they can be similar based on their functionality (independent of their text). The first kind of clone is often the result

of copying a code fragment and pasting into another location. In the following we provide the types of clones based on both the textual (Types 1 to 3) and functional (Type 4) similarities [5]:

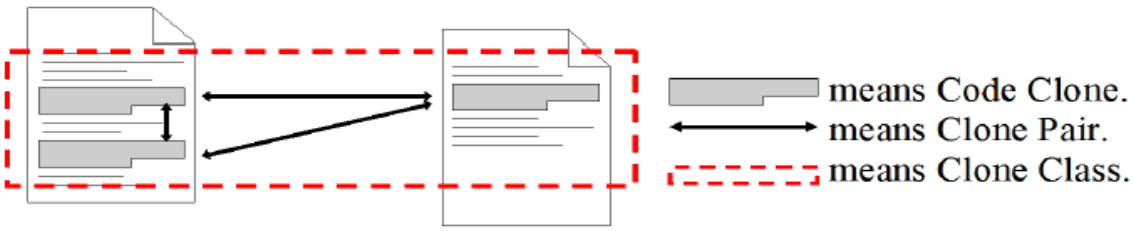


Figure 3: Clone relationship [4]

Type 1: Identical code fragments except for variations in whitespace, layout and comments.

Type 2: They are syntactically identical fragments, except for variations in identifiers, literals, types, whitespace, layout and comments.

Type 3: Copied fragments with further modifications such as changed, added or removed statements, in addition to variations in identifiers, literals, types, whitespace, layout and comments.

Type 4: Two or more, code fragments that perform the same computation but are implemented by different syntactic variants.

Definition 4: *Clone Pair:* A pair of code fragments is called a clone pair if there exists a clone relation between them, i.e., a clone pair is a pair of code fragments which are identical or similar to each other. For instance, the first code segment (a) in fragment 1 of Figure 4, together with segment (a) in fragment 2, forms a clone pair. Clones may subsume each other and we are typically interested only in the maximally long clones. For instance, segment (a) could be joined with the subsequent segment (b) in fragment 1 of Figure 4 to form a larger clone together with the joined segments (a) and (b) in fragment [5]. A maximally long clone pair is one whose two fragments can be extended neither to the left nor to the right to form a larger clone.

Definition 5: *Clone Class.* A clone class is the equivalence class formed by the (non-maximal) clone relation, that is, it is the maximal set of code fragments in which any two of the code fragments is a clone pair. For the three code fragments of Figure 4, we get the clone class <F1(b), F2(b), F3(a)> where the three code fragments F1(b), F2(b) and F3(a) each form clone

pairs with the others, that is, there are three clone pairs, <F1(b), F2(b)>, <F2(b), F3(a)> and <F1(b), F3(a)> [34].

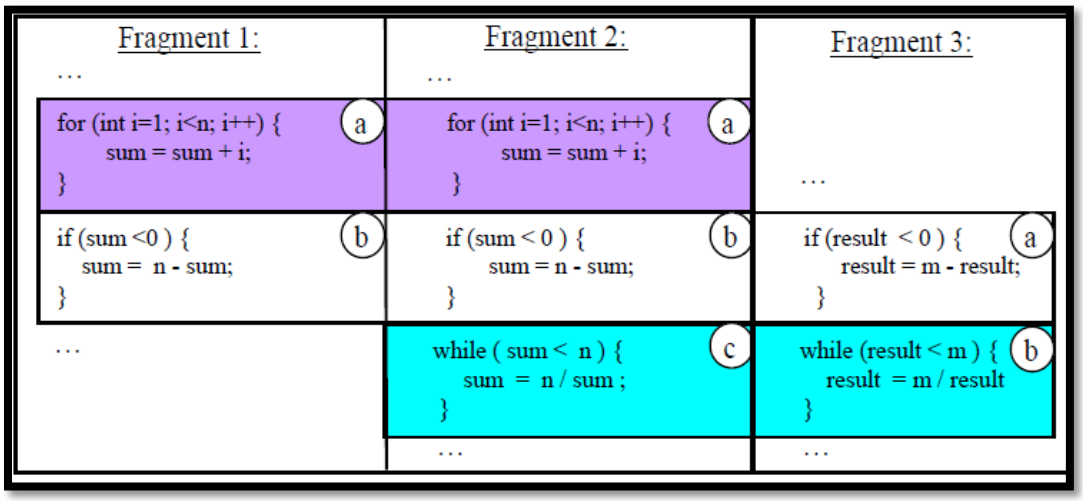


Figure 4: Code Example showing Clone pair and Clone Class [5]

Definition 6: *Clone Class Family.* The group of all clone classes that have the same domain is called a clone class family or super clone. The domain of a clone class is the set of source entities from which its source fragments stem [5]. The particular source entities to be considered as domains depend on the particular programming language or scope of interest, but common examples are files, functions, classes, or packages.

2.1.2 Reason of Software Cloning

Clones do not occur in the software themselves. There are various reasons that tend the developer to do cloning [5]. Figure 5 displays various factors for which clones can be introduced in the source code and a short description for some of the factors are discussed below:

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

ii. **Language Limitation**: Clones can be introduced due to the limitations of the language, especially when the language in question does not have sufficient abstraction

mechanisms. Sometimes, the developers are forced to copy because of limitations of their knowledge in that particular programming language.

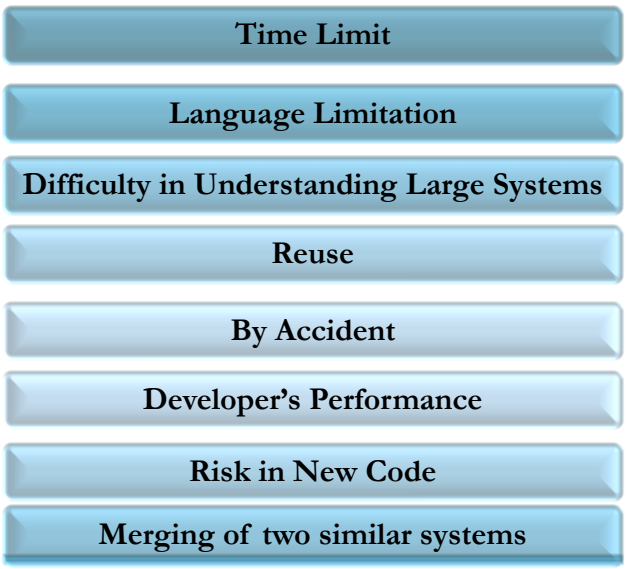


Figure 5: Reasons of Cloning

iii. **Difficulty in Understanding Large System**: It is generally difficult to understand a large software system. These force the developers to use the example-oriented programming by adapting previously developed existing code.

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

v. **By Accident**: Code cloning may be accidentally. There may be a case that two software developers may come with same solution. Programmers may unintentionally repeat a common solution for similar kind of problems using the common solution pattern of his memory to such similar problems. Therefore, several clones may unknowingly be created to the software systems.

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

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

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

2.1.3 Types of Clones

The degree of similarity varies from an exact superficial copy to more semantically similar regions of code (they do the same thing) or structurally similar regions of code (similar patterns of statements) [25]. Clones are categorized into four types:

Type I: Exact Software Clones (Changes in layout and formatting)

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

Type II: Near-Miss Software Clone (Renaming Identifiers and Literal Values)

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

Type III: Near-Miss Software Clone (Statements added/deleted/modified in copied fragments)

Type III is copy with further modifications, e.g. a new statement can be added, or some statements can be removed. The structure of code fragment may be changed and they may even look or behave slight differently.

Type IV: Near-Miss Software Clone (Statements reordering/control replacements)

Type IV clones are the results of semantic similarity between two or more code fragments. In this type of clones, the cloned fragment is not necessarily copied from the original. The same kind of logic may be implemented making the code fragments similar in their functionality [5]. Example study of Clone types is shown in Figure 6.

Example code fragment

```
void sumProd(int n) {     //s0
int sum=0;                //s1
int product =1;           //s2
for (int i=1; i<=n; i++) { //s3
    sum=sum + i;          //s4
    product = product * i; //s5
    fun(sum, product); }} //s6
```

Type 1:

```
void sumProd(int n) {     //s0
  int sum=0;              //s1
  int product =1;         //s2
  for (int i=1; i<=n; i++) { //s3
      sum=sum + i;        //s4
      product = product * i; //s5
      fun(sum, product); }} //s6
```

Changes in whitespace

```
void sumProd(int n) {     //s0
int sum=0;                //s1
int product =1;           //s2
for (int i=1; i<=n; i++)  //s3
    sum=sum + i;          //s4
    product = product * i; //s5
    fun(sum, product); }} //s6
```

Changes in comments

```
void sumProd(int n) {     //s0
  int sum=0;              //s1
  int product =1;         //s2
  for (int i=1; i<=n; i++) { //s3
      sum=sum + i;        //s4
      product = product * i; //s5
      fun(sum, product); }} //s6
```

Changes in formatting

Type 2:

```
void addTimes(int n) { //s0
int add=0;             //s1
int times =1;          //s2
for (int i=1; i<=n; i++) { //s3
    add=add + i;       //s4
    times = times * i; //s5
    fun(add, times); }} //s6
```

Renaming of identifiers

```
void sumProd(int n) {     //s0
double sum=0.0;           //s1
double product =1.0;      //s2
for (int i=1; i<=n; i++) { //s3
    sum=sum + i;          //s4
    product = product * i; //s5
    fun(sum, product); }} //s6
```

Renaming of Literals and Types

Type 3:

```
void sumProd(int n) {       //s0
int sum=0;                  //s1
int product =1;             //s2
for (int i=1; i<=n; i++) {  //s3
    if (i % 2 == 0) sum+= i; //s4m
    product = product * i;  //s5
    fun(sum, product); }}   //s6
```

Modification of lines

```
void sumProd(int n) {       //s0
int sum=0;                  //s1
int product =1;             //s2
for (int i=1; i<=n; i++)    //s3
    if (i % 2 == 0){        //s3b
    sum=sum + i;            //s4
    product = product * i;  //s5
    fun(sum, product); }}   //s6
```

Addition of new of lines

```
void sumProd(int n) {       //s0
int sum=0;                  //s1
int product =1;             //s2
for (int i=1; i<=n; i++) {  //s3
    sum=sum + i;            //s4
    //s5 line deleted
    fun(sum, product); }} //s6
```

Deletion of lines

Type 4:

```
void sumProd(int n) {     //s0
int sum=0;                //s1
int product =1;           //s2
int i = 0;                //s7
while (i<=n) {            //s3'
    sum=sum + i;          //s4
    product = product * i; //s5
    fun(sum, product);    //s6
    i =i + 1; }}          //s8
```

Reordering of Statements

```
void sumProd(int n) {     //s0
int product =1;           //s2
int sum=0;                //s1
for (int i=1; i<=n; i++) { //s3
    sum=sum + i;          //s4
    product = product * i; //s5
    fun(sum, product); }} //s6
```

Control Replacements

Figure 6: Types of Clone

2.1.4 Advantages of Clone Detection

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

i. **Detect Library Candidates**

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

ii. **Bug Detection**

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

iii. **Program Understanding**

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

iv. **Code Compaction**

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

v. **Find Usage Patterns**

If all the cloned fragments of a same source fragment can be detected, the functional usage patterns of that fragment can be discovered. This is the reason why the software clone detection gains considerable attention.

2.1.5 Software Clone Analysis

In reality code duplication leading to cloning is a common habit. Many authors have reported 7-23% code duplication; and in some cases even 59% [56]. Duplication and redundancy can increase the size of the code, make it hard to understand the many code variants, and cause maintenance difficult. The goal of avoiding clones has provided the momentum to investigations on software reuse, software refactoring, modularization, and parameterization.

Clone analysis studies report a number of values indicating how much code was cloned. In their extensive review of clone analysis research [5] report that between 5% and 20% of code in large-scale software systems is duplicated or copied [25]. Baker reported up to 38% of code in the systems she studied was cloned [2]; Its found 59.3% of the system they studied was copied code [32]. Clone analysis has typically been used to measure the quality of source code, and indicate areas for refactoring [1] and improvement. The general argument against cloning (i.e., the duplication of similar or identical code) is that similar regions of code represent unnecessary duplication within the program, which increases the cost and complexity of maintaining the code as programmers must track more code, make repeated edits to multiple clones, and can easily propagate bugs [25].

2.1.6 Overview of Clone Detection Techniques

Code cloning detection had been an active research for almost two decades. The detection of code clones is a two phase process which consists of a transformation and a comparison phase. In the first phase, the source text is transformed into an internal format which allows the use of a more efficient comparison algorithm. During the succeeding comparison phase the actual matches are detected. Due to its central role, it is reasonable to classify detection techniques according to their internal format.

The techniques cover the work on text, lexical and syntactic information, software metrics, and program dependency graphs [4].

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

vi. **Hybrid**- detection techniques that use a combination of the other clone detection techniques.

2.2 Design Model (UML) Based Clone Detection

As a consequence of the highly reuse-oriented approach, the identification of common elements in different parts of the software provides an important asset for the model-based development process [42].

2.2.1 Design Model Based Development

A model plays the analogous role in software development that blueprints and other plans play in the building of a skyscraper. Models help to understand the system by simplifying some of the details. The choice of what to model has an enormous effect on the understanding of the problem and the shape of the solution. UML is a modeling language that is used for specifying, visualizing, constructing, and documenting the artifacts of software systems. UML is a pictorial language used to make software blue prints. UML is described as a general purpose visual modeling language used at design level. It was initially started to capture the behavior of complex software and non software system and now it has become an OMG standard [46].

Model-based development is becoming an increasingly common development methodology. Modeling a software system has become an integral part of the software development and maintenance process and therefore has a major economic and strategic value for the software-developing organizations. Nevertheless little work has been done on a quality defect that is known to seriously hamper maintenance productivity in classic code based development: Cloning. However, there has been very little work on cloning in the context of modeling at design level. An approach for the detection of clones is to detect them in UML diagrams during modeling at design level.

It is well known that most code clones are created by ad-hoc-reuse through copy/past. Code fragments are copied but no explicit link is established between them. Since this practice is common in modeling, too, it is plausible to assume that the same holds for models. In fact, something worse happens: the two parallel structures discussed in Figure 7 are affected in different ways by editing operations like copy and paste.

It is common practice among industrial modelers to interact with the model primarily or only through the diagrams, i. e. the model views during designing phase [43]. This leads to a hazardous feature interaction between common editing commands: copying a diagram element may either create only a new (visual) reference to a pre-existing model element, or also a new copy of the model element. Deleting a diagram element, on the other hand, is usually implemented as deleting only the visual reference, but leaving the corresponding model element untouched. So, a modeler may inadvertently create clones when they just wants another visual reference, or when they delete the visual reference only, instead of both the diagram element and the model element. Figure 7 sketches a typical situation.

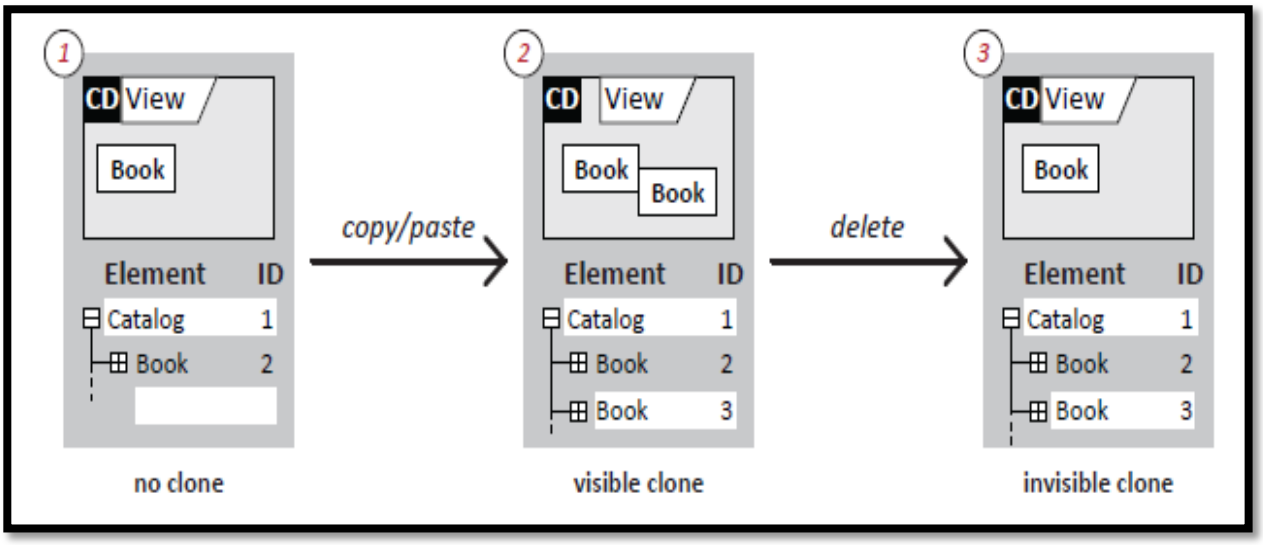


Figure 7: Simple copy/ paste or editing operations can create clones in models [44]

Probably the biggest problem with source code clones is defining exactly what is and what not a clone [45] is. For model clones, this is at least not as difficult, as it can be visualized easily. A model clone is a set of similar or identical fragments in a model. Unfinished modeling contains lot of clones that are needed to be removed at this level only. As shown in Figure 8 the common data is pulled up so that no commonalities may occur.

In source code, elements like types, procedures, and so on are identified by their names. In general, code fragments are identified exactly by their textual representation. Copying a text fragments obviously retains this identity. That is to say, source code clones are identical (to begin with). In many modeling environments, on the other hand, model elements have internal

identifiers. Since these internal identifiers are usually understood as globally unique, a deep copy of a model element will consistently change the identifiers in the duplicate. Thus, model clones are equal, but not identical.

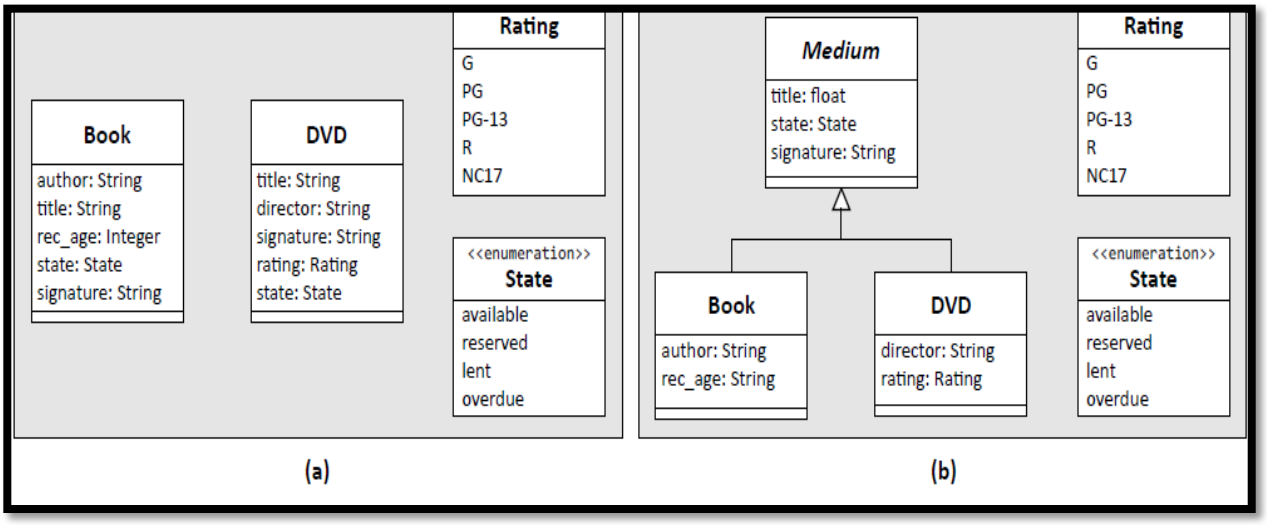


Figure 8: (a) Unfinished Modeling (b) Finished Clone Free Modeling [43]

2.2.2 Definition of Model Clones

UML models can be regarded as sets of instances of Meta model elements which may have links to other such elements. So, a first attempt to define model clones might simply be: any model element e that bears a high degree of similarity to some other model element e'. On the other hand, model elements like Class or Activity should not be considered in isolation, as individual model elements. Rather, they are the roots of trees of elements. For instance, a Class owns a set of Properties and Operations, which may own Types and Parameters in turn. So, as a second step, we might define model clones as pairs of model fragments, where a model fragment is a (complete) tree of model elements. This leads to the following definition.

Definition (*Model Fragment, Model Clone*): A model fragment is a set of model elements that is closed under the containment-relationship. A model clone is a pair of model fragments such that there is a high degree of similarity between the fragments [44].

2.2.3 Types of Model Clones

There are several different ways of defining code clones. The following list enumerates the most commonly cited clone type classification [47] [48].

Type I: *Exact clone*: A duplicate that is identical except at most changes to whitespaces and comments.

Type II: *Renamed clone:* A duplicate with consistent changes to identifiers of variables, types, or functions

Type III: *Parameterized clone*: A duplicate allowing arbitrary changes, additions or removals of parts. This type of clone is also called near-miss clone.

Type IV: *Semantic clone:* A duplicate in content only that may be due to code copying, convergent development, or other processes.

2.3 Clone Detection with Refactoring Patterns

Refactoring code clones is an effective way to reduce code clones in a software system. Refactoring patterns are used to find the clone codes. It is a typical activity to remove code clones.

"*Every pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.*" – Christopher Alexander.

2.3.1 Refactoring

Refactoring is an effective technique to conduct the perfective maintenance. It is defined as "the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure[50]." It's important to note that refactoring don't change or improve software from a functional point of view [49]. The program is intended to do the same thing after refactoring that it did before refactoring.

Today, many development methods have a different approach to software design. Especially agile development methods – most prominently extreme programming – no longer treat software

design as a clearly and rigidly defined constant that is defined at the beginning of a development project. Design improvements become established as an important and independent activity during development and evolve into an integral part of this process. This activity is called refactoring. First of all, refactoring means, changing the internal structure of software to make it easier to read and modify without altering its observable behavior. Besides acknowledging this rather technical definition, many developers also associate a process-related aspect and a certain attitude with the refactoring term. In the context of extreme programming, refactoring means first and foremost an ongoing and repeated reflection about the software's structure and improving it in small increments.

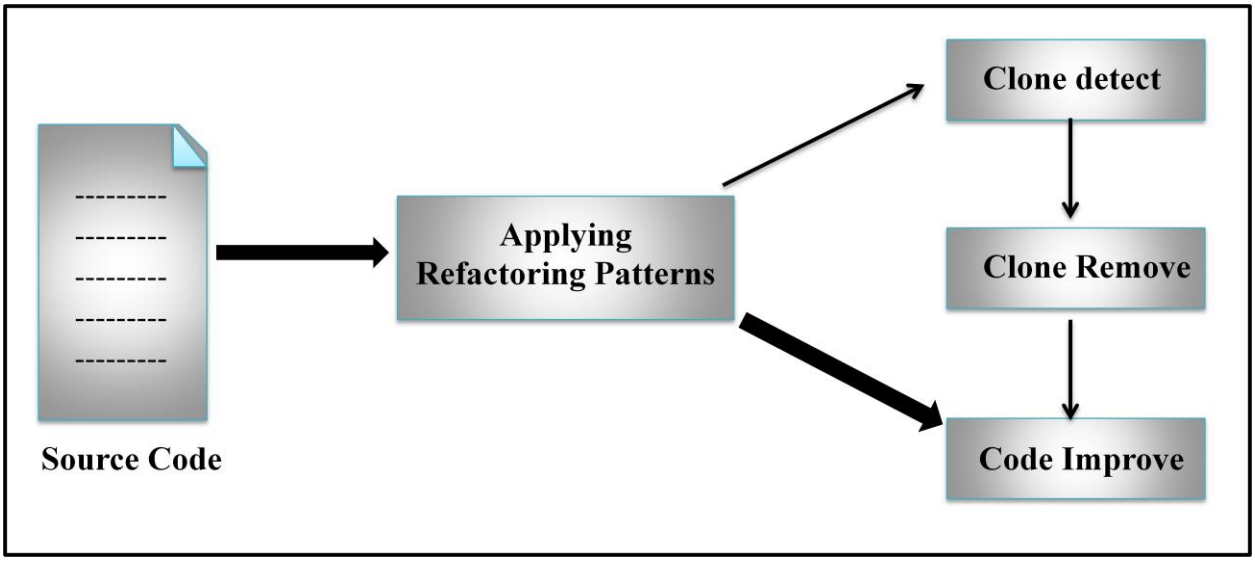


Figure 9: Identifying Clones through Refactoring Patterns

2.3.2 Refactoring Patterns

Various refactoring patterns are used to remove code clones like [26] "Extract Method" and "Pull Up Method" that are related to the code clone. However, quite often one must use a series of refactoring to actually remove duplicated code, as in Transform Conditionals into Polymorphism where duplicated conditional logic is refactored over the class hierarchy using polymorphism [8]. With refactoring tools like the refactoring browser [7] emerging from research laboratories into mainstream programming environments, refactoring is becoming a mature and widespread technique.

Refactoring pattern as shown in Figure 9 improves the code by detecting clones and then removing them. Here four cloning patterns are discussed namely extract, pull up, template and strategy. If the same code structure is in more than one place, it's sure that code will become better if it is unified. The simplest duplicated code problem is having the same expression in two methods of the same class. Then Extract pattern is used and the code is invoked from both places.

Another common duplication problem is having the same expression in two sibling subclasses. This duplicity is removed by using Extract pattern in both classes then Pull Up Field. If the code is similar but not the same then Extract pattern is used to separate the similar bits from the different bits. If the methods do the same thing with a different algorithm Strategy pattern can be used [51].

2.3.2.1 Extract Pattern

Refactoring with *Extract Method* is useful when a method is too long to understand, or when common blocks of code appear in multiple places. Code is extracted to create a separate method, and the original method is changed to invoke the new method [51]. Extract Method means extraction of a part of existing method as a new method, and extracted part is replaced by a new method caller shown in figure 10.

First, it increases the chances that other methods can use a method when the method is finely grained. Second, it allows the higher-level methods to read more like a series of comments. Overriding also is easier when the methods are finely grained.

In general, this pattern is applied to the case that there is a too long method. In applying the pattern to code clones, a new method, that is a code fragment of code clone, is defined and the original code clones are replaced by the new method caller. As the result, we can remove the code clones [36].

2.3.2.2 Pull Up Pattern

Pull Up Method is a simple refactoring pattern. It means pulling up a method which defined in child class to its parent class. If the parent class has several child classes and some of them have the same method (that is, code clone), pulling up the method can remove the code clone [36].

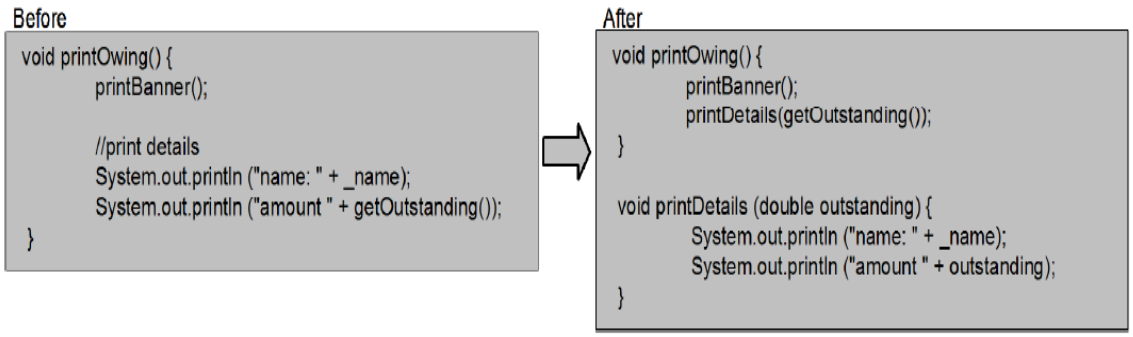


Figure 10: Example of Extract Method [36]

This refactoring is especially useful when subclasses of a certain class share some functionality but each implements it separately. Moving the shared functionality to the super class will keep the functionality of the subclasses as it is, and give us less code duplication and better code readability [52]. The easiest case of using *Pull Up Method* occurs when the methods have the same body, implying there's been a copy and paste.

Eliminating duplicate behavior is important. Although two duplicate methods work fine as they are, they are nothing more than a breeding ground for bugs in the future. Whenever there is duplication there is risk that an alteration to one will not be made to the other. Usually it is difficult to find the duplicates. Figure 11 shows the example of pull up method.

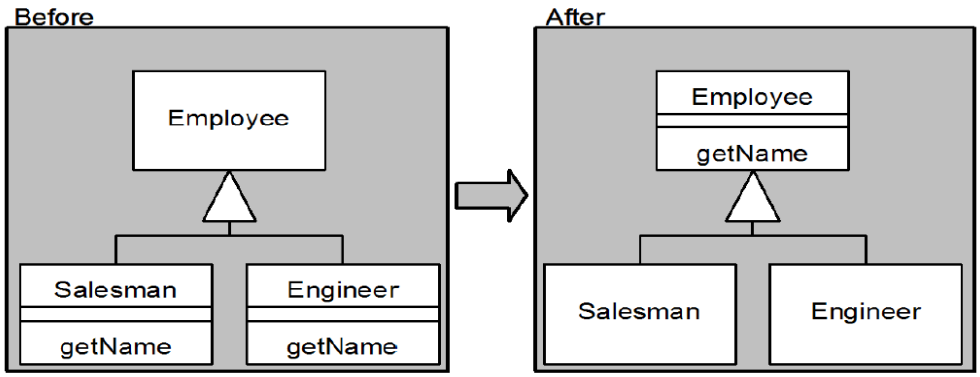


Figure 11: Example of Pull Up Method [36]

2.3.2.3 Template Pattern

The dictionary definition of a template is a preset format, used as a starting point for a particular application so that the format does not have to be recreated each time it is used. On the same idea

is the template method is based [52]. A template method defines an algorithm in a base class using abstract operations that subclasses override to provide concrete behavior. Example of Template pattern is shown in figure 12.

The template method occurs frequently, at least in its simplest case, where a method calls only one abstract method, with object oriented languages. If a software writer uses a polymorphic method at all, this design pattern may be a rather natural consequence. This is because a method calling an abstract or polymorphic function is simply the reason for being of the abstract or polymorphic method. The template method may be used to add immediate present value to the software or with a vision to enhancements in the future [51].

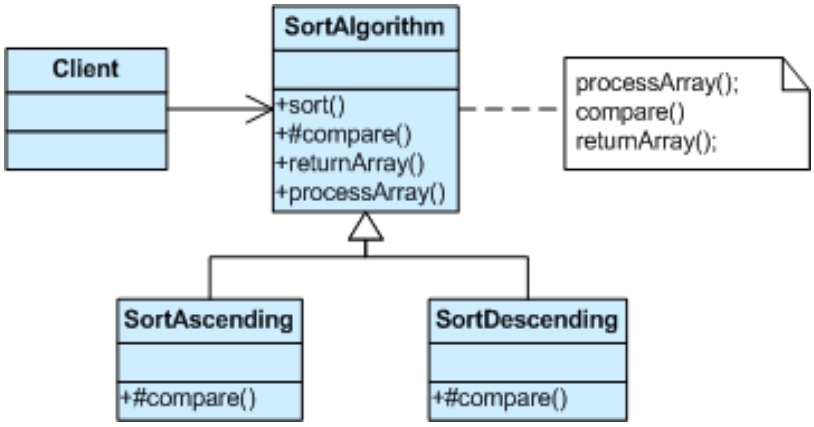


Figure 12: Example of Template Pattern [51]

The Template Method pattern should be used: to implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary and when refactoring is performed and common behavior is identified among classes. A abstract base class containing all the common code (in the template method) should be created to avoid code duplication.

2.3.2.4 Strategy Pattern

There are common situations when classes differ only in their behavior. For this case is a good idea to isolate the algorithms in separate classes in order to have the ability to select different algorithms at runtime.

A strategy is an algorithm. Strategies often have internal variables that record the state of the algorithm. At the end of the algorithm, the variables might record the result, but in general they

are only meaningful during the execution of the algorithm. A strategy is either selected by an outside agent or by the context. A strategy tends to have a single "start" method, and it calls all the rest. There is a lot of cohesion between the methods of a strategy [52]. Figure 13 shows an example of Strategy Pattern.

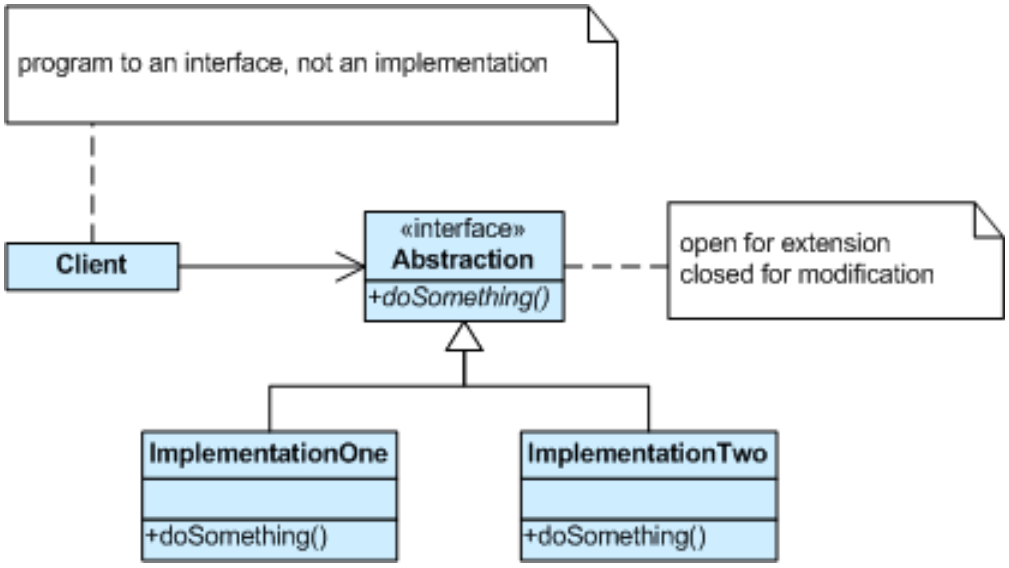


Figure 13: Example of Strategy Pattern [52]

2.4 Cloning and Software Maintenance

The impact of clones is of special concern from a software maintenance point of view. The software life cycle comprises two major parts; first we define the specification and implement it; then, we need to maintain the finished product and evolve it to better suit user needs. However for software development it has been found that maintenance and evolution are also critical activities from the cost perspective and might comprise up to 80% of the overall cost and effort [15] [17].

Fowler suggests that code duplication or cloning is a bad smell and thus one of the major indicators of poor maintainability [15]. Cloning is an easy, tempting alternative to the hard work of actually refactoring the code. Maintaining the cloning relationship is thus a very important consequence of copying and pasting. Without tracking clones over time, identifying and consistently changing clones can be problematic [28] [29] and increases the maintenance cost.An intrinsic property of software in a real-world environment is its need to evolve. As the software

is enhanced, modified and adapted to new requirements, the code becomes more and more complex and drifts away from its original design [41]. Because of this, the major part of the total software development cost is devoted to software maintenance [53]. Better software development methods and tools do not solve this problem, because their increased capacity is used to implement more new requirements within the same time frame, making the software more complex again. To cope with this spiral of complexity there is an urgent need for techniques that reduce software complexity by incrementally improving the internal software structure. The research domain that addresses this problem is referred to as restructuring [54] or, in the case of object-oriented software development, refactoring [55].

As such, the maintenance of clones can be considered to fall under the spectrum of software maintenance, a phase in software development that can encompass up to 90% of the total effort in software development [56]. However, clone maintenance can occur during any software lifecycle phase that involves coding as programmers continue to fix or refine the code that may contain clones. Cloning can reduce increasing the cost of maintenance and also improves the quality of software.

Chapter 3

Problem Statement

Software Cloning effects software maintenance and other engineering efforts. Cloning at design as well as at code level is seemed as an obstacle so it is needed to be removed. And hence cloning has grown as an active area in software engineering research community yielding numerous techniques, various tools and other methods for clone detection and removal. Cloning in source code has been reported for different programming languages and application domains.

There are various techniques for detecting clones at code level. But no such technique or methodology is available that could detect the clones at design level. Design is one of the starting phases of software development. It would be more effective and useful if clones are detected at this level. It will enhance the maintenance with comparatively less efforts. These detected clones can be easily visualized and hence analyzed in UML diagrams and thus cloning can be removed.

The refactoring patterns are used to improve the code. The same if integrated with cloning can easily detect the clones and then can remove them. No such integration methodology is there for cloning. Hence integrating patterns in cloning will not only enhance the code but will also detect the clones which are then removed from code.

Problem Identification:

i. We need a better approach for the analysis of clone detection, its removal and also elimination of flaws present.
ii. An approach that could integrate cloning at design as well as code level for the effective maintenance is needed.
iii. As it is based on Code Detection, an automated system or tool is needed, that can automatically detect and remove clones. This will definitely contribute to reduce the maintenance efforts.

This thesis presents a 3-way approach for detecting and removing clones. This work contributes a methodology by integrating and merging Model-Based Visual Analysis, Pattern Based

Semantic Analysis and Syntactical Analysis of Code for Clone detection and thereby improves the code, design, maintenance, quality and software as a whole.

In this dissertation, a methodology is proposed that integrates work in three stages. Firstly, clones are detected at design level through Model-Based Analysis of UML diagrams and are then removed. Secondly, Pattern Based Analysis is done. And thirdly, Syntactical clones are detected. This clone detection and removal technique is a 3- Way approach that enhances and improves the coding as well as designing of the software. Also a tool named PC Detector is developed to automate pattern and clone detection process and thereby its removal.

Chapter 4

4.1 Proposed Model

Embracing software cloning is the cornerstone of software development and maintenance. Being able to detect clones more efficiently will improve its quality, increase its reuse and will also reduce the overall cost of maintenance. Manually tracing the clones is difficult. In proposed work an automated tool for detecting patterns and clones is developed.

In this thesis, a 3-way approach is proposed based on Model Based Visual Analysis; Pattern Based Semantic Analysis and Syntactical Code Analysis for detection and removal of clones. This 3-way approach detects clones that cannot be detected by other methods at both design and code level. The tool is based on (a) simple string matching and parameterized string matching, (b) refactoring pattern detection and removing the clones by synthesizing the duplication information found, and (c) matrix representation visualization as a helpful means in analyzing duplication.

The tool also makes a token sequence from the input code after normalization and filtration. The purpose is to transform code portions in a regular form to detect Type 2 clone code portion. Another purpose is to filter out code portions with specified keywords. Representing a source code in matrix format enables us to detect clones within same and different files also. Using this tool it is easy to identify (a) duplicated code between several files, (b) within the same file. The clones detected provide useful information for practical software maintenance and reengineering tasks.

The proposed model as shown in figure 14 is divided into three phases: Model Based Visual Analysis, Pattern-Based Semantic Analysis and Syntactical Code Analysis. At design level, the clones are detected and removed through UML model-based visual approach. At code level clone detecting patterns and their solutions are applied that remove the clones. Even after Pattern Based Semantic Analysis some syntactically similar code clones are remained that are further detected. Pattern- Clone Detector (PC Detector) tool is developed to automate the task of pattern and syntactical clone detection.

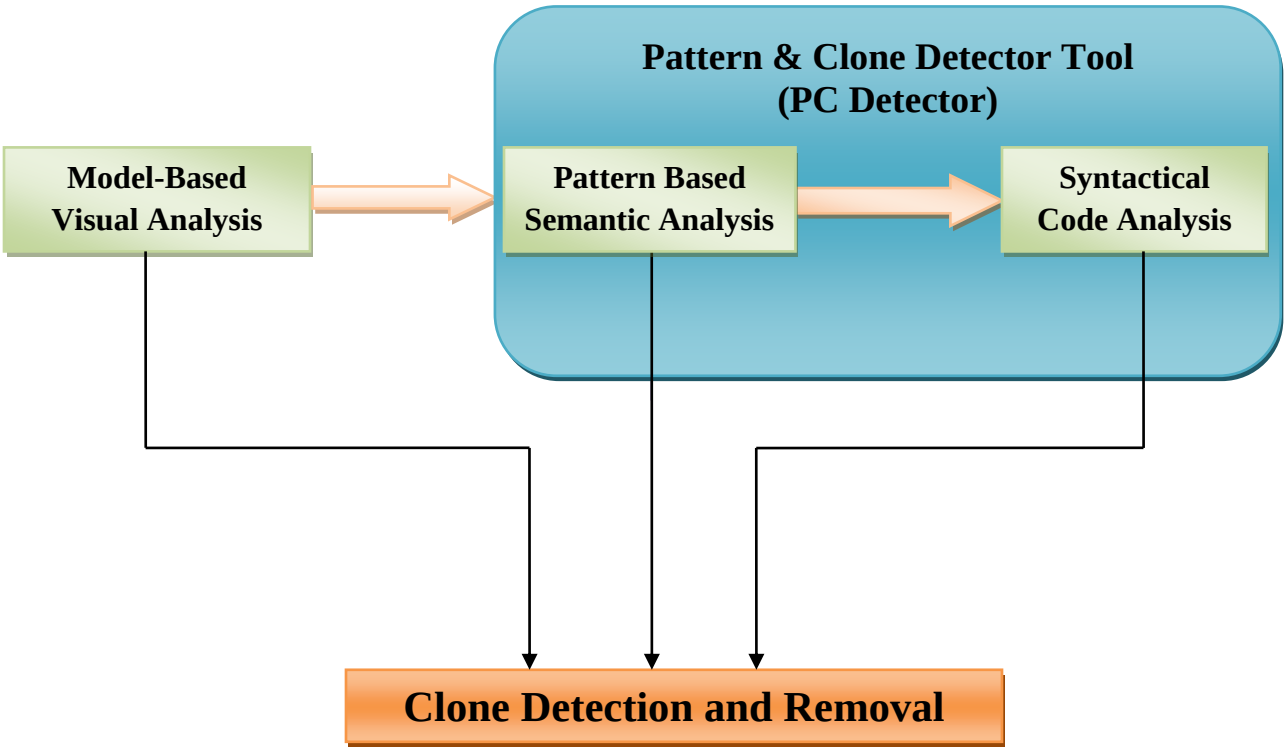


Figure 14: A 3-Way Approach for Clone Detection and Removal with PC Detector

4.2 Model Based Visual Analysis through UML

During design phase of software development process modeling of system is done using UML diagrams. With the help of Use case, Class diagrams and Activity diagrams software structure and its behavior can easily be understood and hence it helps in detecting the clones.

We will use the case study of Hospital Management System (HMS) as our running example. There are occurrences of common methods and variable among classes. It can't be said just looking at the view that they are clones or not.

Model clones through unfinished modeling are needed to be detected and removed. Some clones may be characterized as the remains of unfinished modeling. Use case diagram are used to view overall behavior of system, class diagram directly helps in implementing the code, activity diagrams are used to show various actions step-by-step. Thus removing clones at design level only will further help at coding level.

4.2.1 Case Study: Hospital Management System (HMS)

Here a case study of Hospital Management System (HMS) is taken to show UML diagrams to detect the clones at design level and then removing them. The **Hospital Management System** is an integrated end-to-end System that provides relevant information across the hospital to support effective decision making for patient care, hospital administration and critical financial accounting, in a seamless flow.

The HOSPITAL MANAGEMENT SYSTEM (HMS) covers many hospital activities. It also allows hospitals to choose from the various modules to match their specific needs. Every hospital big or small keeps the records of its patients including the registration details of the patient and the fee payments. Patients are categorized as In Patients and Out Patients. The entry of patients is determined whether he has arrived in emergency, OPD or for a routine check-up. The patient who gets admitted is provided with a room according to his/her choice. The patient is allotted a doctor according to his illness. The doctor may refer the patient to another doctor with expertise of the illness. On discharge, the patient is required to settle the bills sent by the accounts department of the hospital.

Out Patient segment keeps the record of outdoor patients. It has various features like adding new patient details, searching, updating or deleting patient records, preparing medical bill, keep track of patient records and also the bill reports. In Patient segment keeps the record of admitted patients. It has various features like adding new patient details, searching, updating or deleting patient records, preparing medical bill, keep track of patient records and also the bill reports. It also keeps record of discharged patients.

Here the Hospital Management System (HMS) case study is used as a running example, a small analysis-level model expressed in following UML diagrams in order to show the clones.

- Use-case Diagram
- Class Diagram
- Activity Diagram

4.2.2 Use-Case Diagram

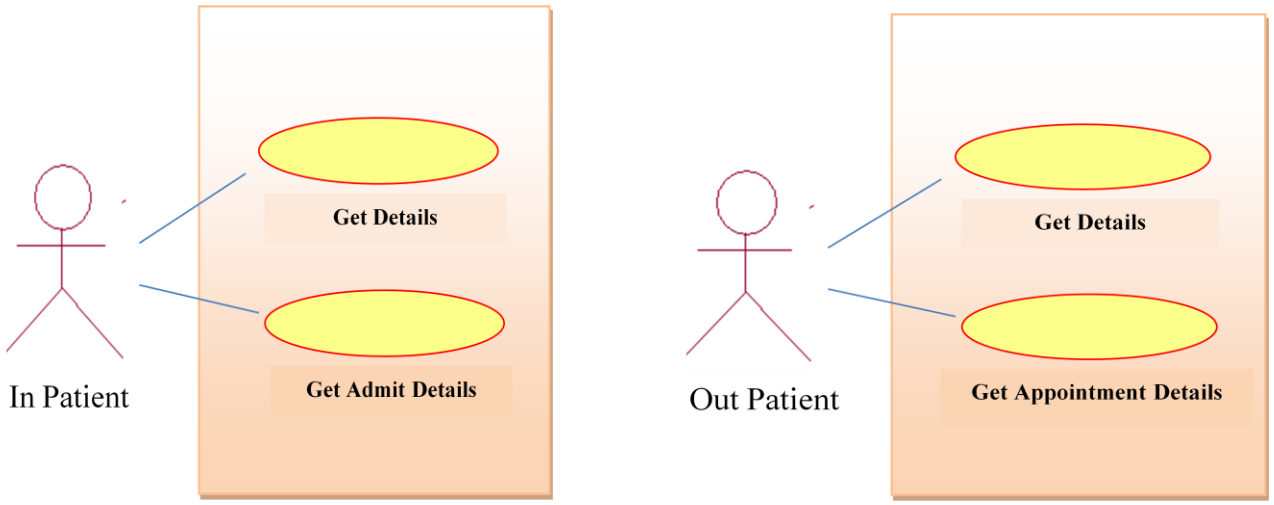


Figure 15 (a): Use Case Diagram showing Unfinished Model

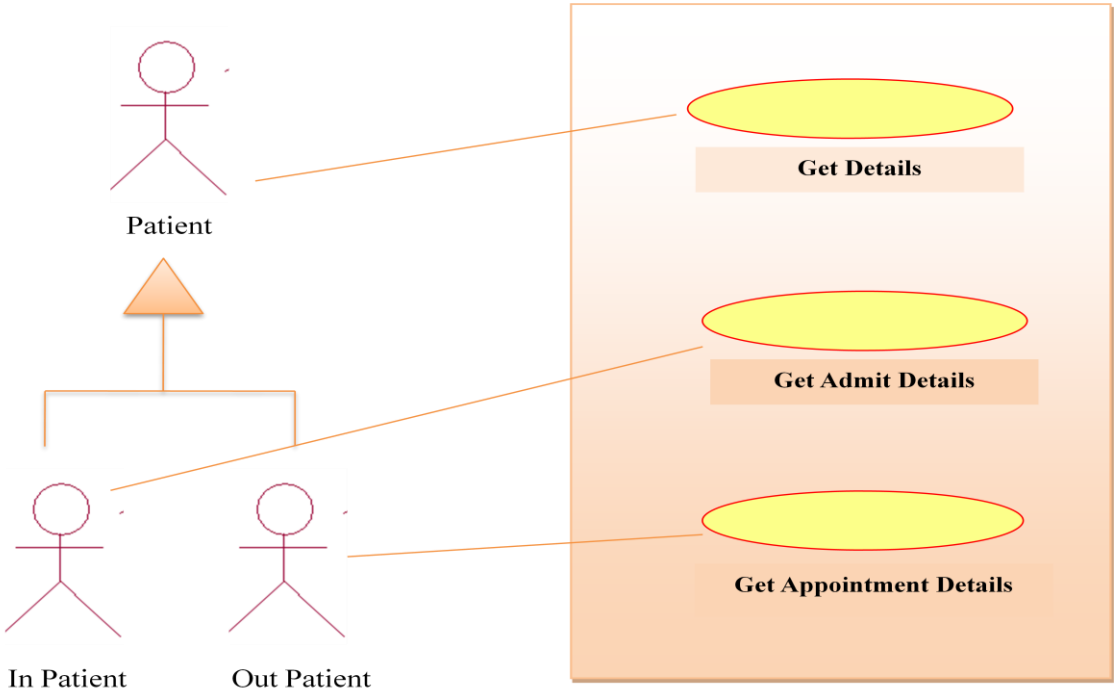


Figure 15 (b): Use Case Diagram showing Finished Model

The HMS case study is used to show how the clones can easily be detected and how the problem of their removal is solved. Figure 15 (a) shows the use case diagram of the category of patients- InPatients and OutPatients. It can be seen that inpatients and outpatients have same behavior and common attributes. Visualizing these we can see that they are actually clones to some extent. So

we can easily remove these clones as shown in the figure 15 (b). As getDetails() is a common method of InPatients and OutPatients. This can be pulled up thus removing the common data, i.e. clones.

4.2.3 Class Diagram

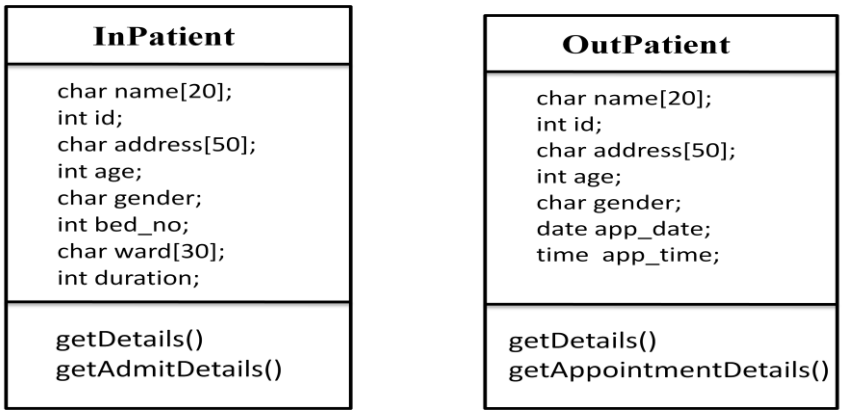


Figure 16 (a): Class Diagram showing Unfinished Model

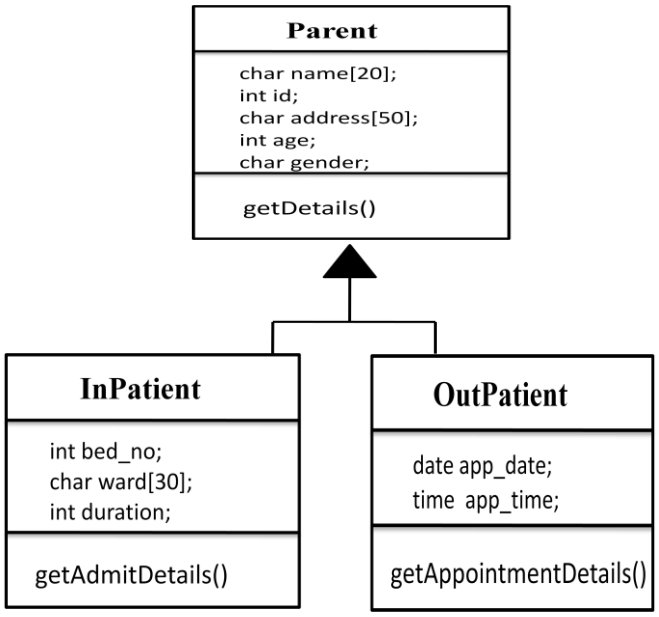


Figure 16 (b): Class Diagram showing Finished Model

Figure 16(a) shows a schematic representation of a part of the HMS highlighting the class structures. It is seen in the figure 16(a) that there are occurrences of common methods and variable of both the two classes named InPatient and OutPatient in the diagram. The method getDetails and various attributes like name, id, age, contact,and others that are identical in both

classes InPatient and OutPatient. They could be factored out into a class Patient which is a common superclass to both InPatient and OutPatient , yielding the model shown in Figure 16(b).

4.2.4 Activity Diagram

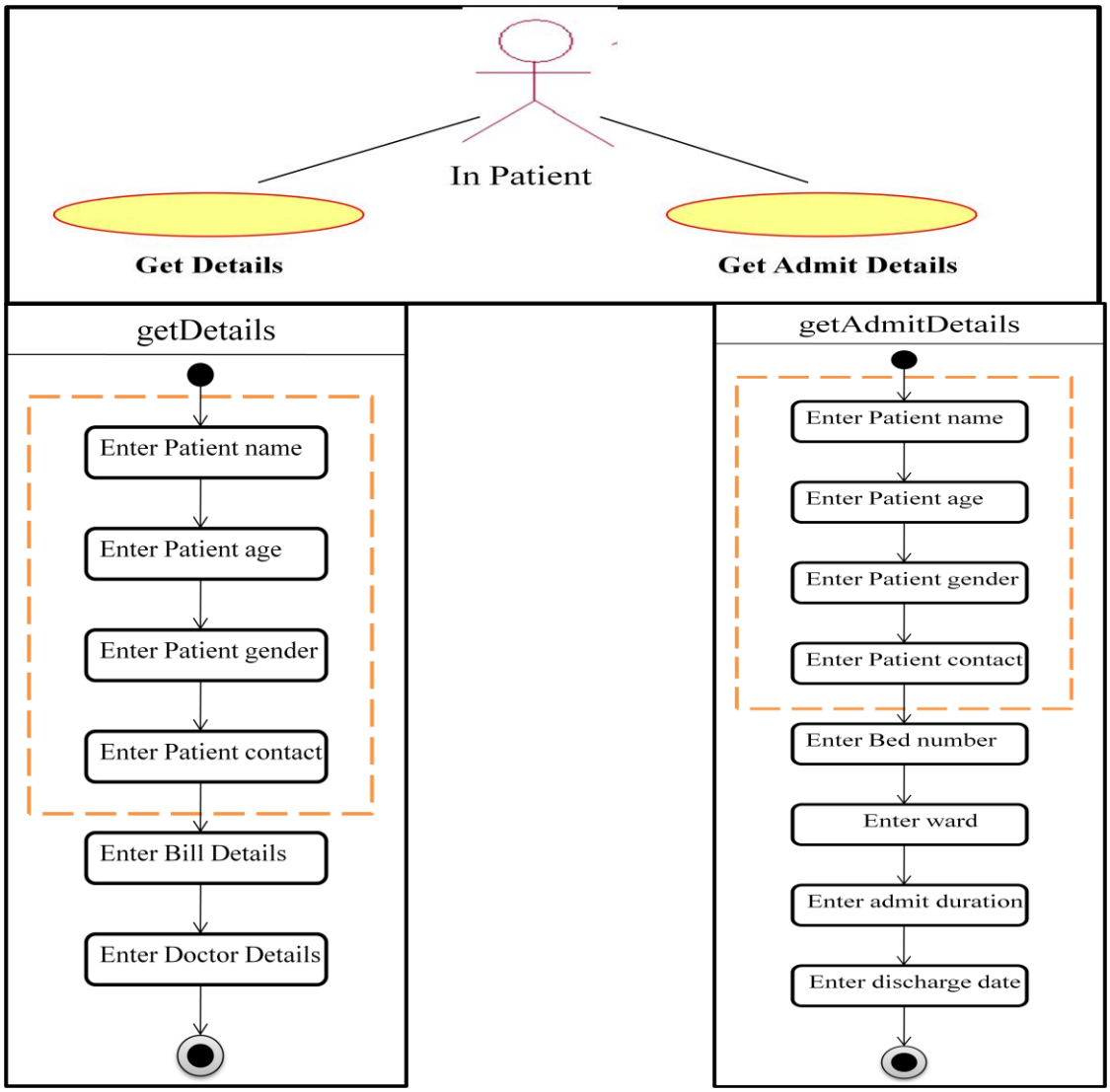


Figure 17(a): Activity Diagram showing Unfinished Model

Activity diagram shows series of actions performed. This will not only remove the outer clones but also the clones among the same class or of same actor. As shown in figure 17 (a) , the input of personal details of patients is common in both the methods of getDetails() and getAdmitDetails(). So the detection of clones is further analyzed and thus they are removed. So

the common entry of personal details is extracted and made common among both methods thus removing the clones as shown in figure 17 (b).

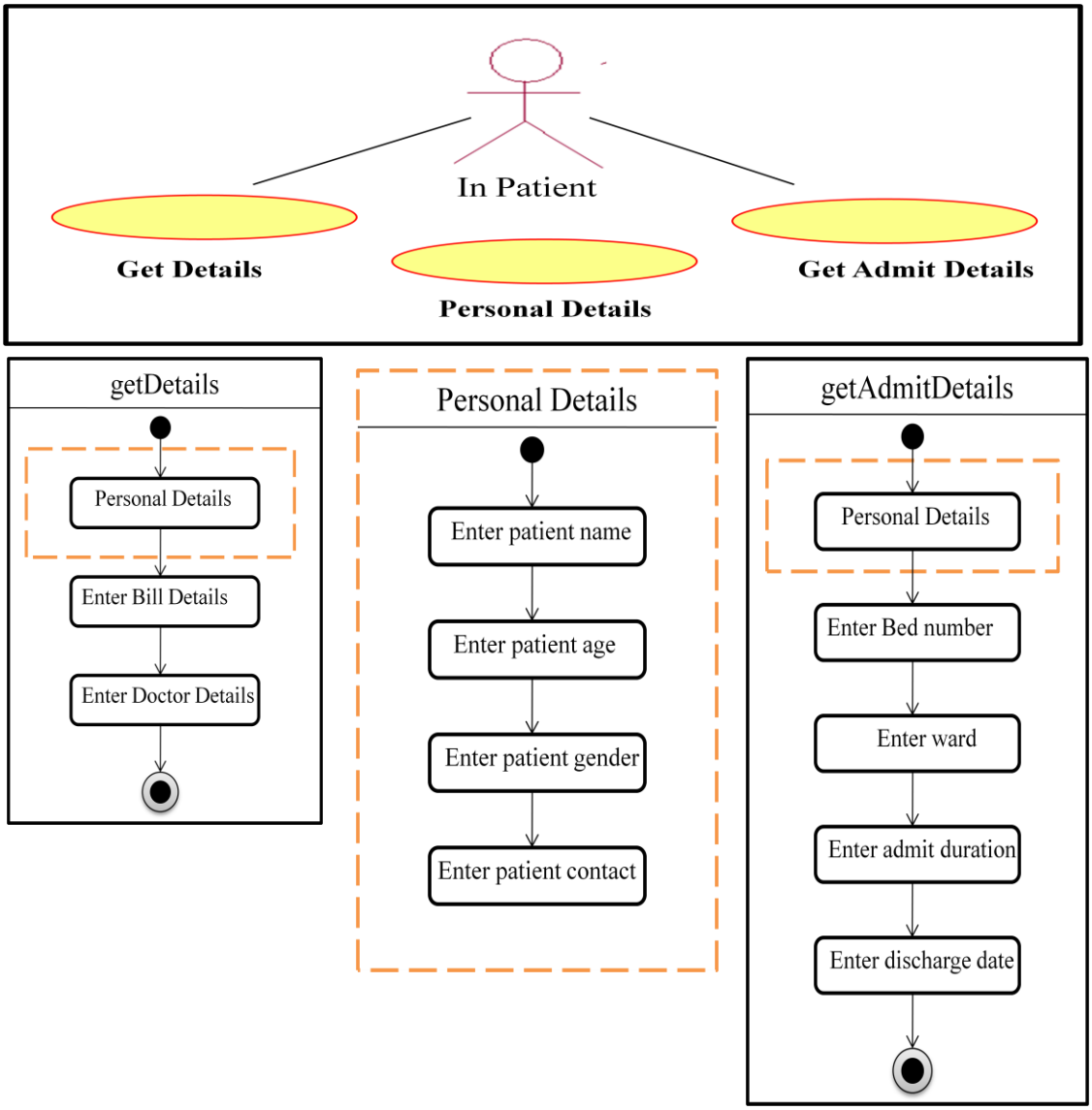


Figure 17 (b): Activity Diagram showing Finished Model

4.3 Pattern Based Semantic Analysis

Pattern based semantic analysis detects cloning patterns and remove them. There are various refactoring patterns where clones can be detected. Automating this task of recognizing the patterns and applying appropriate solution will detect the clones and also remove them with greater accuracy. Here Pull Up pattern is automated using P C Detector tool. The tool will detect common data members and member functions among the classes. It outputs a new parent class and makes other classes its child classes. The process is explained with the help of flowchart shown in figure 18 followed by its algorithm.

4.3.1 Flowchart of Pattern Detection Process

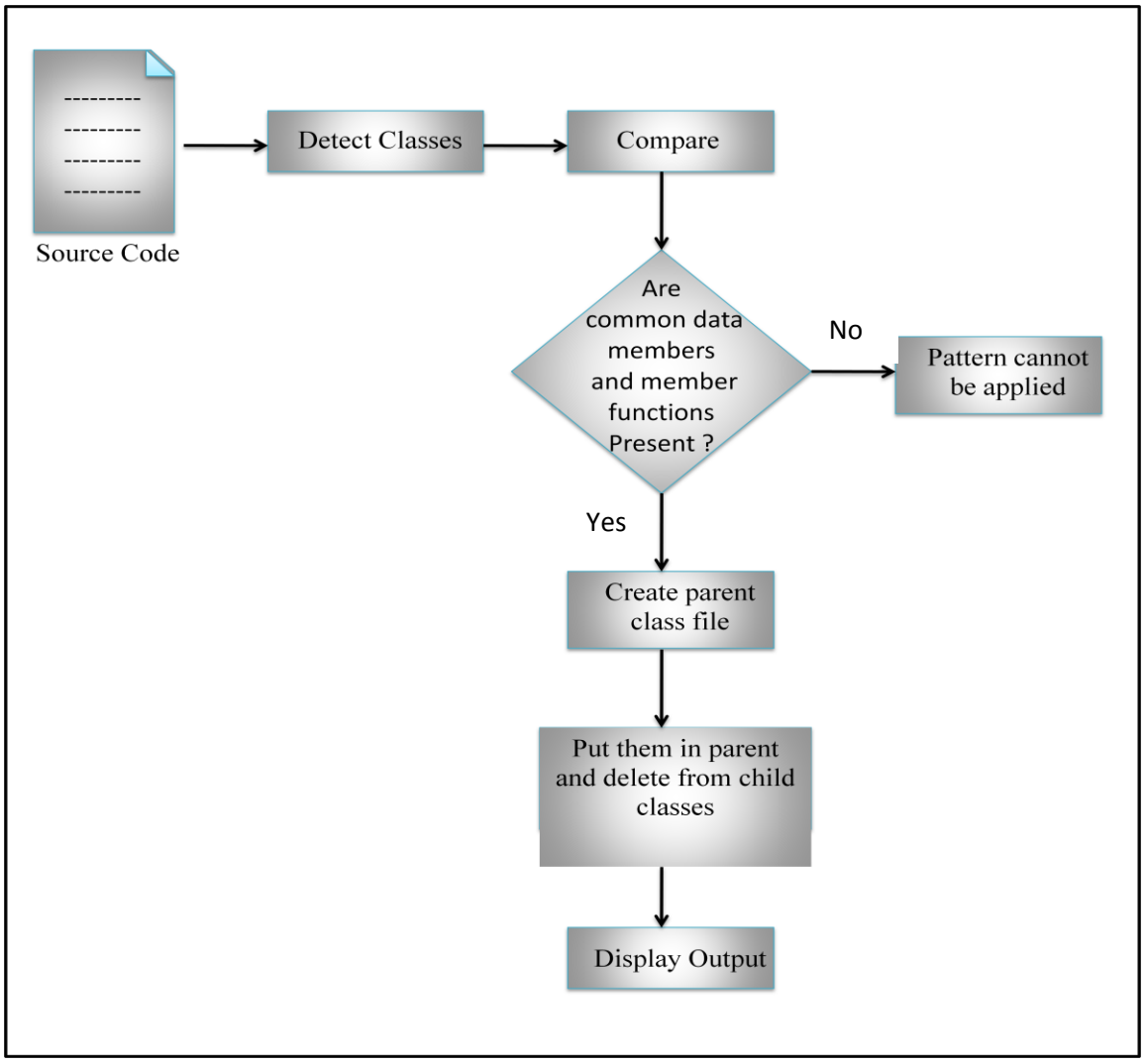


Figure 18: Flowchart of Pattern Detection

4.3.2 Algorithm of Pattern Detection Process

Input: Source code

Step 1 Detect classes from the given source code.

Step 2 Compare the classes detected from the source code for the commonalities (i.e. common data members and member functions).

Step 3 If there are no common data members and member functions present, then pattern cannot be applied.

Step 4 If common data members and member functions are present, then create a parent class file.

Step 5 Put the identified common data and member functions in the newly created parent file and delete from child classes.

Step 6 Display the output.

4.4 Syntactic Detection of Clones

Even after detection and removal of code at design level and further through pattern, there are some syntactic code clones that are needed to be detected. For this purpose P C Detector detects clones. Presently it is detecting Type1 and Type 2 clones. Later it can be easily enhanced to detect type 3 and type 4 clones.

4.4.1 Flaw in Type 2 Clones:

Type 2 clones are syntactically identical fragments, except for variations in identifiers, literals, types, whitespace, layout and comments. Two code fragments having same data structure and same number of literals with different name can be considered as clones of each other because they can replace each other as their outputs are same. But if two code fragments have renamed literals and different data structures, e.g. one code fragment has integer data type and second one has double data type, these two code fragments cannot be considered as clones as they can never produce same output.

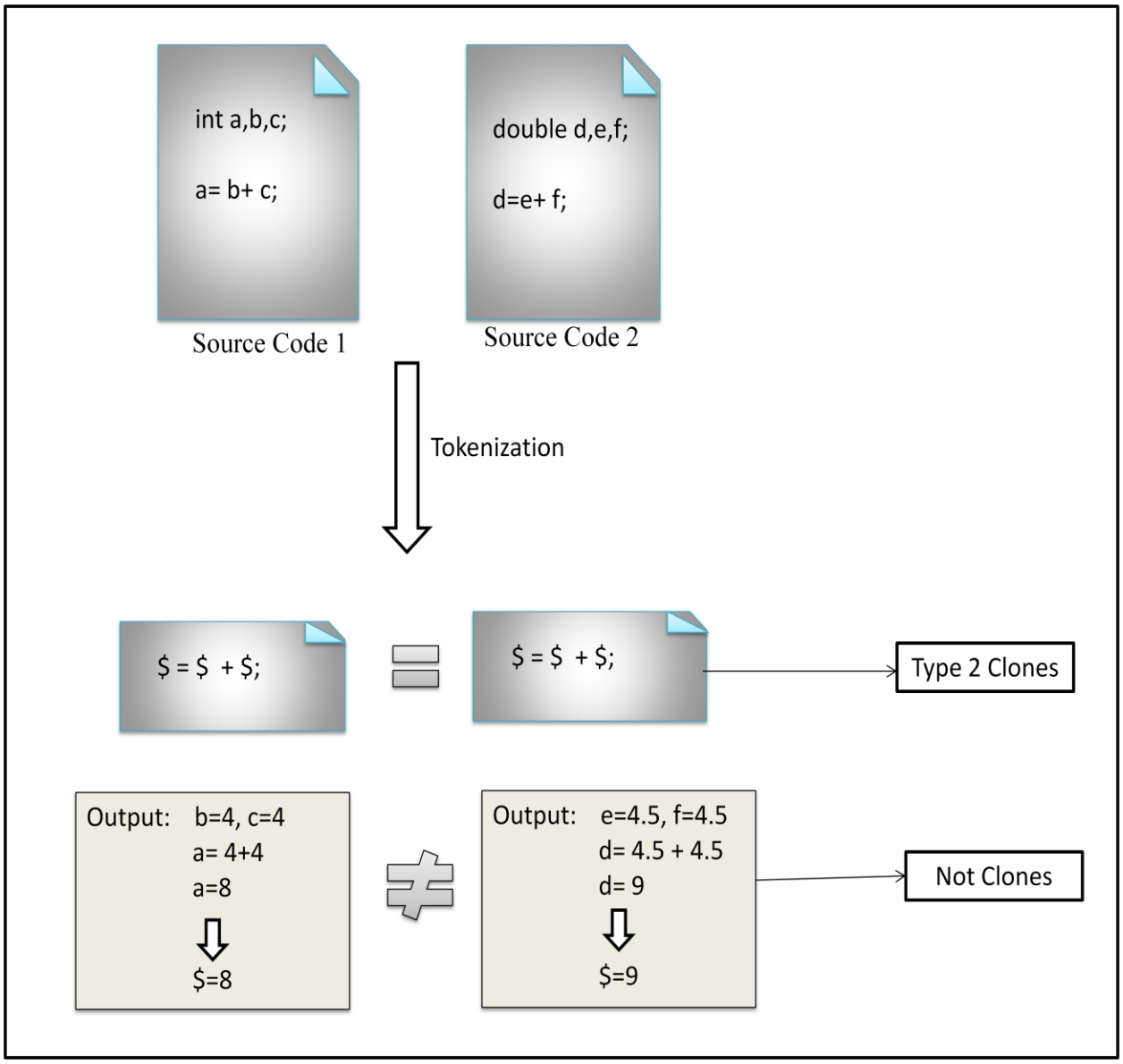


Figure 19: Example explaining the Flaw in Type 2 Clones with respect to Data types

This clause shows a flaw in the definition of type clones and is proved by schematic diagram shown in figure 19. There are two code fragments – source code 1 and source code 2. In order to detect Type 2 clones the variables are converted into similar tokens. The tokenized code shows that they are clones of each other but as shown in figure 19 they are producing different outputs. Hence during clone detection and comparison data types are needed to be considered.

4.4.2 Flowchart of Clone Detection Process

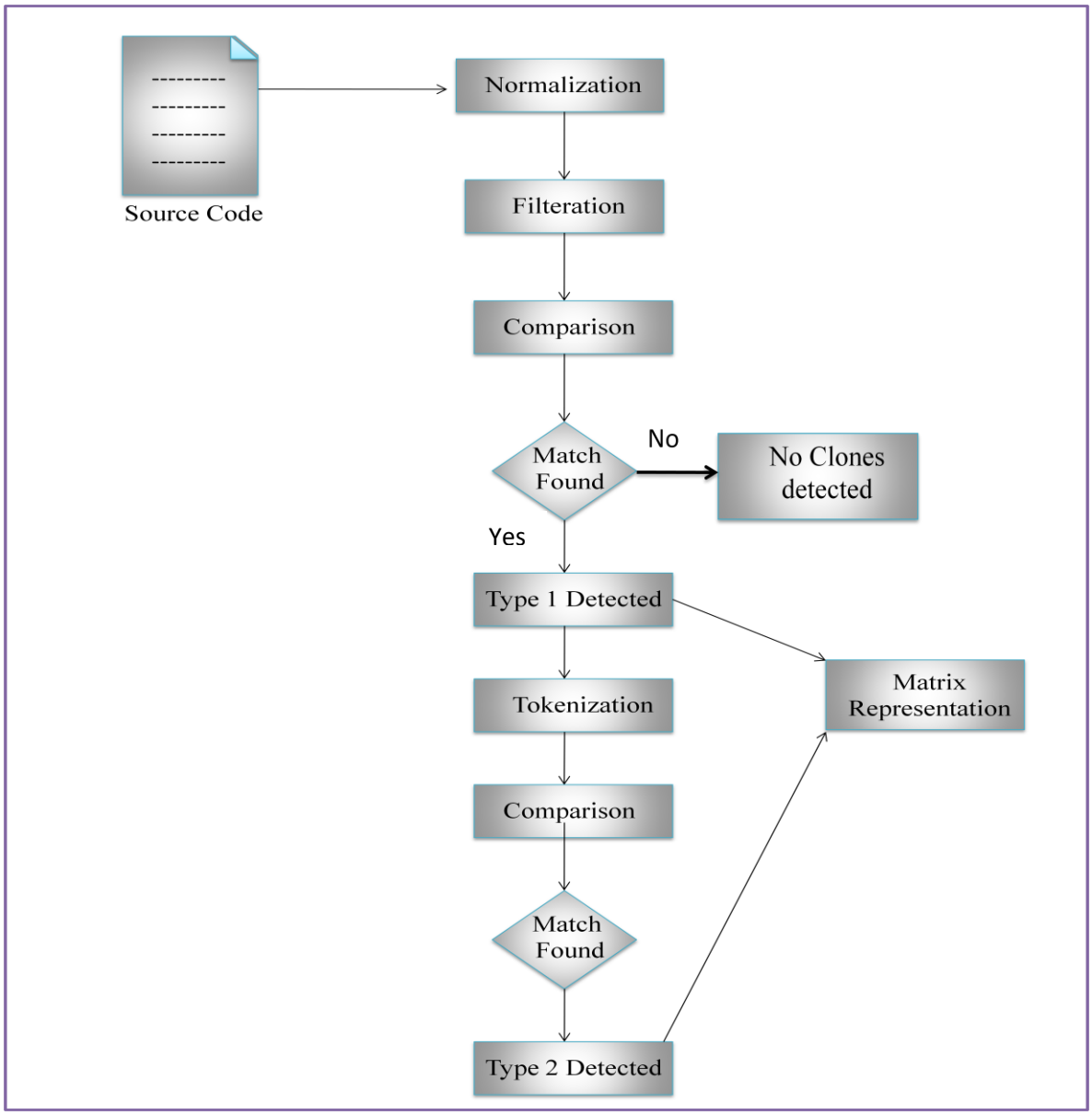


Figure 20: Flowchart of Clone Detection Type1 and Type 2

The flowchart in figure 20 shows the process of Clone detection of Type1 and Type 2 and gives the output in the form of matrix. In between the code is normalized, filtered and tokenized. The process is explained in the following algorithm.

4.4.3 Algorithm of Clone Detection Process

Input: Two source codes

Step 1 Take two source codes in which clones are to be detected.

Step 2 Normalize both the source codes by removing the comments and white spaces.

Step 3 Filtrate the source code obtained after normalization by removing the elements of the code such as void main(), header files, getch(), etc.

Step 4 Compare the two files after filtrations to check whether they are same or not.

a. If they are not same, it means there are no clones in the codes and exit.
b. If similarities are found, it means code contains Type I clones.

Step 5 After the identification of Type I clone, convert the literals present in the code into some type of token (a $ sign is taken here for this purpose).

Step 6 Compare the tokens of the two codes.

Step 7 If match is found, it means Type II clones are present in the two codes.

Step 8 Represent the identified clones of Type I and Type II in the form of a matrix.

Output: Matrix Representation of the identified clones.

4.5 PC Detector: Implementation and Working

Proposed clone detector is implemented in C# using development environment in . Net

i. The tool detects object oriented base Pull Up pattern from C++ source file and outputs the clone free enhanced code.
ii. The tool also extracts clones from C and C++ source files. It detects clone in same file or among different file and report its output in matrix format.

The working of tool is basically divided into two modules

i. Pattern detection
ii. Clone detection

4.5.1 Pattern Detection Process

In pattern detection the tool detects Pull Up pattern in source code and refactor the code by removing clones. The tools take C++ file as input by browsing it from location where it is placed. On detection of Pull Up, a new classes created with named as 'Parent ' class which contain all the common data member and member function of the classes. The common data members and member function are deleted from there respective classes and extracted up in the parent class.

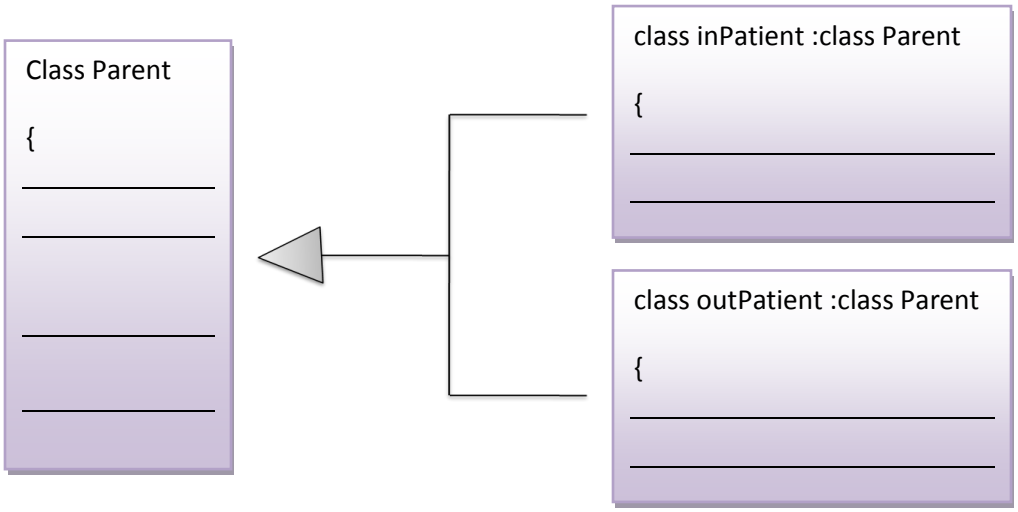


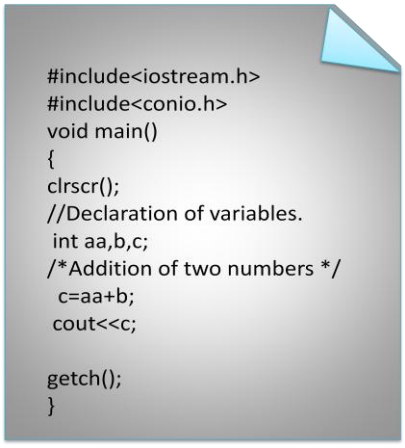
Figure 21: Example of Pull Up Pattern Detection

4.5.2 Clone Detection Process

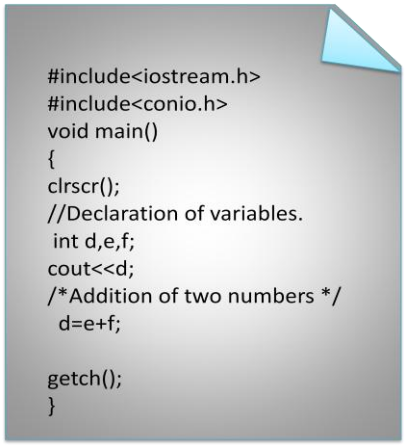
In clone detection module, the tool detects syntactical clones. This tool detects Type 1 and Type 2 clones. Type 1 clones remove white spaces, comment or Layout in the code fragment. Type 2 clones detects names of user defined identifiers - variables, constant, classes, methods etc and rename them as $ token. The clones can be present in the same file or among different files. Technique used for clone detection is string matching and parameterized String matching on the token stream of input code. The clones are detected in various stages as shown in figure 20 of flow chart. The clone detection has further two sub modules, one for detection of clone in same file and other for detection of clones in different files. The option is provided for the user to detect the clone from any file.

4.5.2.1 Clone-Detecting Process

Clone detection is a process in which the input is source files and the output is matrix showing which lines are having clones. The entire process of our clone detecting technique is shown in Figure 20. The process consists of four steps. Once the source file is input in the form of C or C++ file it goes through following stages for detection of Type 1 and Type 2 clones.



Source Code File 1

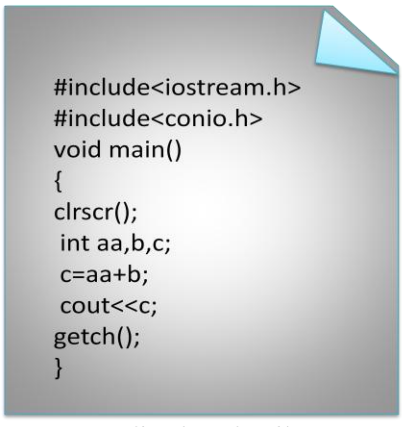


Source Code File 2

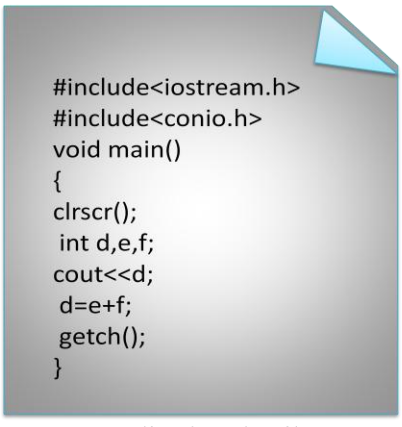
Figure 22: Sample Codes

Step 1: Normalization

In this phase the code (shown in figure 22) is normalized by removing all the comments or whitespaces. The code obtained is normalized (shown in figure 23) and is input to next stage of filtration.



Normalized Code File 1



Normalized Code File 2

Figure 23: Normalized Code

Step 2: Filtration

This phase is an important phase as it filters the codes from the common data like main function, header files that are commonly found in all codes yet not considered as clones as there detection does not matters in case of clone detection. Also other common functions like getch( ), clrscr( ), etc are also filtered from the code.

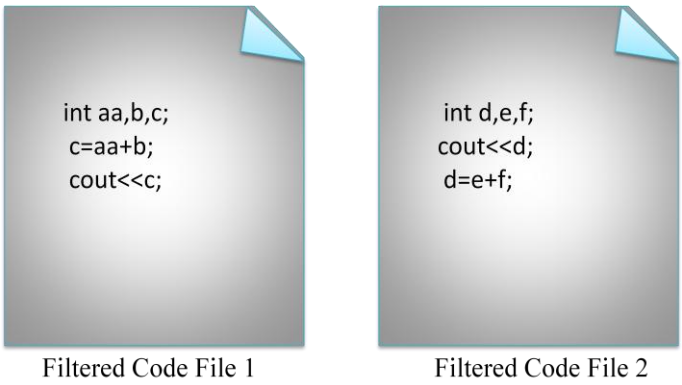


Figure 24: Filtered Code

Step 3: Comparison

This phase inputs the filtered code that is to be compared for clone detection. The textual comparison technique is used to find if any text matches. If any match is found it means they are clones of each other. Thus type 1 clones are detected.

Step 4: Tokenization

Each line of source file is divided into tokens corresponding to a lexical rule of the programming language. For detection of Type 2 clones we need to tokenize the variables, constants, class, method names as a special token '$' so as to check if they are same or not. The code is converted so as to detect type 2 clones that are different by names yet are clones of each other. As shown in the fig. 25, the two statements are same; their intent is same to add two values. Thus they are detected as Type 2 clones.

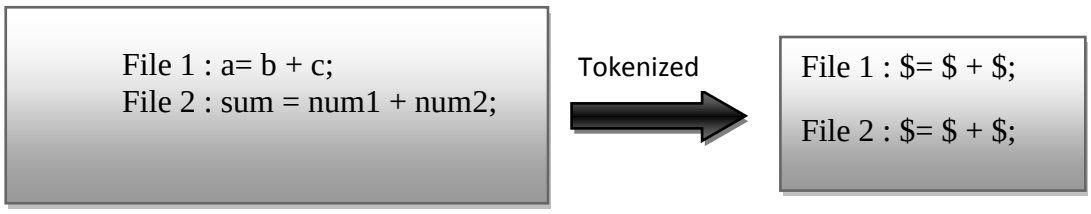


Figure 25: Example of Tokenization

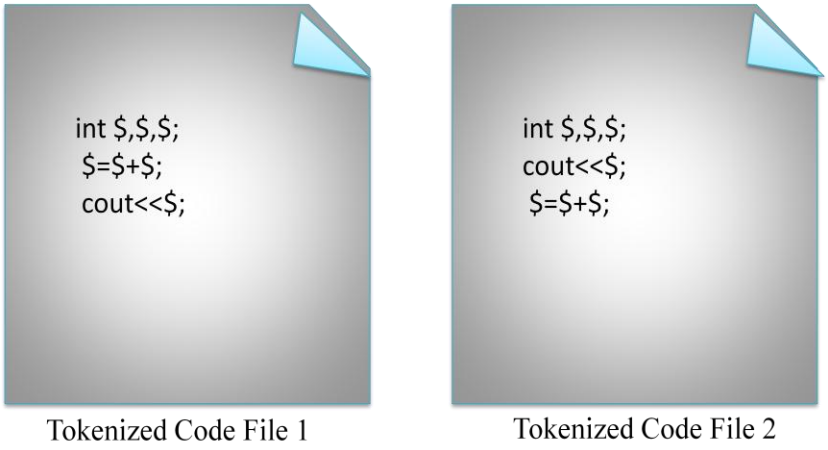


Figure 26: Tokenized Code

Step 5: Matrix Representation

The clones detected are represented in the form of matrix. The row of matrix represents line number of first file. The column of matrix represents line number of second file which is compared with first file to detect if clones are present. In case of detecting clones in same file, both row and column are represented by line numbers of same file. If a match occurs, it is represented in the matrix by value 1 or else by 0 values as shown in figure 27.

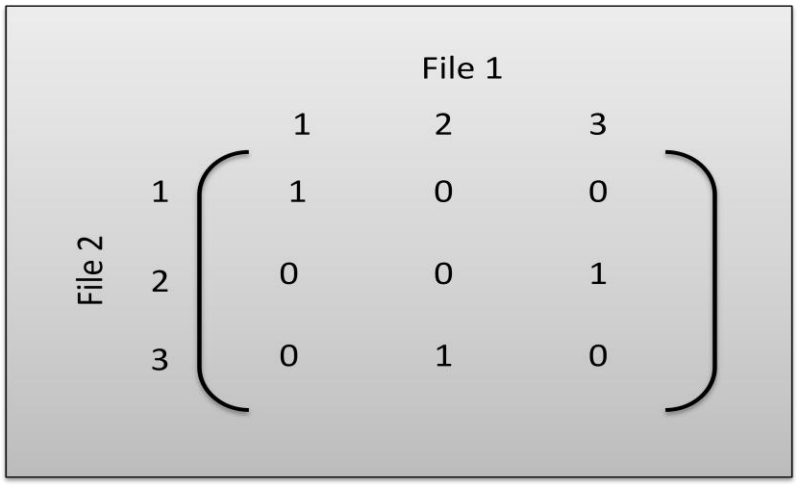


Figure 27: Matrix representing Clones present in two files

4.6 Tool Snapshots

4.6.1 Start up Form of PC Detector:

This screenshot shows the start up form of pattern and clone detector (PC Detector) tool. The tool is divided into two modules pattern detection and clone detection. After that main form is displayed.

Snapshot 1: Start up Form

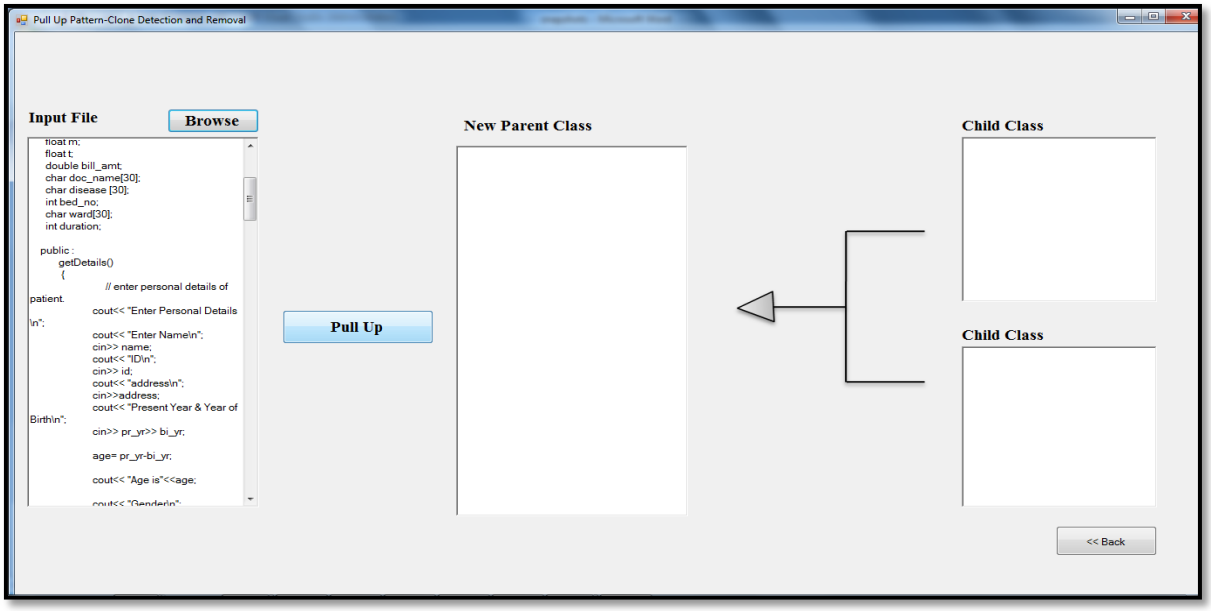
4.6.2 Main Form of PC Detector:

The tool is divided into two modules: pattern detection and clone detection. On clicking the pattern detection button a form opens for detecting the pull up pattern. On clicking clone detection button the file form is displayed.

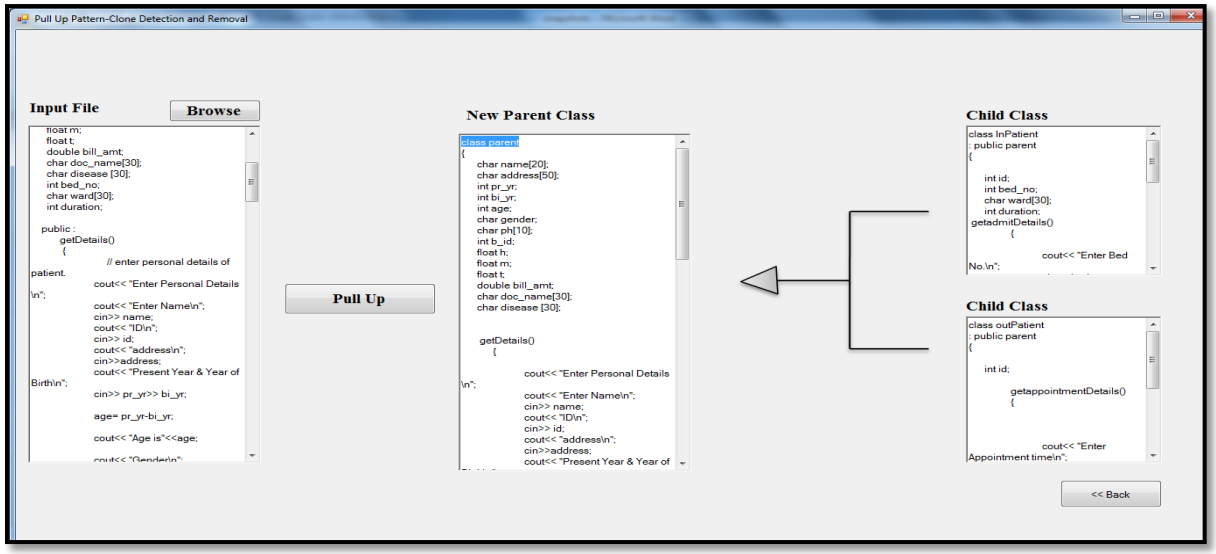
Snapshot 2: Main Form

4.6.3 Pull Up Pattern Module:

This is the pull up pattern form. C++ file is taken as input which contains classes from where its parent class can be made. C++ source file is browse and the file is displayed in first textbox. On clicking to pull up button, a new class is created named as class parent and is displayed in the second text box. The other classes become its child classes and are displayed in their respective textboxes.



Snapshot 3: (a) Pull Up Pattern Input



Snapshot 3: (b) Pull Up Pattern Output

4.6.3.1 Input C++ File under Case Study of Hospital Management System

Class InPatient

```
#include<iostream.h>
#include<conio.h>
class InPatient
{
     char name[20];
     int id;
     char address[50];
     int pr_yr;
     int bi_yr;
     int age;
     char gender;
     char ph[10];
     char doc_name[30];
     char disease [30];
     int bed_no;
     char ward[30];
     int duration;

   public :
         getDetails()
          {
                        // enter personal details of
patient.
                    cout<< "Enter Personal
Details\n";
                    cout<< "Enter Name\n";
                    cin>> name;
                    cout<< "ID\n";
                    cin>> id;
                    cout<< "address\n";
                    cin>>address;
                    cout<< "Present Year & Year of
Birth\n";
                    cin>> pr_yr>> bi_yr;
                    age= pr_yr-bi_yr;
                   cout<< "Age is"<<age;
                   cout<< "Gender\n";
                    cin>>gender;
                    cout<< "Phone_No\n";
                    cin>> ph;

                    // Enter all hospital bill related
details.
                    cout<< "Enter Bill Id\n";
                    cin>> b_id;
                    //Enter doctor details.
                    cout<< "Enter consulting
doctor\n";
                    cin>> doc_name;
                    cout<< "Type of disease";
                    cin>> disease;
                }
              getadmitDetails()
             {
                    //admit related details.
                       cout<< "Enter Bed No.\n";
                       cin>> bed_no;
                       cout<< "Ward ";
                       cin>> ward;
                       cout<<"Enter admit duration";
                       cin>> duration;
                       cout<<"Enter discharge date\n;
                       cin>>discharge_date;
             }
 public :
             displayDetails()
             {
                 cout<<"Details are as follows:\n";
                  cout<<"NAME:"<< name;
                  cout<<"ID:"<< id;
                  cout<<"ADDRESS:"<< address;
                  cout<<"AGE:"<< age;
                  cout<<"GENDER:"<< gender;
                  cout<<"CONTACT_NO:"<< ph;
                  cout<<"BILL ID:"<< b_id;
                  cout<<"BILL AMOUNT:"<< bill_amt;
                  cout<<"CONSULTING DOCTOR
NAME:"<< doc_name;
                  cout<<"TYPE OF DISEASE:"<<
disease;
             }

             displayadmitDetails()
             {
                       cout<< "Bed No.:"<<bed_no;
                       cout<< "Ward:"<<ward;
                       cout<<"Admit
Duration:"<<duration;
                       cout<<"Discharge
Date:"<<discharge_date;

             }

};
```

Class OutPatient

```
#include<iostream.h>
#include<conio.h>
class OutPatient
{
     char name[20];
     int id;
     char address[50];
     int pr_yr;
     int bi_yr;
     int age;
     char gender;
     char ph[10];
     char doc_name[30];
     char disease [30];
     time app_time;
   public :
         getDetails()
          {
                        // enter personal details of
patient.
                    cout<< "Enter Personal
Details\n";
                    cout<< "Enter Name\n";
                    cin>> name;
                    cout<< "ID\n";
                    cin>> id;
                    cout<< "address\n";
                    cin>>address;
                    cout<< "Present Year & Year of
Birth\n";
                    cin>> pr_yr>> bi_yr;
                    age= pr_yr-bi_yr;
                   cout<< "Age is"<<age;
                   cout<< "Gender\n";
                    cin>>gender;
                    cout<< "Phone_No\n";
                    cin>> ph;

                    // Enter all hospital bill related
details.
                    cout<< "Enter Bill Id\n";
                    cin>> b_id;
                    //Enter doctor details.
                    cout<< "Enter consulting
doctor\n";
                    cin>> doc_name;
                    cout<< "Type of disease";
                    cin>> disease;
                }
              getappointmentDetails()
             {
                      //Appointment detail.
                       cout<< "Enter Appointment
time\n";
                       cin>>time;

             }
     public :
            displayDetails()
            {
                cout<<"Details are as follows:\n";
                 cout<<"NAME:"<< name;
                 cout<<"ID:"<< id;
                 cout<<"ADDRESS:"<< address;
                 cout<<"AGE:"<< age;
                 cout<<"GENDER:"<< gender;
                 cout<<"CONTACT_NO:"<< ph;
                 cout<<"BILL ID:"<< b_id;
                 cout<<"BILL AMOUNT:"<< bill_amt;
                 cout<<"CONSULTING DOCTOR
NAME:"<< doc_name;
                 cout<<"TYPE OF DISEASE:"<<
disease;
            }

            displayAppointmentDetails()
            {
              cout<< "Appointment time:"<<time;
                }
};
```

4.6.3.2 Output File: Parent and Child Classes after Pattern detection and Clone Removal

```
class parent
{
      char name[20];
      char address[50];
      int pr_yr;
      int bi_yr;
      int age;
      char gender;
      char ph[10];
      char doc_name[30];
      char disease [30];

       getDetails()
           {
                    cout<< "Enter Personal
Details\n";
                    cout<< "Enter Name\n";
                    cin>> name;
                    cout<< "ID\n";
                    cin>> id;
                    cout<< "address\n";
                    cin>>address;
                    cout<< "Present Year & Year of
Birth\n";
                    cin>> pr_yr>> bi_yr;
                    age= pr_yr-bi_yr;
                    cout<< "Age is"<<age;

                    cout<< "Gender\n";
                    cin>>gender;
                    cout<< "Phone_No\n";
                    cin>> ph;
                    cout<< "Enter consulting
doctor\n";
                    cin>> doc_name;
                    cout<< "Type of disease";
                    cin>> disease;
              }
        displayDetails()
             {

                        cout<<"Details are as
follows:\n";
                  cout<<"NAME:"<< name;
                  cout<<"ID:"<< id;
                  cout<<"ADDRESS:"<< address;
                  cout<<"AGE:"<< age;
                  cout<<"GENDER:"<< gender;
                  cout<<"CONTACT_NO:"<< ph;
                  cout<<"BILL ID:"<< b_id;
                  cout<<"BILL AMOUNT:"<< bill_amt;
                  cout<<"CONSULTING DOCTOR
NAME:"<< doc_name;
                  cout<<"TYPE OF DISEASE:"<<
disease;
             }
```

```
 class InPatient: public parent
{

      int id;
      int bed_no;
      char ward[30];
      int duration;
 getadmitDetails()
             {

                        cout<< "Enter Bed No.\n";
                        cin>> bed_no;
                        cout<< "Ward ";
                        cin>> ward;
                        cout<<"Enter admit duration";
                        cin>> duration;
                        cout<<"Enter discharge date\n;
                        cin>>discharge_date;
             }
             displayadmitDetails()
             {
                        cout<< "Bed No.:"<<bed_no;
                        cout<< "Ward:"<<ward;
                        cout<<"Admit
Duration:"<<duration;
                        cout<<"Discharge
Date:"<<discharge_date;

             }
};
```

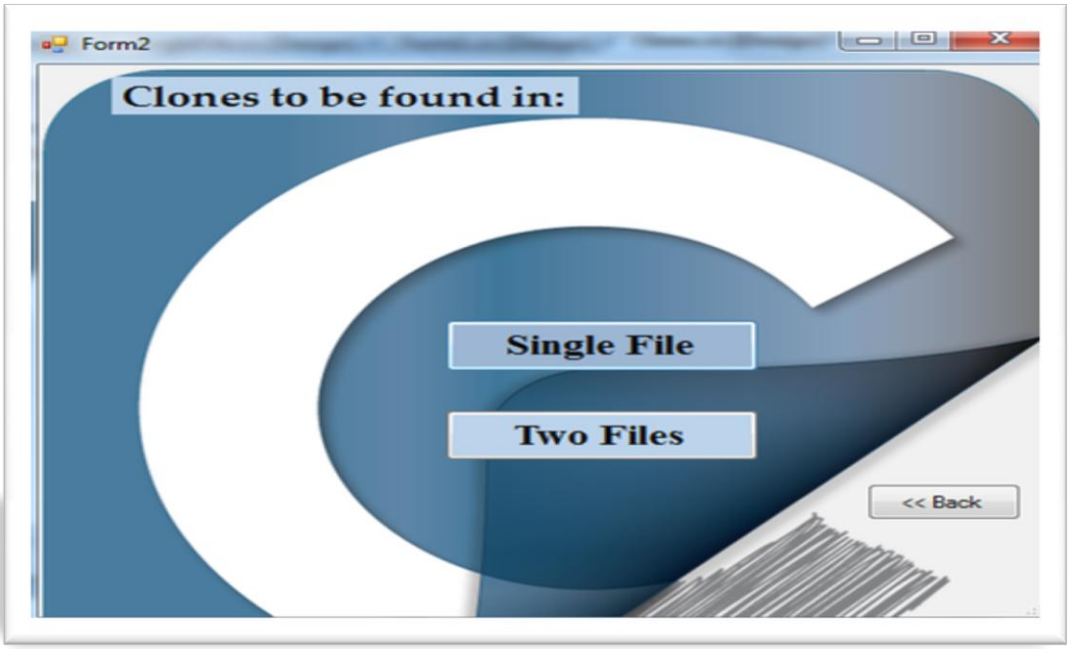
```
class outpatient: public parent
{

      int id;

             getAppointmentDetails()
             {



                        cout<< "Enter Appointment
time\n";
                        cin>>time;

              }
             displayappointmentDetails()
             {

                        cout<< "Appontment
time:"<<time;


               }
};
```

4.6.4 Clone Detection Module:

On clicking clone detection button another form is displayed to detect clones in same file or in other files. On clicking to single file or two files button their respective forms are displayed.

Snapshot 4: Clone Detection Module

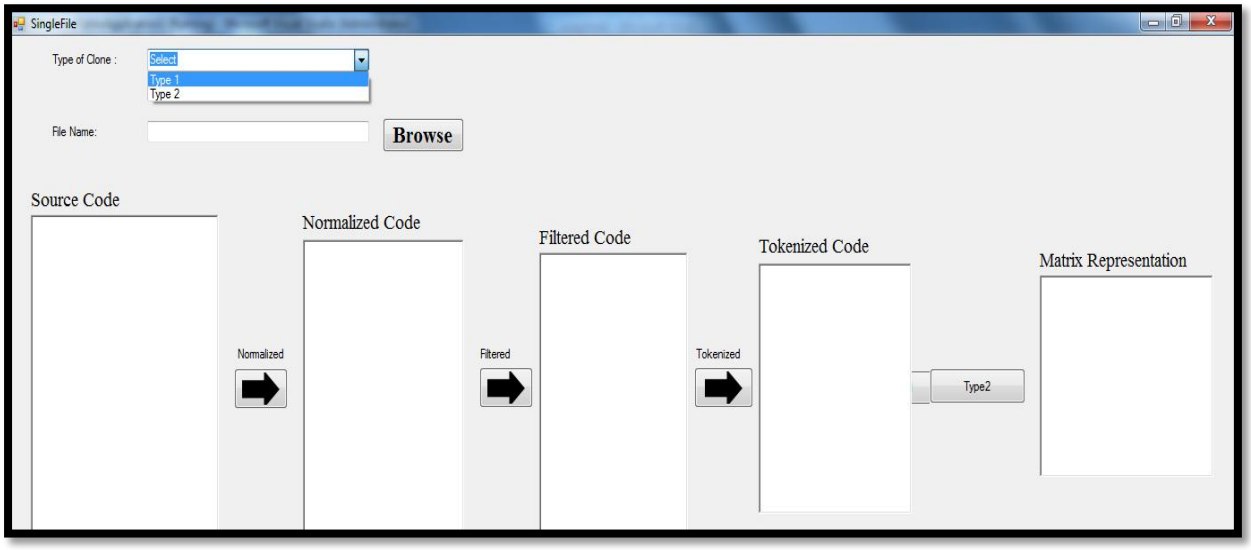
The tool has the option to identify duplicated code between several files and within the same file by clicking on single file or two file button.



Snapshot 5: Clone Detection in Single and in Two Files

4.6.5 Single File Form:

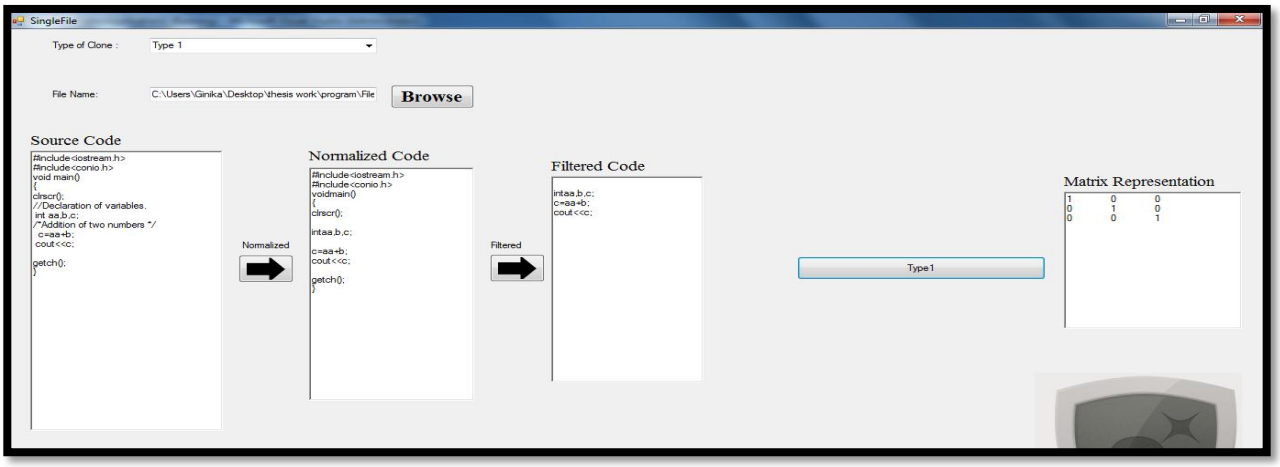
This form detects clone in single file. Firstly a combo box is there to choose which type of clones is to be detected-Type1 or Type2. A file is browsed and is displayed in source code text box. Various steps of clone detection are then shown in the text boxes and finally they are represented in matrix format.



Snapshot 6: Single File Form

4.6.5.1 Type 1 Clone

Type 1 Clone detection is shown in following snapshot. A C++ file is browsed; it is converted to normalized form. It is then filtered and finally represented in form of matrix showing the clones.

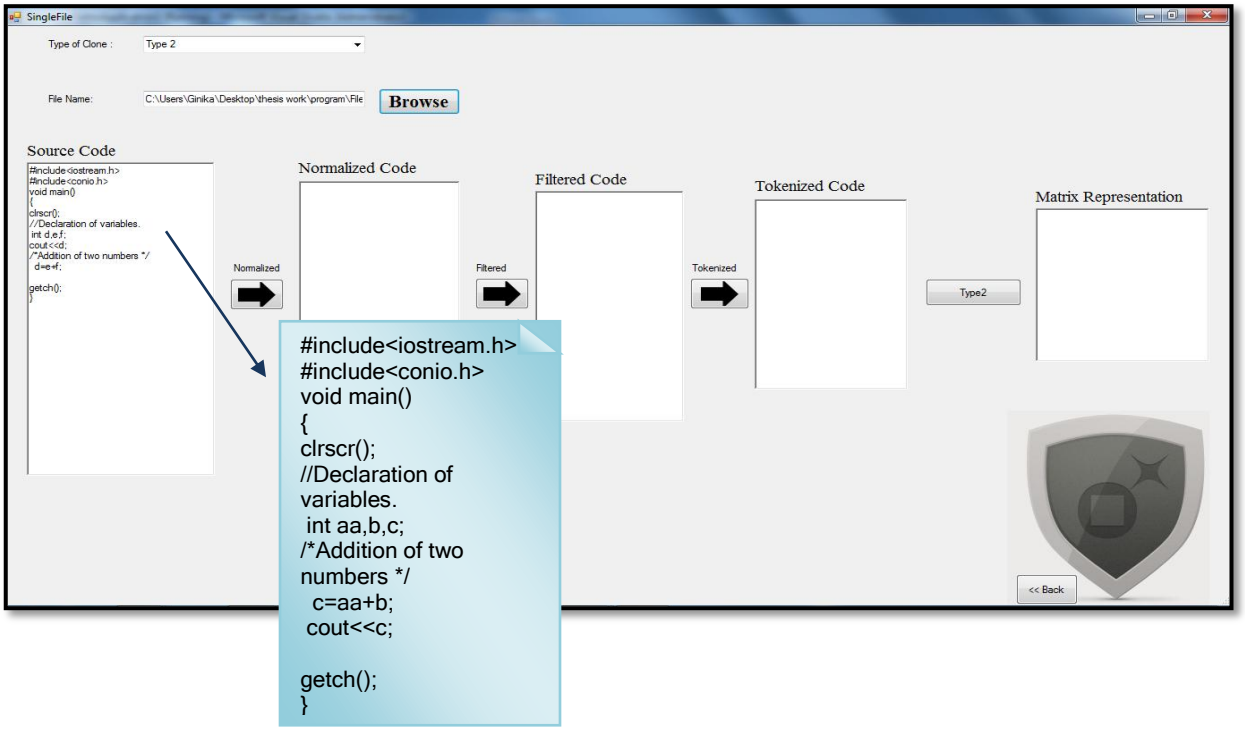


Snapshot 7: Type 1 Clone

4.6.5.2 Type 2 Clones

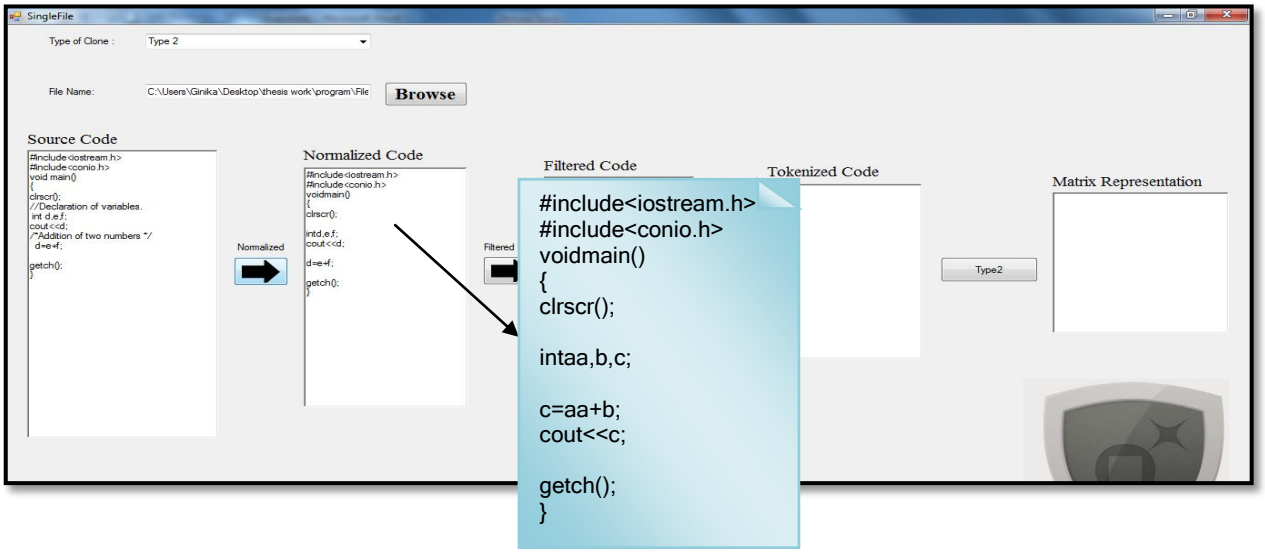
Type 2 clones are detected and shown in matrix form in following steps:

Step 1: Input Source Code



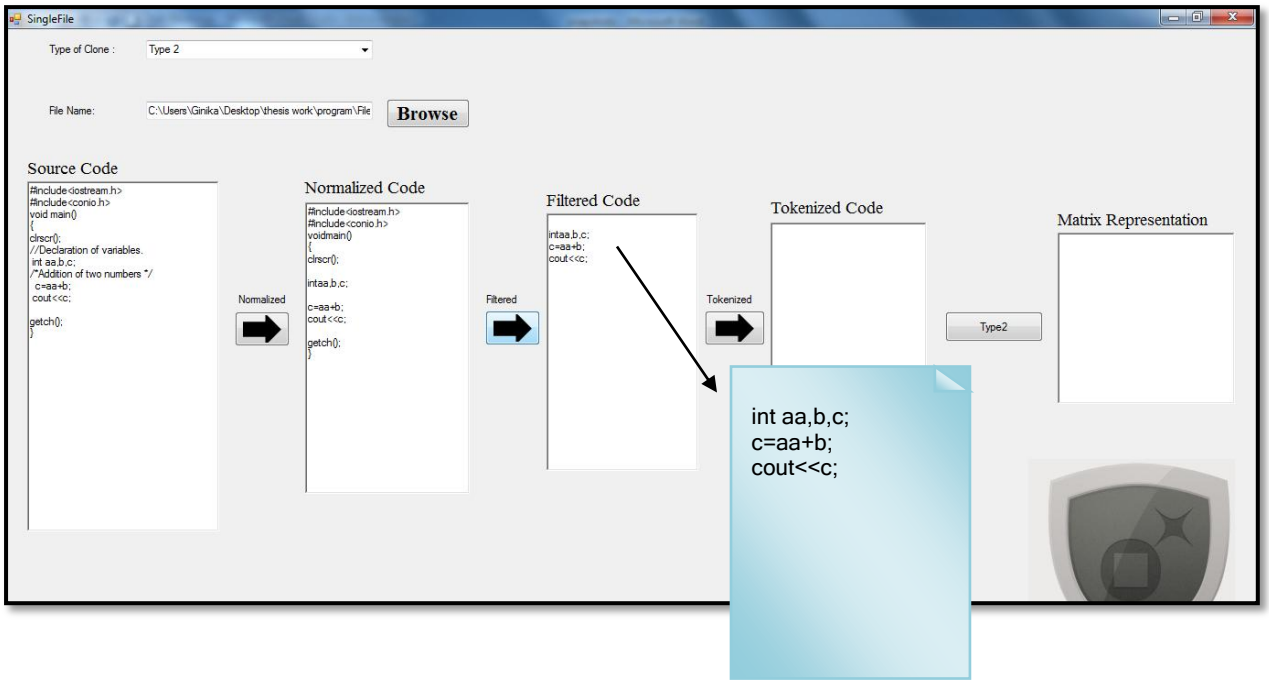
Snapshot 8: Type 1 Clone - Input Source Code

Step 2: Source code converted to Normalized code



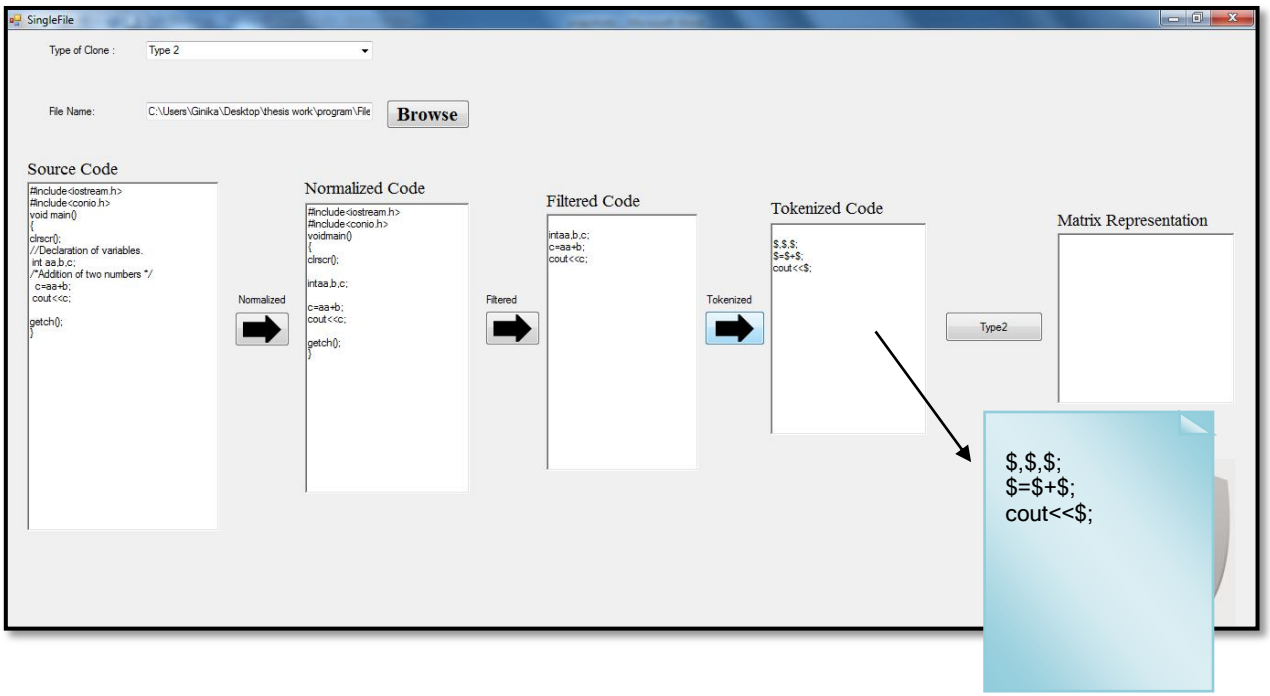
Snapshot 9: Type 1 Clone - Source code converted to Normalized code

Step 3: Normalized code converted to Filtered code



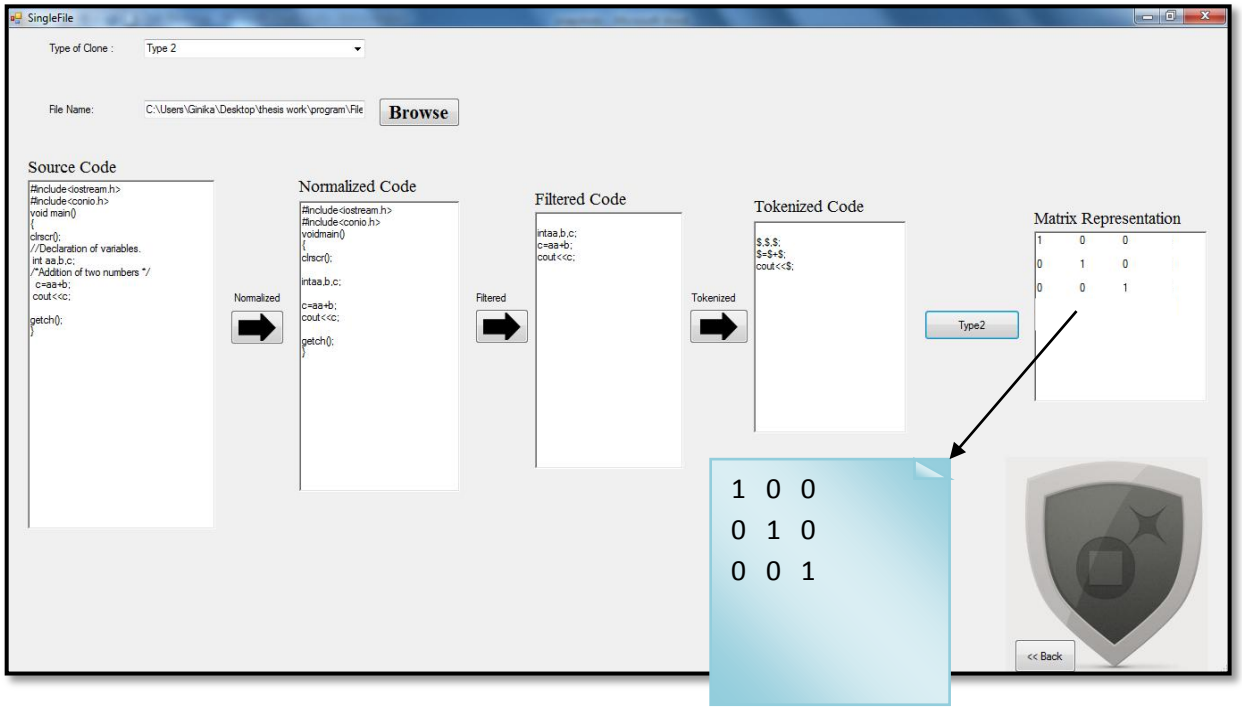
Snapshot 10: Type 1 Clone - Normalized code converted to Filtered code

Step 4: Filtered code converted into Tokens



Snapshot 11: Type 1 Clone - Filtered code converted into Tokens

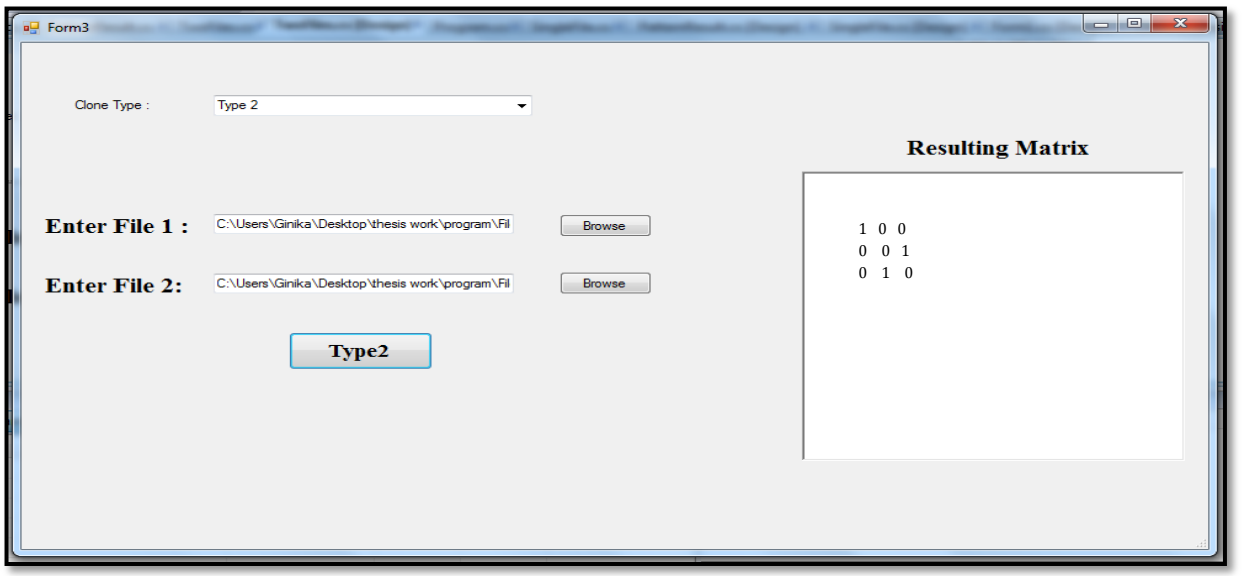
Step 5: Matrix Representation of Clones



Snapshot 12: Matrix Representation of Clones

4.6.6 Clone Detection in Two Files

This form detects the clones in two different files. The steps followed are same as followed in detecting clones in same file.



Snapshot 13: Clone Detection in Two Files

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The proposed methodology addresses the problem of clone detection and removal. The clone detection and removal is done using the 3-way approach of integrating Model Based Visual Analysis using UML, Pattern Based Semantic Analysis using P C Detector tool and Syntactically Code Analysis to detect Type1 and Type 2 clones using P C Detector.

It is possible to outwit the hindrance of clones by applying a 3-way approach of detecting the clones at design and code level. The process is automated by developing a tool that requires no parsing yet is able to detect a significant amount of code duplication.

Since code clones are believed to reduce the maintainability of software, several code clone detection techniques and tools have been proposed. This thesis proposes a new Pattern and Clone Detection approach as PC Detector tool that works on the transformation of input source text into normalized code, then filtering the code, comparing and detecting Type 1 clones. Also a token-by-token comparison is done on the filtered code to detect type 2 clones.

Summarizing the Work Done:

i. An effective 3-way approach is proposed for the analysis of clones, their detection, and removal thus enhancing the quality and maintenance of software.
ii. Detecting the clones at design level by using UML diagrams and hence removing them.
iii. Applying patterns at code level to detect and remove semantic clones. After semantic code removal, the code is further analyzed to detect syntactical clones.
iv. Developing Pattern and Clone Detection tool and hence automating the task of pattern and syntactical clone detection.

5.2 Future Work

i. PC Detector detects Type 1 and Type 2 clones. This filtered token based clone detection method, is very efficient and scalable to large software systems. This can further be extended to detect Type3 and Type4 clones.

ii. PC Detector can easily detect Refactoring Patterns. Presently, it is working on Pull Up Pattern only but it can be extended to detect other cloning patterns like Extract, Template, and Strategy Patterns.

iii. Cloning at design level can further be enhanced by automating the Model Based Analysis.

iv. Currently PC Detector detects clones in C and C++ source files. This tool can be adapted to other Object Oriented Languages like Java, C#, etc.

v. Software Cloning can be integrated in Agile Experiments which can improve its maintenance phase.

References

[1] Lakhotia A., Li J., Walenstein A. and Yang Y., "Towards a Clone Detection Benchmark Suite and Results Archive", in *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, pp. 285-286, 2003.

[2] Baker S., "A Program for Identifying Duplicated Code", *Computing Science and Statistics*, vol. 24, pp. 49-57, 1992.

[3] Boehm B. W., "A Spiral Model of Software Development and Enhancement", *Software Engineering Notes*, vol. 11, no. 4, 1994.

[4] Roy C. K., Cordy J. R. and Koschke R., "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", *Science of Computer Programming*, vol. 74, no. 7, pp. 470-495, May 2009.

[5] Roy C. K. and Cordy J. R., "A Survey on Software Clone Detection Research", Queen's School of Computing, Technical Report No. 2007-541, vol. 115, September 2007.

[6] Kapser C. and Godfrey M., "Cloning Considered Harmful", in *Proceedings of the 13th Working Conference on Reverse Engineering*, pp. 19-28, 2006.

[7] Kapser C. and Godfrey M., "A Taxonomy of Clones in Source Code: The Reengineers Most Wanted List", in *Proceedings of Working Conference on Reverse Engineering*, 2003.

[8] Liu C., Chen C., Han J. and Yu P. S., "GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis", in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 872-881, August 2006.

[9] Krueger C. W., "Software Reuse", *ACM Computing Surveys (CSUR)*, vol. 24, no. 2, pp. 131-183, June 1992.

[10] Roberts D. B., "Practical Analysis for Refactoring", Ph.D. thesis, University of Illinois, 1999.

[11] Choi E., Yoshida N., Ishio T., Inoue K. and Sano T., "Finding Code Clones for Refactoring with Clone Metrics: A Case Study of Open Source Software", in *Proceedings of The Inst. Of Electronics, Information and Communication Engineers (IEICE)*, pp. 53-57, July 2011.

[12] Nickell E. and Smith I., "Extreme Programming and Software Clones", in *Proceedings of Second International Workshop on the Detection of Software Clones (IWDSC)*, November 2003.

[13]"Extreme Programming: A Gentle Introduction", available at http://www.extremeprogramming.org, February 2002.

[14] Brooks F., "The Mythical Man-Month: Essays On Software Engineering, Reading Mass", Addison-Wesley Pub. Co., 1975.

[15] Rahman F., Bird C., Devanbu P., "Clones: What is that smell?", accepted to Empirical Software Engineering, an International Journal 2011 Springer-Verlag.

[16] Rysselberghe F. V., Demeyer S., "Evaluating Clone Detection Techniques" in *Proceedings of 19th IEEE International Conference on Automated Software Engineering (ASE)*, pp. 336-339, September 2004.

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

[18] Mili H., Mili A., Yacoub S. and Addy E., "Reuse Based Software Engineering: Techniques, Organization, and Controls", 2002.

[19] Johnson J.," Identifying Redundancy In Source Code Using Fingerprints", in *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: Software Engineering*, pp. 171-183, 1993.

[20] Mayrand J., Leblanc C. and Merlo E., "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics," in *Proceedings of the 12th 45 International Conference on Software Maintenance (ICSM'96)*, pp. 244–253, November 1996.

[21] Philipps J. and Rumpe B., "Roots of Refactoring", in *Tenth OOPSLA Workshop on Behavioral Semantics*, October 2001.

[22] Beck K., "Extreme Programming Explained: Embrace Change", Addison-Wesley, 2000.

[23] Kim M., Sazawal V., Notkin, D. and Murphy, "An Empirical Study Of Code Clone Genealogies", in *Proceedings of European Software Engineering Conference and Foundations of Software Engineering*, pp. 187-196, 2005.

[24] Lindstrom L. and Jeffries R., "Extreme Programming and Agile Software Development Methodologies", *Information Systems Management*, vol. 2, no. 13, pp. 41-52, 2004.

[25] Jones M., "Remix and Reuse of Source Code in Software Production", Ph.D. Thesis, University of Illinois, 2010.

[26] Fowler M. and Beck K., "Refactoring: Improving the Design of Existing Code", The Addison Wesley Object Technology Series, Addison Wesley, Boston, 2000.

[27] Kim M., Sazawal V., Notkin D. and Murphy G., "An Empirical Study of Code Clone Genealogies", in *Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 187-196, 2005.

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

[29] Jablonski P. and Hou D., "Aiding Software Maintenance with Copy-and-Paste Clone-Awareness", in *Proceedings of IEEE 18th International Conference on Program Comprehension*, pp. 170-179, 2010.

[30] Al-Ekram R., Kapser C., Holt R. and Godfrey M., "Cloning by Accident: An Empirical Study of Source Code Cloning across Software Systems", in *Proceedings of the International Symposium on Empirical Software Engineering*, pp. 376-385, 2005.

[31] Moser R., Sillitti A., Abrahamsson P. and Succi G., "Does Refactoring Improve Reusability?", in *Proceedings of the 9th international conference on Reuse of Off-the-Shelf Components*, pp. 287-297, 2006.

[32] Ducasse S., Rieger M. and Demeyer S., "A Language Independent Approach For Detecting Duplicated Code", in *proceedings of the IEEE International Conference on Software Maintenance*, pp. 109-119, 1999.

[33] Demeyer S., Ducasse S. and Nierstrasz O., "Finding Refactorings via Change Metrics", in *Proceedings of the 15th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pp. 166-177, 2000.

[34] Kamiya T., Kusumoto S., and Inoue K., "CCFinder: A MultiLinguistic Token-Based Code Clone Detection System for Large Scale Source Code", *Journal IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654-670, July 2002.

[35] Kataoka Y., Ernst M., Griswold W. and Notkin D., "Automated Support For Program Refactoring Using Invariants", in *proceedings of International Conference on Software Maintenance (ICSM '01)*, pp. 736-743, November 2001.

[36] Higo Y., Kamiya T., Kusumoto S. and Inoue K., "ARIES: Refactoring Support Environment Based on Code Clone Analysis", in *proceedings of 8th IASTED International Conference on Software Engineering and Applications*, 2004.

[37] Krinke J., "Is Cloned Code older than Non-Cloned Code?", in *Proceedings of the 5th International Workshop on Software Clones*, pp. 28-33, 2011

[38] Reifer J. D., "How to Get the Most out of Extreme Programming/Agile Methods", in *Proceedings of the Second XP Universe and First Agile Universe Conference on Extreme Programming and Agile Methods*, pp. 185-196, 2002.

[39] Mens T. and Demeyer S., "Software Evolution", Ph.D. Thesis, Universität Bremen, Germany, 2007.

[40] Basit H. A. and Jarzabek S., "A Case of Structural Clones", in *3rd International Workhsop on Software Clones*, 2009.

[41] Bois B.D., Amsel A., Eetvelde N.V., Stenten H., Demeyer S., Mens T. and Gorp P.V, "A Discussion of Refactoring in Research and Practice", *Computer and Information Science*, 2004.

[42] Deissenboeck F., Hummel B., Juergens E., Schätz B., Wagner S., Girard J., Teuchert S., "Clone Detection in Automotive Model-Based Development", in *Proceedings Of The 30th International Conference on Software Engineering*, pp. 603-612, 2008.

[43] Störrle H., "Towards Clone Detection in UML Domain Models", in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pp. 285-293, 2010.

[44] Storrle H., "Towards Clone Detection in UML Domain Models", PhD Thesis, Technical University of Denmark (DTU), 2011.

[45] Kapser C., Anderson P., Godfrey M., Koschke R., Rieger M., Rysselberghe F. and Weigerber P, "Subjectivity In Clone Judgment: Can We Ever Agree?", Technical Report 06301,

[46] "Unified Modeling Language",
Available at http://en.wikipedia.org/wiki/Unified_Modeling_Language

[47] Koschke R., "Survey of research on software clones. In Duplication, Redundancy, and Similarity in Software" (2006), A. Walenstein, R. Koschke, and E. Merlo, Eds., no. 06301 in Dagstuhl Seminar Proceedings, Intl. Conf. and Research Center for Computer Science, Dagstuhl Castle.

[48] Tiarks R., Koschke R. and Falke R, "An Assessment of Type-3 Clones as Detected by State-of-the-Art Tools", *in Proceedings of the 2009 Ninth IEEE International Working Conference on Source Code Analysis and Manipulation*, pp. 67-76, 2009.

[49] Higo Y., Kamiya T., Kusumoto S. and InoueK., "Refactoring Support Based on Code Clone", *Computer and Information Science*, vol. 3009, pp. 220-233, 2004.

[50] Fowler M., "Refactoring: Improving the Design of Existing Code", Addison Wesley, 1999.

[51] "Refactoring", available at http://sourcemaking.com/refactoring.

[52] "Refactoring Patterns", available at http://www.dofactory.com/Patterns.aspx.

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

[54] Robert S., "Tutorial on Software Restructuring", Chapter: An introduction to software restructuring, IEEE Press, 1986.

[55] William F., "Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks", PhD Thesis, University of Illinois, 1992.

[56] Tairas R., "Representation, Analysis and Refactoring Techniques to Support Code Clone Maintenance", PhD Thesis, University of Alabama, 2010.

List of Publications

1. Mahajan G., and Ashima., "Software Cloning in Extreme Programming Environment", *International Journal of Research in IT and Management*, vol. 2, no. 2, pp. 1906-1919, February 2012.(Published)

2. Mahajan G., and Ashima., "Software Cloning in Extreme Programming Environment", in *Proceedings of International Conference on Competitiveness and Innovativeness in Engineering, Management and Information Technology*, January 2012.(Published)

3. Mahajan G., and Ashima, "Implementing a 3-Way Approach of Clone Detection and Removal using PC Detector Tool", *in Proceedings of CSI 6th International Conference on Software Engineering*, 2012. (Communicated)