“Code Point” – Software Code Clone Analysis Visualizer

Prepared by

K.G.D.K.I. Seneviratna

(158775M)

Supervised by

Mr. Chaman Wijesiriwardana

Master of Science in Information Technology

University of Moratuwa, Sri Lanka

2019

“Code Point” – Software Code Clone Analysis Visualizer

Prepared by

K.G.D.K.I. Seneviratna

(158775M)

Supervised by

Mr. Chaman Wijesiriwardana

Master of Science in Information Technology

University of Moratuwa, Sri Lanka

2019

Declaration

Declaration We declare that this thesis is our own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Full Name of the Student:	Kavisha Seneviratna
Registration No:	158775M
Signature of Student:	....
Date:	15/05/2019

Supervised by;

Name of Supervisor: Mr. Chaman Wijesiriwardana

Abstract

Code clone detection is a common practice in the software industry. In order to maintain quality code, identify code clones and take relevant actions to remove duplicates. And also, that can use to measure the quality of the work of the developer. Clone detection results are often voluminous and difficult to present. Most clone presentations focus on the quantitative clone results but do not relate them to the structure of the analyzed system. This relevant study implies the code clone detection techniques and currently available visualization solution. The proposed system is to fulfill the gaps between currently available code clone visualization techniques which can be used in large-scale projects. Given system will deliver the code clone results to end user in an effective and practical manner. “Code Point” has focused on more informative details and how that can be used in the actual working environments.

Table of Contents

List of Figures

List of Tables

Chapter 1

2. Introduction

1.1 Prolegomena

This chapter provides the main objectives of this research and then briefly explains the background and the motivation factors to this research. It also defines the problem statement and purpose of the research.

1.2 Objectives

- Conduct a study on code clone types and currently available code clone detection algorithms and tools.
- Analyze currently available data visualization techniques and tools.
- Analyze code clone data.
- Present Analysed data in a useful manner.
- Provide clone analysis among solution repository versions.
- Evaluate the novel solution.
- Prepare thesis
- Prepare a review paper based on the problem domain and present in a conference.

1.3 Background and Motivation

Quality of a software matters a lot in the field as it's a measure to maintain the standards of a product. Because of that, there are different kinds of applications introduced to maintain the quality of a software. When comes to software maintenance, code clones play a major role affecting to performance of the project. According to previous studies as the code base increases, the results of a clone detection tool can be huge; making it exceedingly difficult to identify the critical clones that need to be removed or fixed. It has been observed by many researchers, [1,8] that 5% to 20% of large software systems are clones, thus validating the reason for a large number of instances of code clones in the reports. Therefore, these software systems contain a significant amount of code clones. With the importance of clones, Detection of them takes more attention in the field of software maintenance.

For example, cloning is often used as an informal reuse strategy [2]. To uncover this practice, quantitative clone results are not sufficient. By relating the structure of the code in the clone interpretation directly, identifying cohering clones is easier. Furthermore, it helps to uncover misconfigurations of clone detection. Having the knowledge that some clone classes are, for example, locally restricted makes it easier to identify generated code which can then be excluded from the analysis. Additionally, during inspections of industrial software systems [3], noticed that in some cases the code base accidentally contained the same files in different, mostly outdated, versions. Knowing at first glance that some system parts are strongly cloned with others helps to exclude these special cases code from the analysis.

As a result of that, a number of clone detection techniques and tools have been proposed and implemented by many researchers [4]. But due to ineffective visualization of data could not be able to get the maximum use out of code clone detection. Currently, most of the systems provide the text-based representations that can be used in a limited scope. When it comes to association with other code quality calculate matrix [5] could not be able to use code clone data in an effective manner. Another drawback of textual representation is when comes to a higher level of data, will take a considerable amount of time to analyze the code. That will be not practical to analyze each and every occurrence.

Separating the code clones of importance from a large number of clones is a major problem because of the following reasons.

- Real-world software projects almost always run short of time and resources. There is a need to meet the project release deadlines and also fix the issues identified by code assessment and clone detection tools before each release. Given these constraints, in case of code bases of large sizes, fixing all clones in such a scenario is not a practical solution.

- Each project has specific needs and requirements and no project will ever need to fix each instance of code clones reported by clone detection tools.

None of the existing tools have attempted to facilitate maintenance of these clones in terms of the order in which the remedial actions could be taken [6]. And not many of these have attempted to provide a unifying framework towards the maintenance of code clones. What is typically required for clone management is a tool that has the ability to prioritize the clones for fixing and if possible, provide hints towards fixing. Also, such a method or criterion should be suitable to all industrial software, irrespective of size, programming language, or domain. Towards this direction, there is a lack of holistic view or framework for addressing code clones and their effective management [7].

1.4 Problem in brief

Lack of tools which represent the code clone results in an effective manner and since then missing out the chances of improving code quality due to gaps between code analysis data.

1.5 Proposed Solution

Provide a solution to detect code clones in the different stages of the project and implement a system to visualize code clone results in a friendly manner.

1.6 Resource Requirements

Hardware

- Laptop / PC

- Intel Core i3 or Above
- 4GB RAM or Above
- Windows 7/8/10

Software

- Visual Studio 2010 (ASP.NET MVC 5, Web API 2, & C#)
- Data visualization tools

1.7 Structure of the thesis.

The documentation of this thesis is outlined in the following way. Chapter 1 provides the background to this research and briefly describes the context of this research. It also defines the problem statement and the purpose of this research. Chapter 2 provides a survey of literature and explains the main phases involved in building a model. Chapter 3 specifies the technology adopted for this research work while the fourth chapter provides the details on the research approaches. The analysis and design part is specified in Chapter 5. The next Chapter explains on the implementation phase. The last Chapter provides a discussion on the proposed methodology.

1.8 Summary

This chapter mainly provides an overview of this research with an explanation of the research background and the problem definition. The next chapter will describe and summarize the existing research work carried out related to this domain.

Chapter 2

2. State of the art in software code clone analysis visualization

2.1 Introduction

Chapter 1 set the background for this thesis by highlighting the research problem, hypothesis, objectives and the solution for code clone result visualization. This chapter reviews the literature findings in code clone detection visualization and defines the research problem for this project. It analyzes various technologies used in code clone detection and discusses the identified possible technologies to address the research problem. The material in this chapter is presented under 5 headings, namely, early development of gestation of code clone visualization, latest development, breakthroughs, future trends and research challenges in code clone visualizations. The findings are summarized at the end of this chapter.

2.2 Background and Fundamentals

2.2.1 Code clone definition

A code clone can be defined as a segment or a portion of a code that is similar or identical to another segment or portion of code [13]

Code clones are classified under 3 categories based on; I) the similarities between two code segments, ii) clone instance position in program and iii) refactoring opportunities with the replicated code. This classification is graphically depicted in Figure 2.1 for more clarity. [5]

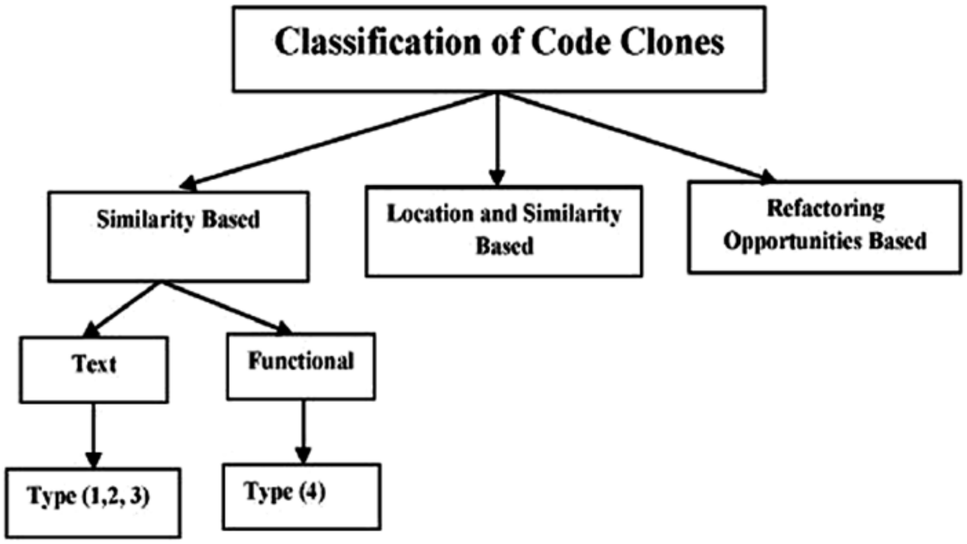

Classification of Code Clones
Similarity Based
Location and Similarity Based
Refactoring Opportunities Based
Text
Functional
Type (1,2, 3)
Type (4)

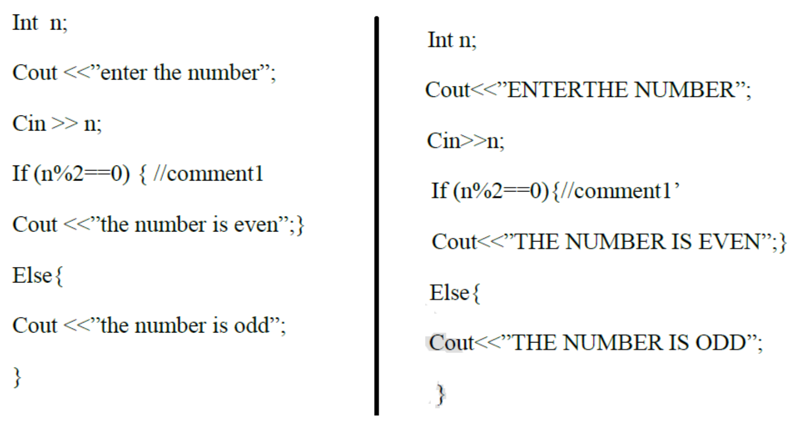
Figure 2.1: Classification of Code Clones [5]

2.2.2 Clone Types

Considering the first of the three classifications mentioned under chapter 1.2.1., two code fragments can be considered as similar based on the program content or based on the functionality. Functional similarity codes are also known as semantic codes. There are four widely recognized clone types out of which the first three types are based on textual similarity and the fourth type the on functional similarity.

Type 1 (Exact Clones)

This type of clones are an exact copy of the original code except for a few variances in whitespace, comments and code layout. These variances are insignificant when considering the language definition. Hence, they are considered as identical clones. Figure ‘A’ shows an example of this type of clones. [3, 5, 7, 13]



```
Int  n;

Cout <<"enter the number";

Cin >> n;

If (n%2==0) { //comment1

Cout <<"the number is even";}

Else{

Cout <<"the number is odd";

}
```

```
Int n;

Cout<<"ENTERTHE NUMBER";

Cin>>n;

 If (n%2==0){//comment1'

 Cout<<"THE NUMBER IS EVEN";}

Else{

Cout<<"THE NUMBER IS ODD";

 }
```

Figure 2.2: Type 1 Code Clone

Type 2 (Parameter Substituted Clones)

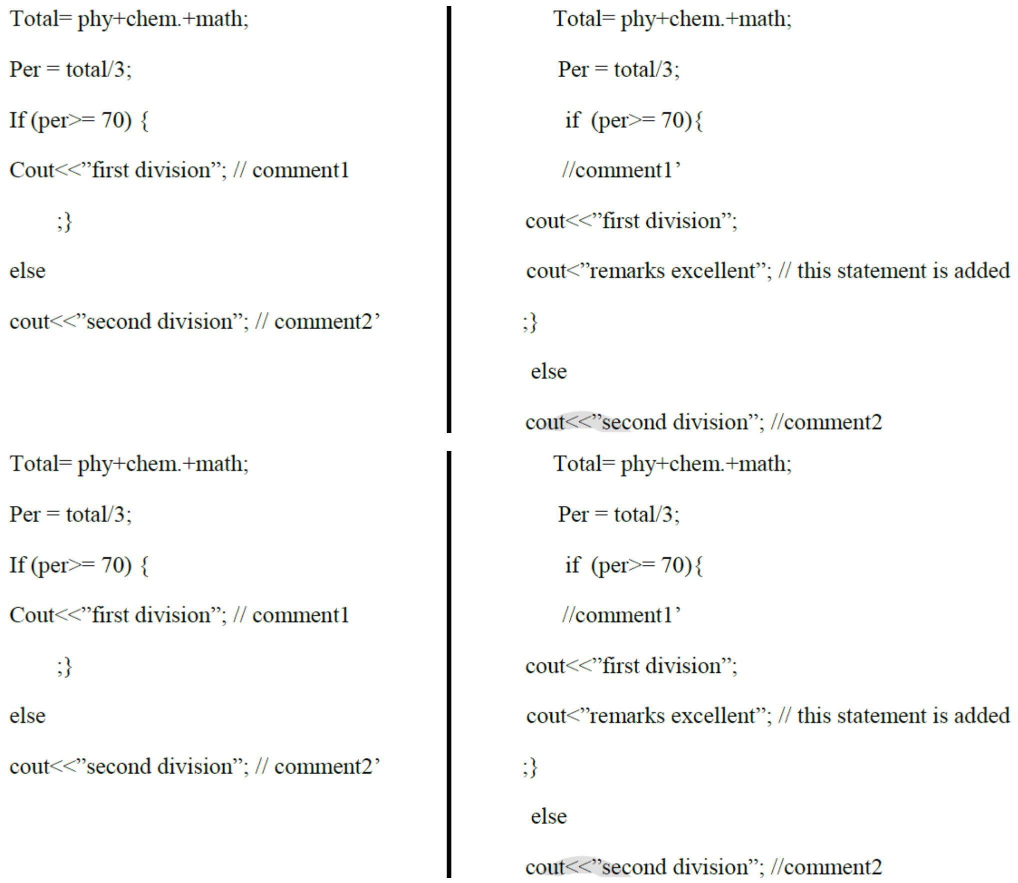
Two identical code fragments can be considered as a Type 2 clone where there are variations in certain parameters such as variables, functional identifiers, literals, and types. These may also have changes in comments and layout. However, the syntactic structures of both codes will be similar. [3, 5, 7, 13]

```
If (salary >= 5000) {                    |      if  (sal >= 5000)
Bonus = 0.5*salary; // comment1          |        {//comment1'
}                                        |        bon=0.5*sal;
else                                     |          }
Bonus = 0.2*salary; // comment2          |        else
                                         |      bon=0.2*sal;
If (salary >= 5000) {                    |      if  (sal >= 5000)
Bonus = 0.5*salary; // comment1          |        {//comment1'
}                                        |        bon=0.5*sal;
else                                     |          }
Bonus = 0.2*salary; // comment2          |        else
                                         |      bon=0.2*sal;
If (salary >= 5000) {                    |      if  (sal >= 5000)
Bonus = 0.5*salary; // comment1          |        {//comment1'
}                                        |        bon=0.5*sal;
else                                     |          }
Bonus = 0.2*salary; // comment2          |        else
                                         |      bon=0.2*sal;
```

Figure 2.3: Type 2 Code Clone

Type 3 (Near-miss Clones)

In this type of clones, the copied code fragment is further modified by adding, removing or changing some statements. Type 3 clones may include changes done in Type 2 clones as well. [3, 5, 7, 13]



```
Total= phy+chem.+math;
Per = total/3;
If (per>= 70) {
Cout<<"first division"; // comment1
        ;}
else
cout<<"second division"; // comment2'
```

```
    Total= phy+chem.+math;
     Per = total/3;
      if  (per>= 70){
      //comment1'
cout<<"first division";
cout<"remarks excellent"; // this statement is added
;}
 else
cout<<"second division"; //comment2
```

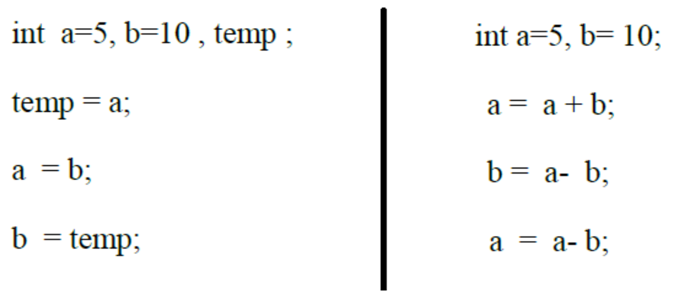
```
Total= phy+chem.+math;
Per = total/3;
If (per>= 70) {
Cout<<"first division"; // comment1
        ;}
else
cout<<"second division"; // comment2'
```

```
    Total= phy+chem.+math;
     Per = total/3;
      if  (per>= 70){
      //comment1'
cout<<"first division";
cout<"remarks excellent"; // this statement is added
;}
 else
cout<<"second division"; //comment2
```

Figure 2.4: Type 3 Code Clone

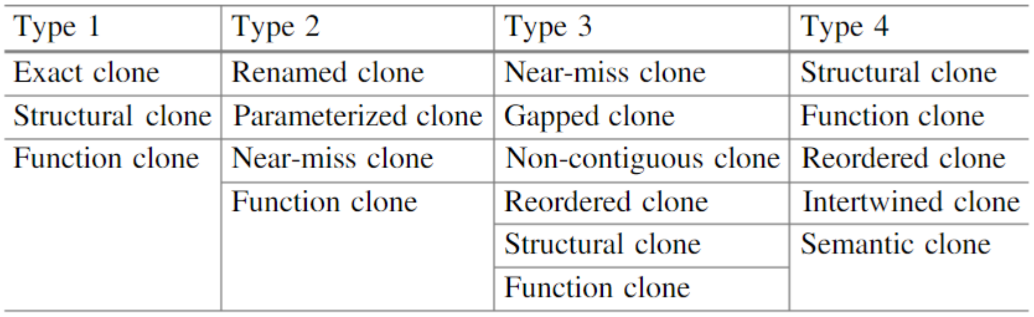
Type 4 (Similar Functionality Clones)

Two or more code segments that may not be copied, but performs the same computation or functionality are considered as Type 4 clones. These code fragments may be developed by different teams, however as they are implementing the same logic, they are semantically similar to each other. [5, 7, 13]



```
int  a=5, b=10 , temp ;        int a=5, b= 10;

temp = a;                       a =  a + b;

a  = b;                         b =  a-  b;

b  = temp;                      a  =  a- b;
```

Figure 2.5: Type 4 Code Clone



Type 1	Type 2	Type 3	Type 4
Exact clone	Renamed clone	Near-miss clone	Structural clone
Structural clone	Parameterized clone	Gapped clone	Function clone
Function clone	Near-miss clone	Non-contiguous clone	Reordered clone
	Function clone	Reordered clone	Intertwined clone
		Structural clone	Semantic clone
		Function clone	

Table 2.1: Summary of clone taxonomy [5]

2.2.3 Clone Relations

It is easier to understand the cloning status of a code base by considering code clone relations in groups opposed to individual clone fragments. Code clone relations can be described as the method in which clones are reported. This identifies clones as clone classes, clone pairs, and clone sets.

Clone classes and clone pairs describe similarities between different code fragments. If there is some similar sequence in the code, it is identified that a clone-relation exists between the code fragments. These terms are explained further below. [7]

- **Code Fragment:** A code fragment can be defined as any sequence of code lines with similarities in its content. It may or may not have comments.

- **Clone Pair:** A pair of code fragments that has similarities between them can be defined as a clone pair.
- **Clone Set:** Clone set consists of all the identical or similar code fragments in a code base.
- **Clone Class:** A clone set where any two code fragments form a clone pair.

2.3 Code Clone Detection

Sometimes engineers use code cloning in order to save time where similar functionalities or methods are used in different sections of the same code base. Although it may seem harmless, while impacting the quality of the code adversely, it also makes code maintenance a very tedious task. For example, if an error was identified in one code segment, the correction has to be applied in the clones as well. Due to such issues, code clone detection has become a popular area of research and many tools, techniques and approaches have been identified and developed to detect code clones.

2.3.1 Code Clone Detection Approaches

This section discusses some of the clone detection approaches that are widely used and reported by researches.

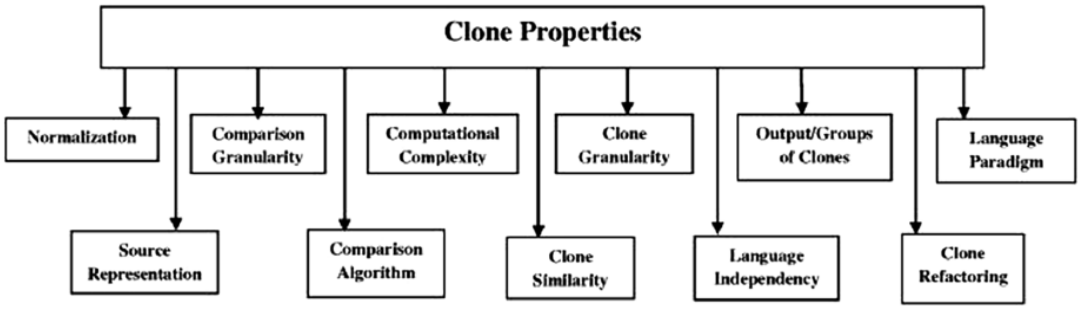


Figure 2.6: Types of Code Clone Properties [5]

Gautam & Saini analyses code clone detection techniques on the basis of code clone properties which are depicted in Figure 2.6. They describe each property as follows [5];

- **Normalization:** applying refinements before the actual comparison (e.g. removing white space and comments)
- **Source Representation:** refers to the result after transformation
- **Comparison Granularity:** granularities are used for particular techniques in the comparison phase
- **Comparison Algorithm:** used to detect dissimilar code clones
- **Computational Complexity:** complexity is based on types of comparison algorithms and transformations
- **Clone Similarity:** different kinds of clones can be identified by various techniques
- **Clone Granularity:** granularity can be fixed for free
- **Language Independency:** verifies the language sustainability of a detection tool
- **Output/Groups of Clones:** shows the types of outputs as clone pairs or clone classes or as both
- **Clone Refactoring:** codes that have been restructured without changing its external behavior
- **Language Paradigm:** the programming language targeted for the particular method of interest

Some of the above properties are discussed further in the section below.

2.3.1.1 Text-based Approach

In the text-based approach, the source fragments are analyzed as a subsequence of text. The two segments are compared textually with each other based on different transformations like white space, newline and removing comments, etc. to locate sequences of same strings [5]. This approach uses little or no transformation or normalization on the source code before the comparison. Sometimes in software clone detection, the source code is directly used [6].

Within this approach, there are several techniques used to achieve results. One of the leading text-based clone detection approaches proposed by Ducasse et al, has used string-based Dynamic Pattern Matching (DPM) to textually compare whole lines that have been normalized to ignore whitespace and comments. [6]

Another approach by Marcus & Maletic uses latent semantic indexing (LSI) for finding code segments that are similar. However, the use of LSI limits the comparison to comments and identifiers [6].

The text-based technique is more efficient compared to other approaches. The drawback is, it can identify Type I, Type II and Type III code clones only. This cannot identify clones which has the same logic but in different coding (Type IV).

2.3.1.2 *Lexical Approaches (Token-based Approach)*

In the token-based technique, it is required to create the sequence of the tokens first, based on the code that is compared. In order to generate tokens, Lexer is required. The purpose of Lexer is to convert the source code into tokens after which various transformations are performed by adding, changing or deleting any tokens. The token sequence is then scanned in order to find the duplicated code or duplicated subsequence of tokens and the code fragment identified as the duplicated code is returned as clones. Kamiya et al. has broken the above-explained process into four steps for clarity. [13]

i. Lexical Analysis: dividing each line of the source file into tokens as per lexical rules and forming a sequence of tokens
ii. Transformation: replacing the identifiers with customized tokens using transformation rules
iii. Match Detection: comparing the token sequence of lines and reporting similar lines as code pairs

iv. Formatting: converting each location of clone pair into line numbers on the original

Efficient token-based techniques are generally based on suffix trees [3]. Originally it was used to search strings efficiently. Brenda Baker [8] initiated using suffix trees for token-based code clone detection. This technique is suitable for very large programs as it is linear in time and space to the program length. In Brenda’s technique, consistently parameterized code clones are abstracted from concrete names and values of parameters but they are not identified from their order.

Token-based or Lexical approaches detect Type I and Type II code clones only.

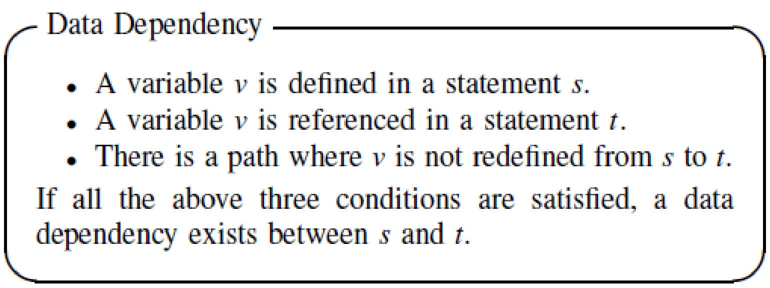
2.3.1.3 Graph-based Approaches

Graph-based approaches use program static analysis to develop a Control Flow Graph (CFG) or a Program Dependence Graph (PDG) for each code segment [14]. PDG is used as an intermediate source to identify subgraphs to detect code clones. This technique uses a program slicing based approach to detect clones because of which it is capable of detecting non-contiguous code clones, unlike other clone detection techniques [10]. However, PDG techniques have a lower performance for detecting contiguous code clones.

Two types of dependencies are considered in PGD;

- Data Dependency

  The logic of data dependency is described in Figure 2.7



Data Dependency

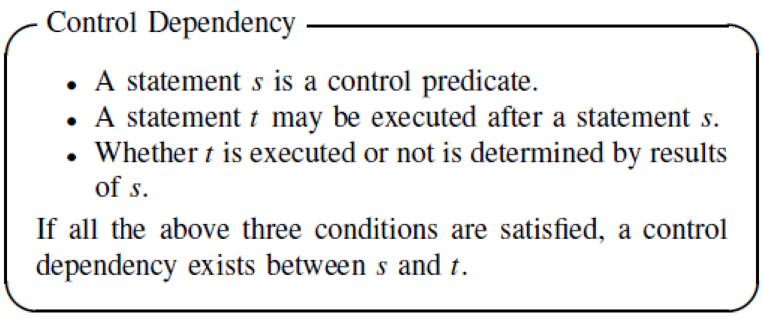
- A variable v is defined in a statement s.
- A variable v is referenced in a statement t.
- There is a path where v is not redefined from s to t.

If all the above three conditions are satisfied, a data dependency exists between s and t.

Figure 2.7: Data Dependency Logic [12]

- Control Dependency

  The logic of control dependency is elaborated in Figure 2.8



Control Dependency

- A statement s is a control predicate.
- A statement t may be executed after a statement s.
- Whether t is executed or not is determined by results of s.

If all the above three conditions are satisfied, a control dependency exists between s and t.

Figure 2.8: Control Dependency Logic [12]

2.3.1.4 Syntax-Tree based Approaches

In syntax-tree based approaches, the source code is transformed into a parse tree or an abstract syntax tree which is then processed using tree matching algorithms or structural metrics to identify code clones.

Researchers report that the results from comparison using syntax-tree based approaches are quite efficient [9]. Further, abstract syntax tree clone analysis is said to be more accurate than a line by line analysis or token-based approach because it builds an abstract syntax tree (AST). The general sequence of steps used by AST to identify code clones is as follows;

i. Hashing each subtree in the AST
ii. Placing each subsequence of the same length in similar buckets (groups) based on the similarity of the hash
iii. Compare the subtrees and subsequences in a bucket against the similarity threshold

This algorithm is used on the basis that code fragments are considered clones if they exceed the similarity threshold at each particular AST node in the hierarchy [1].

Having above mentioned pros, syntax-tree based approaches also has its cons, in that, it is very difficult and complex to create an abstract syntax tree and the scalability is not up to satisfactory level. [9]

2.3.1.5 Metric-based Approaches

Metric-based techniques follow an approach where different types of metrics such as the number of functions, number of lines, etc. are assembled from code fragments and evaluated without comparing the direct source code [5]. Several types of software metrics are used in this technique to find the code clones. In most cases, the source code is converted into the abstract syntax tree (AST) or program data graph (PDG) in order to calculate the metrics. The name, layout, control flow and expression of the functions are used to calculate metrics [7]. One of the popular metric-based techniques uses fingerprinting functions to calculate metrics for syntactic units such as a class, function, method or statement. The resulting values are compared to find clones of the syntactic units [6].

Following are few of the techniques various researchers have built on top of the metric-based approach;

Mayrand et al.: Functions with similar metrics values are identified as code clones using several metrics that are calculated from names, layout, expressions and control flow functions. [6]

Kontogiannis: Two approaches are used to detect clones. [6]

i. Directly compares the metric values as an alternative for similarity at the granularity of begin-end blocks
ii. Compare begin-end blocks on a statement-by-statement basis using dynamic programming (DP) technique

Davey et al.: certain features of code blocks are computed first, and neural networks are then trained to find similar blocks based on the features. This technique can detect clones of Type I, Type II and Type III. [6]

2.3.1.6 Hybrid Approaches (Semantic)

There are numerous hybrid methods that have been presented for code clone detection. The hybrid approach is a collection of several approaches and it can be classified on the basis of preceding techniques [5]. Koschke et al.[23] has presented a hybrid approach for finding Type I and Type II clones which are based on both tree and token methods. In this approach, it generates a suffix tree for serialized AST nodes which is placed in a sequential manner. Then by using the suffix tree-based algorithm, comparisons are performed on the tokens of the AST nodes. Another approach has been used in Microsoft's new Phoenix framework [24]. It can detect exact function clone as well as a parameterized clone with identifier renaming, but not data type changes. Greenan [5] has proposed the sequence matching algorithm for identifying clones in method level which is known as Analogous approach. A dynamic pattern matching as well as characteristics-based hybrid approach is proposed by Balazinska [5] in which method of each body are computed with quality metrics and then evaluated identified clusters by using Patenaude's metric-based approach.

=

2.3.2 Code Clone Detection Tools

Sometimes engineers use code cloning in order to save time where similar functionalities or methods are used in different sections of the same code base. Although it may seem harmless, while impacting the quality of the code adversely, it also makes code maintenance a very tedious task. For example, if an error was identified in one code segment, the correction has to be applied in the clones as well. Due to such issues, code clone detection has become a popular area of research and many tools, techniques[25] and approaches have been identified and developed to detect code clones.

There are many clone detection tools that are discussed in the literature. Described below are some of them.

- SolidSDD

  This tool performs both extraction and visual analysis of code clones. It simplifies the clone extraction process and visual analysis of the extracted results, which in turn makes it possible to complete the whole process in less time. This can be used with code bases written in C, C++, Java, and C#. SolidSDD tool has the capability to identify subsystems that have high clone percentages, types of clones contained in a given file, files that are affected by a particular clone, etc. [15]. It uses a token-based approach to detect the clones.

  SolidSDD tool can be configured to detect clones with various set parameters such as clone length in statements, whether identifier renaming is allowed, gap size of modifications made to a clone (such as inserting or deleting code fragments), filtering of whitespace and comments, etc. Once the clones are extracted, a compound duplication graph is created and stored in an SQLite database. The nodes of the graph represent cloned code fragments and the edges represent clone relations. The structure is added by default from the code directory data. If not, a syntax-based code hierarchy can be used to obtain the structure. Metrics are computed on nodes (code location) and edges (cloned code percentage, number of distinct clones, and if a clone uses identifier renaming).

- Duplo

  This is an open source tool. In order to detect clones, Duplo uses hashed string matching of lines of code. This tool has the capability to compare two or more files to find blocks of matched code. A block can be determined using two parameters. These are the minimum number of characters in a line and minimum consecutive matched lines in a block. In the default settings of the tool, parameters are set as 3 characters and 4 lines respectively. Before the tool initiates the clone detection process, it is programmed to remove whitespace and comments. [17]

- Simian

  Simian, short for similarity analyzer is a text-based clone detection tool. Simian can run on almost any hardware and operating system. It runs natively in .NET 1.1 or higher platforms and on any Java 5 or higher virtual machines. Apart from these, it can identify and extract clones from codes written in multiple programming languages such as Java, C#, C, C++, COBOL, Ruby, JSP, ASP, HTML, XML, Visual Basic, Groovy source code and text files. Simian can also read ini files and deployment descriptors.

  This tool can be used during development as part of the build process and as a guide when refactoring. It helps to raise the quality of the software program [18]. Simian can process and complete clone detection utilizing less memory and less time due to its line-based technique. It further allows the user to configure settings for clone detection. [19]

- CCFinderX

  Two main versions have been released of this tool and both use a token-based approach in detecting code clones. The version that was released first was known as ‘CCFinder’. Before comparing code fragments, this tool replaces user-defined identifiers like variable names or function names with special tokens. This enables the CCFinder to identify Type II clones where the variables have been changed in the duplicate code. Researchers report that this tool can identify all code clones from a code base with millions of lines of code within an hour. Further, like ‘Simian’ tool, CCFinder also can handle multiple programming languages such as C/C++, Java, and COBOL. [19]

The next version released, ‘CCFinderX’ is considered as a major version up from CCFinder as it can handle more programming languages (apart from ones mentioned earlier, Visual Basic and C#) and it also effectively uses resources of multi-core CPUs using multi-threading for faster code clone detection. Further, the detection algorithm of CCFinderX was changed from bucket sort suffix tree which has enabled more coverage (of types of clones) and more accurate results. [19]

- Continuous Quality Assessment Engine

  Also known as ‘Continuous Quality Assessment Toolkit (ConQAT)’ is a toolkit used for software quality analysis and control. ConQAT is available as an open-source tool and it supports the development and execution of software quality analysis rapidly. ConQAT is based on a pipes-and-filters architecture and offers a dataflow language for the specification and parameterization of the clone detection pipeline.

  ConQAT uses its own engine and offers treemap, clone visualizer and family visualizer views for analysis. Clone Visualizer and family visualizer views can group related clone files and visualize. However, they cannot show cloning dependency among subsystems. [21]

2.3.3 Overview of Clone Detection Tools

Approach	**Tool**	**Clone Type**				**Support Languages**	**License**	**Is API/Engine Available**	**Reference**
		1	2	3	4				

Text Based	Duplo	**X**	**X**			C, C++, Java, C#, VB.Net	Free	No	[17]
	Simian (Similarity Analyzer)					Java, C#, C, C++, COBOL, Ruby, JSP, ASP, HTML, XML, Visual Basic, Groovy source code, text files	Free	No	[18]
Token Based	SolidSDD	**X**	**X**			C, C++, Java, C#	Paid	No	[15]
	CCFinderX	**X**	**X**	**X**		Java, C, C++, COBOL, Visual Basic, C#			[19],[20]
Graph-Based	Duplex	**X**	**X**	**X**					[12]
	Komondoor	**X**	**X**	**X**					[10]
Syntax-Tree Based	CloneDR	**X**	**X**						[9]
	Ccdiml					C, C++			[5]
Metric Based	CLAN	**X**	**X**	**X**			N/A	N/A	[10]
	Davey et al	**X**	**X**	**X**			N/A	N/A	
Hybrid	ConQAT	**X**	**X**	**X**		Java, C/C++, C#, ABAP, Python, PL/SQL	Free	Yes	[21]
	Atomiq	**X**	**X**	**X**		C#, Java	Paid	No	[21]

Table 2.2: Summary of CCD Tools

2.4 Code Clone Visualization

2.4.1 Visualization Techniques

Scatter Plot

This is a two-dimensional data visualization technique that uses the same logic as line graphs for plotting data points, the difference being, scatter plot uses dots to represent values of two different variables. It shows the relationship between the two sets of data.

TreeMap

a is a visualization technique used to display hierarchical data. It uses a series of clustered rectangles to represent branches of a tree diagram and it depicts the values or quantities for each category in the area size of the rectangle. The division and ordering of rectangles into sub-rectangles is based on the tiling algorithm used.

Pie Charts

A pie chart is a circular graph of which the full circle represents 100% of the whole and any slices represent portions of the whole. The size of each slice is relative to the value of the category is represents in the group as a whole.

2.4.2 Visualization Tools

- SolidSDD Visualization

  SolidSDD can be considered as a combination of many scalable visualization techniques such as hierarchical edge bundles, table lenses, annotated text views, and linked views. This tool uses many information visualization techniques as well for analysis. Some of these techniques are as follows [15];

- ***Annotated Text:*** shows clones in their file context and allows navigating between all pairs of a clone
- ***Bundled Graphs:*** show clones vs system structure
- ***Table Lenses:*** show clone and file metrics
- ***Linked Views:*** enables navigating between text, clones, and system structure

- ClonEvol

  ClonEvol is a visual analysis tool which can be used to obtain insight into the state and the evolution of a C/C++/Java source code base on project, file and scope level. The information extracted from the software versioning system is combined with the contents of files that change between versions in order to do a proper analysis. ClonEvol uses Subversion (SVN) in order to obtain the code history, Doxygen to analyze the data and Simian to detect code duplicates or code clones.

  The information consolidated through acquisition, analysis, and clone detection is presented in an interactive manner to the user. The focus of ClonEvol is on scalability (in time and space) with regards to data acquisition, data processing, and visualization. It also focuses on ease of use. ClonEvol uses a mirrored radial tree to show the file and scope structures and hierarchically bundled edges to show clone relations. [16]

2.4.3 Overview of Visualizations

The table below summarizes available visualization or clone presentation techniques, details of coverage and clone relation details.

Visualization Technique	**Entity**	**Clone Relation**

Scatter Plot	File, Subsystem	Clone Pair
Hasse Diagram	File	Clone Class
Metric Graph	Code Segment	Clone Class
Hyper-Linked Web Page	Code Segment	Clone Class
Dependency Graph	Subsystem	Clone Pair
Duplication Web	File	Clone Pair
Duplication Aggregation Tree Map	File, Subsystem	Clone Class
System Model View	File, Subsystem	Clone Pair
Clone Class Family Enumeration	File	Clone Class
Clone Coupling and Cohesion	Subsystem	Super Clone
Clone System Hierarchy Graph	File, Subsystem	Clone Pair, Clone Class
Clone Visualizer View	Code Segment, File	Clone Class
Stacked Bar Chart	Code Segment, File	Clone Class
Line Graph	Code Segment, File	
Clone Cluster View	Code Segment	Clone Class

Table 2.3: Categorization of clone presentation techniques [22]

2.5 Summary

In this chapter, we have provided further motivation for this thesis along with background material and related work. After defining the terminology of clones, we explained various reasons behind cloning. Next, we briefly described various effects of clones that can negatively impact development activities. We reviewed techniques for clone detection, analysis, and visualization. We then explained the terminology of clone evolution followed by a brief overview of clone evolution analysis. Finally, we reviewed studies of software and clone evolution visualization.

Chapter 3

3. Technology

3.1 Introduction

The previous chapter contains a critical analysis of the existing code clone detection techniques, tools, and visualizations. It also includes the positive and negative aspects of each tool and item.

This chapter focuses mainly on the technological aspects of the research. There are several components of the project that uses various technologies to achieve the expected results out of it.

3.2 Frontend Technologies

The frontend technologies considered for the implementation of the proposed solution are given below;

- ASP.NET MVC

  Currently, this is the most famous Microsoft web development platform which follows the model-view-controller pattern. It has rich features for handling the presentation layer and communicates with the backend. It supports various UI frameworks like bootstrap and helps to increase the usability of the tool.

- jQuery and AJAX

  AJAX is a web technique which uses to communicate between the client side and server side. It has the ability to communicate within UI and controller without affecting the behavior of the frontend. We can make asynchronous calls to retrieve and send data

3.3 Backend Technologies

- XML

  Extensible Markup Language is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. This will be useful when it comes to dynamic mapping which we can do very effectively.

- SQL Server

  Microsoft SQL Server is a relational database management system developed by Microsoft. As a database server, it serves the primary function of storing and retrieving data as requested by other software applications that run either on the same computer or on another computer across a network. This DBMS works well with ASP .NET solutions.

3.4 Web Services

- WebAPI

  This is a type of web service which is an application programming interface for a web server or a web browser. Web API helps for the decoupling of the program which will give more flexibility to grow and expose internal data to the third parties.

3.5 Code Clone Detection Technologies

As presented in the previous chapter there are several types of methods to identify code clones such as Text-based, Token-based, Graph-based, Syntax-Tree based approaches, etc. Each method has own advantage and disadvantages which provides motivation for the research. When deciding the code clone detection method for the project it should consider main factors like accuracy, efficiency, usability, etc.

3.6 Summary

This chapter briefs the front end, back end and web services that can be used to implement the proposed solution. It also discusses the code clone detection technologies that the author will be able to refer during implementation.

Chapter 4

4. Approach

4.1 Introduction

Chapter 3 described the technology to be used to solve the research problem. This chapter presents our approach to addressing the methodology of code clone results visualization. For this purpose, we describe our hypothesis, inputs, outputs, processes, users, and features in our approach.

4.2 Hypothesis

Our hypothesis is that properly structured code clone visualization can lead to a quality software output. For the stakeholders of software development projects, will be able to do a proper evaluation of the quality of the product. By providing user-friendly visualization and summarized data will help to save a considerable amount of time when analyzing the quality of source code. And also it helps for the developers as well as managers when making decisions of the quality improvements of the work

4.3 Input

Configure the source code repository to identify code duplications. User will be able to configure it as per own preference. Since the application is facilitating to compare code clones within the releases need to configure the repository (TFS or GIT) as well. The application will provide user-friendly interfaces for the inputs.

4.4 Output

There will be two major output elements. One will be text-based representation which will be used by developers to identify code clones. Other Application will be a web portal which includes dashboard will mainly be focused on analytics. There will be graphical representations that will be created based on user preferences. Ex: - Version wise type 2 code clone counts

4.5 Process

Once user request for a clone detection will consider the relevant change set the existing codes. By using a code clone detection algorithm will extract the required data. That will cover all types of code clones. After the data extraction, they will be sent to the data visualization module that will generate relevant graphs. Data visualization methodology will be decided based on the literature review findings.

Code clone analysis data visualization will be presented to the user based on the user type which will be in a useful manner. As an example, a developer needs different types of data which will help him to improve the code quality. He would prefer more technical oriented results from the proposed solution. Aspects of managerial levels are different from the developers. They would prefer more strategical representations. With this proposed solution will be able to cater more personalized results to the end user.

All the results will be saved to the DB which will be able to revise the history based on a person and that will help to provide more detailed reports to the relevant personnel. And same time will be able to help to identify how things have improved in a timely manner.

4.6 Features

- Configure repositories and allow compression between commit versions.
- Code analysis between given project files.
- Code comparison results (Text and Visual based).

- Friendly GUI to resolve code clones.
- YouTube and Forum support on clone types and solutions.

4.7 Summary

This chapter describes the high-level solution approach of this research work. It also lists down the list of features for the proposed solution. The next chapter will explain this with more details.

Chapter 5

5 Analysis and Design

5.1 Introduction

Chapter 4 described the approach of the research problem. According to that research will proceed and this chapter presents the design of the project. The chapter will contain several design diagrams like high-level diagrams, class diagrams and etc. These will be used for the prototype of the project

5.2 High-level design

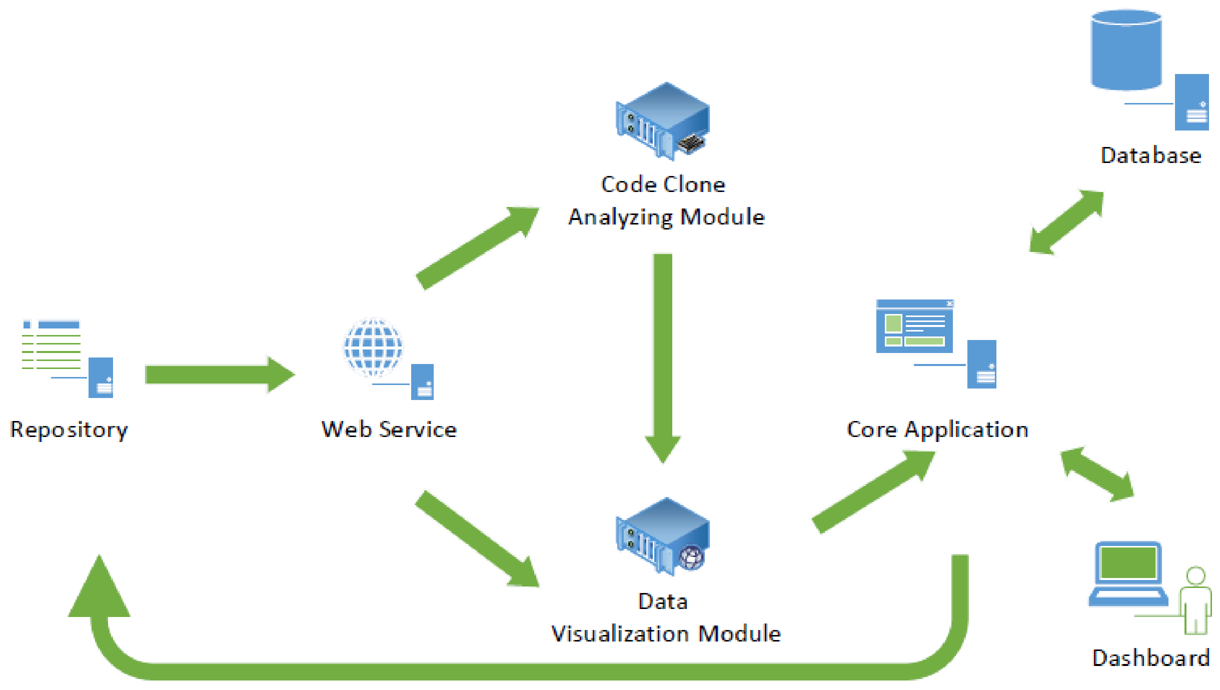


Figure 5.1: High-Level Design Diagram

5.3 Architecture

For the “Code Point” intended architecture to use is service based 3 tire architecture. User will be able to access the ‘Result Analyzer’ setup module and data will be transmitted to the business layer using web service. Advantage of having web service in the middle will provide a facility to expose data different platforms and devices.XML configurations will be used to process data files. From the business layer, it will access the database through the data access layer and returns structured processed data to the dashboard.

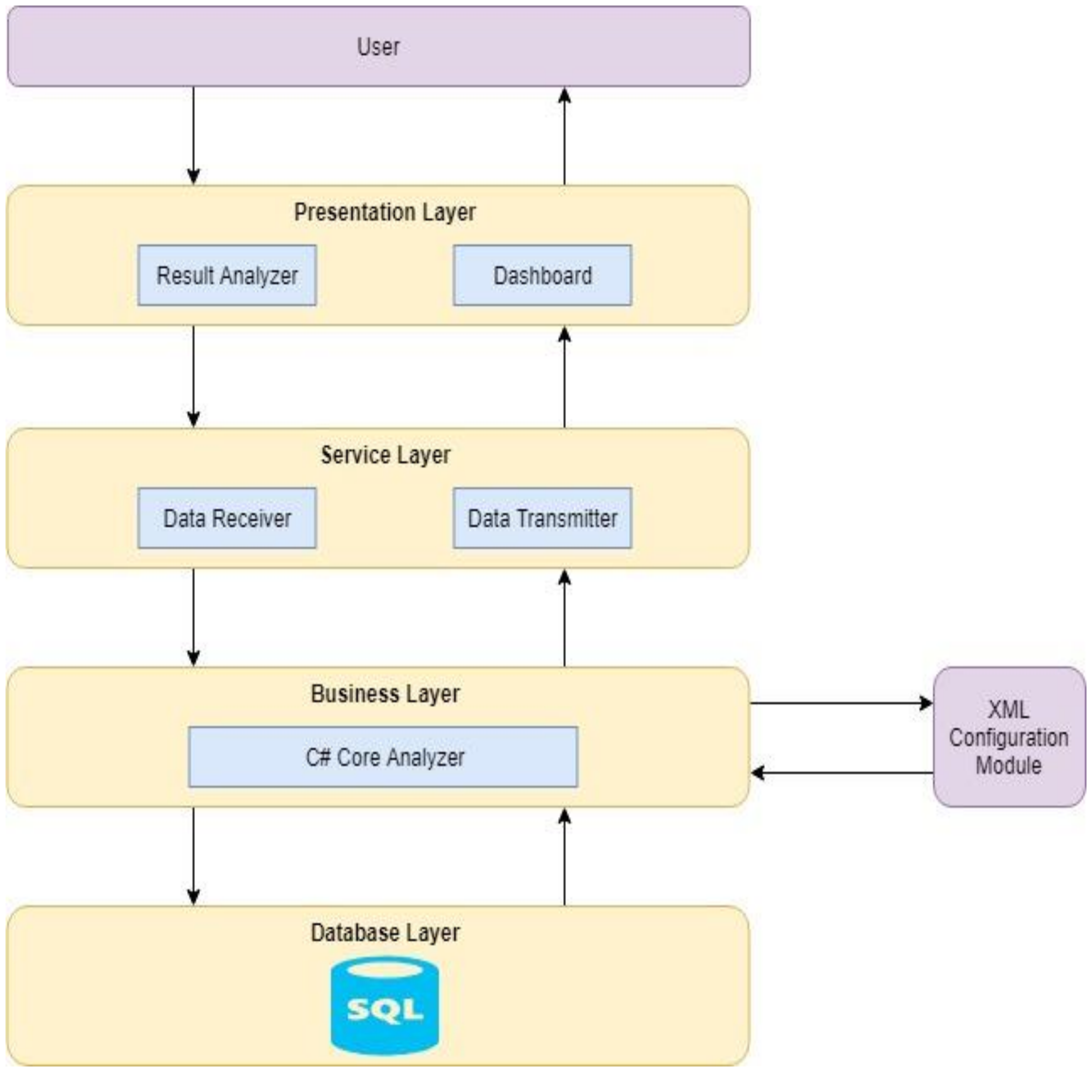


Figure 5.2: Code Point - Solution Architecture

5.4 System Process

Use case diagrams are used to demonstrate a user's interaction with the system. Below figure 5.3 represents the use case diagram of the proposed solution.

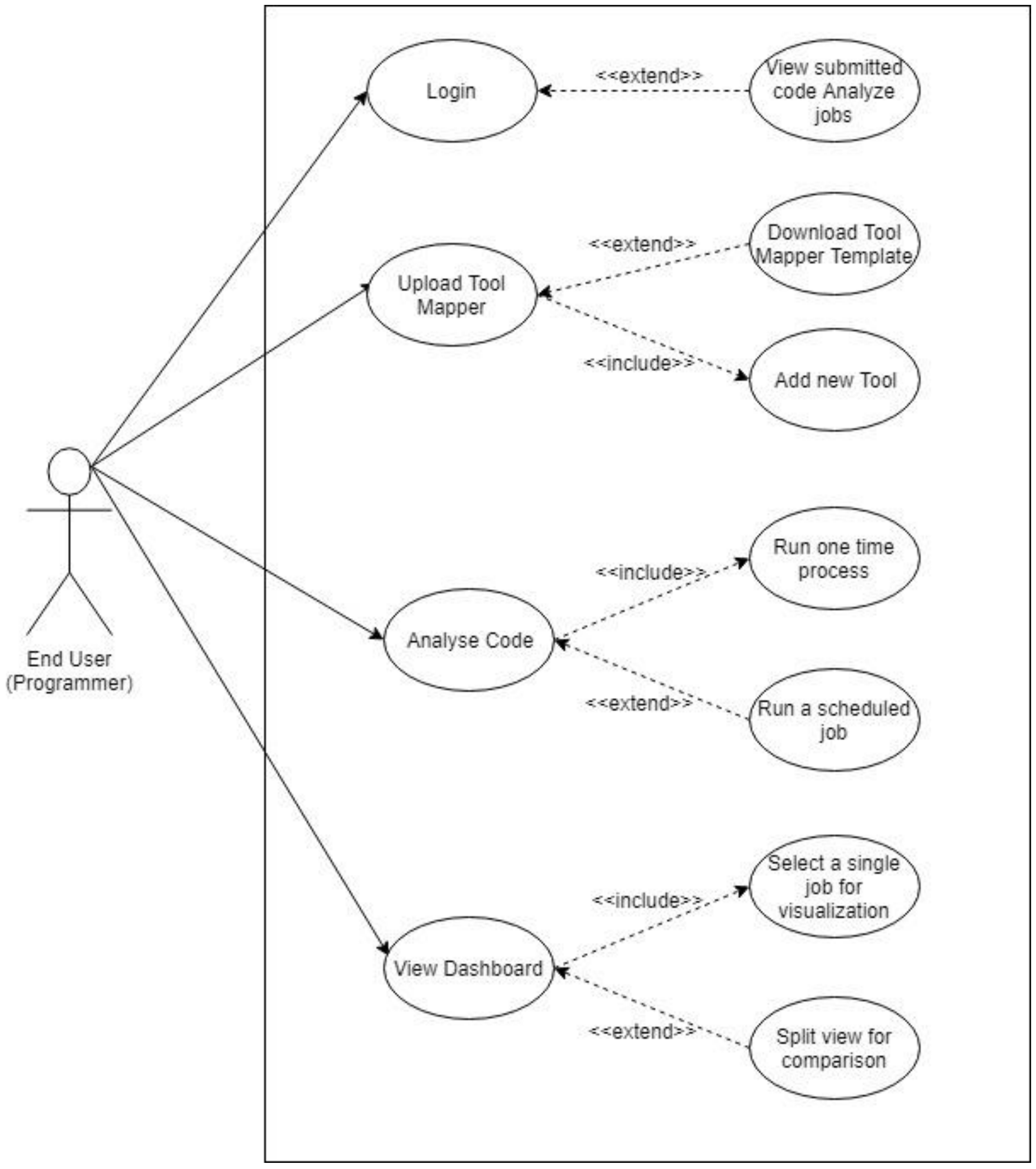


Figure 5.3: Code Point - Use Case Diagram

5.5 Summary

The system architecture, processes, and workflows were designed in this chapter using various designing approaches such as High-Level Design diagrams, Architecture model, and Use case diagrams. These design components will serve as a foundation for the implementation stage.

Chapter 6

6 Implementation

6.1 Introduction

The design techniques for each of the components of the proposed system are comprehensively discussed in the previous chapter.

This chapter examines the techniques and methodologies used and the steps taken in implementing the design of the proposed system. This project has several implementation segments as like user interfaces, code clone identification, processing and visualization

6.2 "Code Point" Solution

When considering the structure of the project, implementation is done by using mainly .Net based technologies. For the frontend has used ASP.NET MVC which provides rich features for the development. As client-side scripting used jQuery and for data exchange have used AJAX.

All the data pushes and retrievals are happening through web API which will provide more flexibility to the program. Since the presentation layer and the business layer is connected via Web API it has become more independent. As the web service mechanism have used WEB API 2.

As figure 6.1 given below have followed 3 tire architecture. Once the user makes a request trough MVC controller/Web API controller it calls the relevant method in the business layer which is responsible for the process of the request. After required processing business layer will call the data access layer which will handle the database transactions.

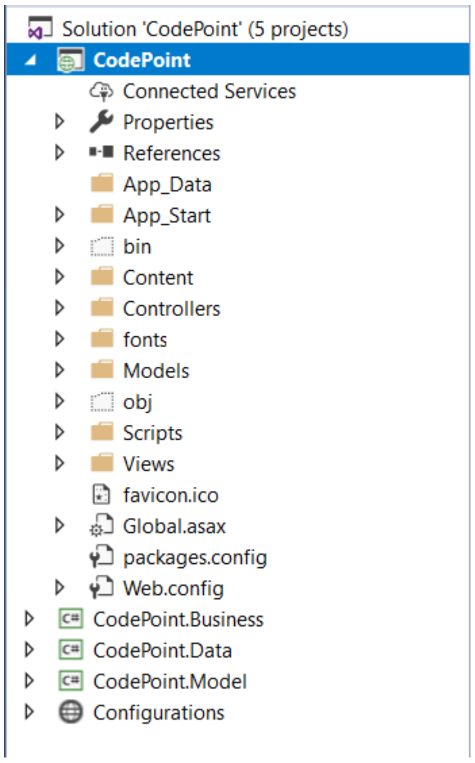

```
Solution 'CodePoint' (5 projects)
  CodePoint
    Connected Services
    Properties
    References
    App_Data
    App_Start
    bin
    Content
    Controllers
    fonts
    Models
    obj
    Scripts
    Views
    favicon.ico
    Global.asax
    packages.config
    Web.config
  CodePoint.Business
  CodePoint.Data
  CodePoint.Model
  Configurations
```

Figure 6.1: Solution Structure

There are 3 main UI components of the code point solution. There are 2 main analysis methods. One method is analyzing the file or projects from scratch. And the other way is analyzing the result files. User will be able to pick as prefers. The final feature is the dashboard which visualizes the extracted and processed code clone related data

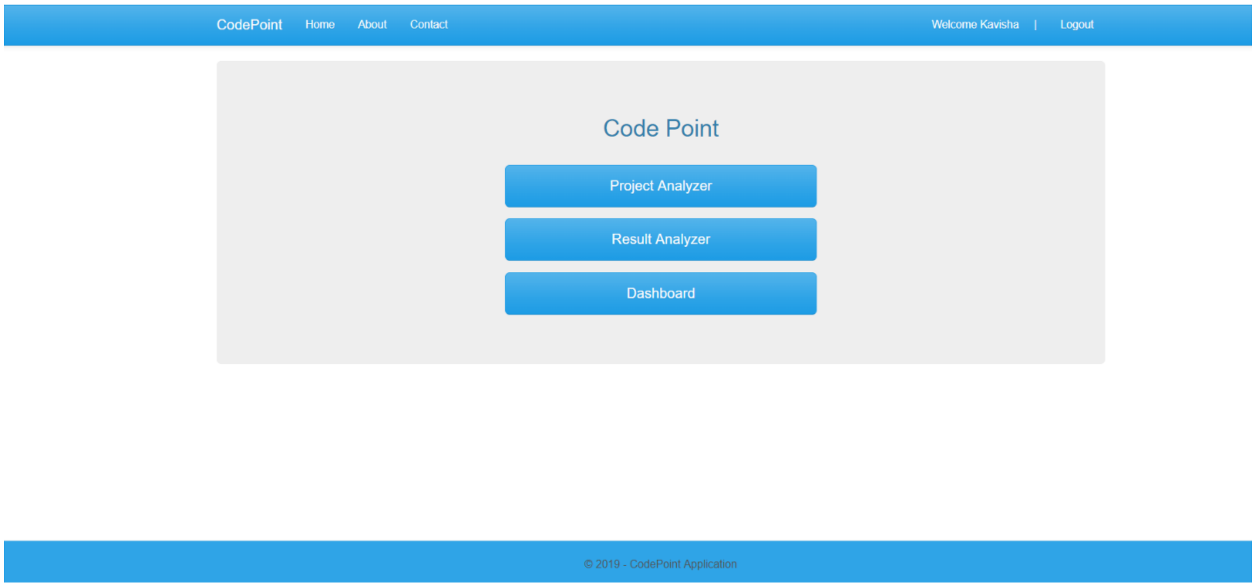


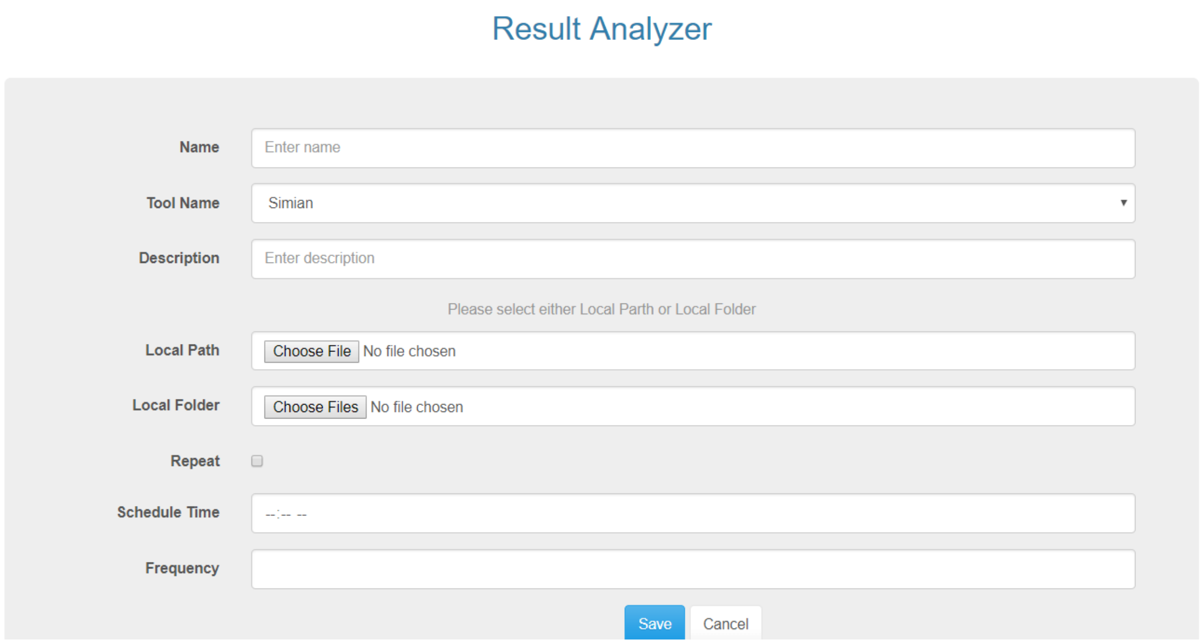
Figure 6.2: Home Page

6.3 Project Analyzer

This application segment will be used to analyze the existing files or projects. There are two ways to do the analysis of the source cord. One way is selecting a project source code/file from a physical path and the other way is setting up a repository. The application will proceed based on the user input for each type and code clone detection engine will process the files. Extracted data will save in the MSSQL database.

6.4 Result Analyzer

Result analyzer is the core component of the research and the main target of this is to integrate different types of code clone detection results extractions with the dashboard. User will be able to save templates of the analysis that he conducts. Basically, the user will have to insert XML file or XML folder path which includes the results of code clone detections. And also user will be able to schedule result file reader which will trigger in a given time period


Result Analyzer

Name	Enter name
Tool Name	Simian
Description	Enter description
	Please select either Local Parth or Local Folder
Local Path	Choose File No file chosen
Local Folder	Choose Files No file chosen
Repeat	☐
Schedule Time	--:-- --
Frequency	

Figure 6.3: Result Analyzing Page

6.5 Dashboard

Dashboard is the visualization component of the project. Dashboard contains several types of graph items which indicates various types of data. When comes to code clone analysis visualization takes a major part. Dashboards should design in a way that users should be able to able to get as much as information. And also it helps to make decisions based on the categories visualizations.

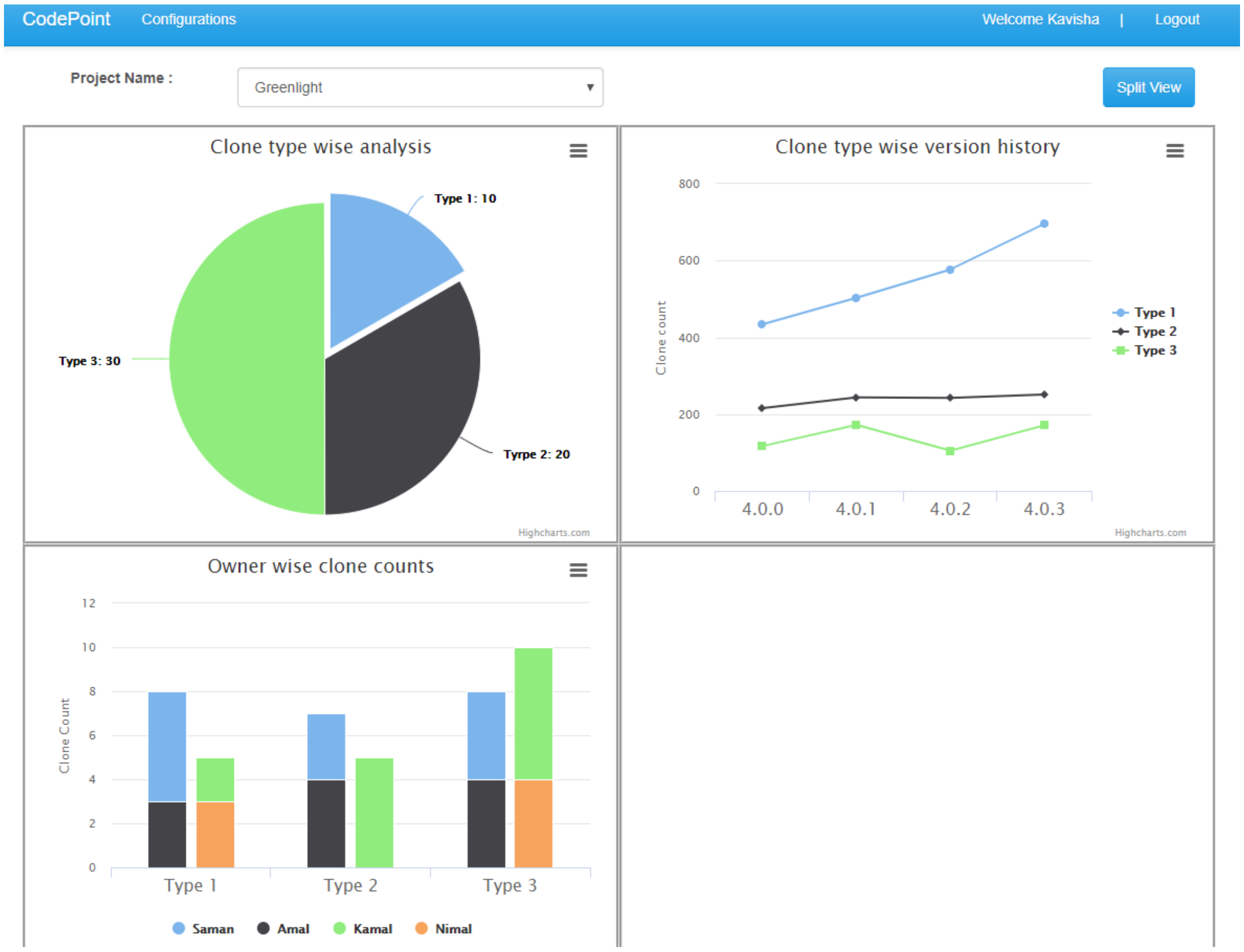
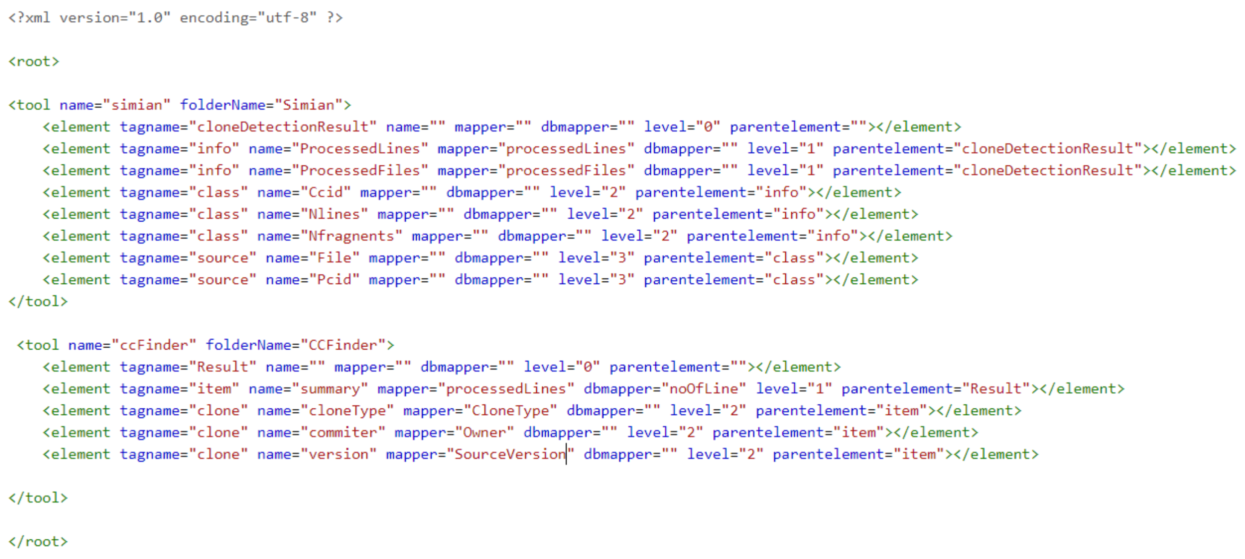


Figure 6.4: Dashboard

6.6 Result file processing module

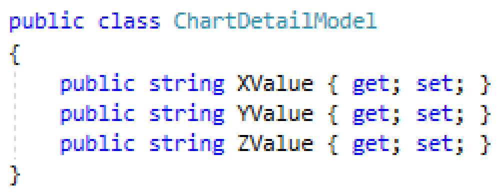
For dynamic result file processing, we need to have a mapping with the application structure. To achieve that goal have used following common XML which contains the mapping of result file and application DB structure.



```xml
<?xml version="1.0" encoding="utf-8" ?>

<root>

<tool name="simian" folderName="Simian">
    <element tagname="cloneDetectionResult" name="" mapper="" dbmapper="" level="0" parentelement=""></element>
    <element tagname="info" name="ProcessedLines" mapper="processedLines" dbmapper="" level="1" parentelement="cloneDetectionResult"></element>
    <element tagname="info" name="ProcessedFiles" mapper="processedFiles" dbmapper="" level="1" parentelement="cloneDetectionResult"></element>
    <element tagname="class" name="Ccid" mapper="" dbmapper="" level="2" parentelement="info"></element>
    <element tagname="class" name="Nlines" mapper="" dbmapper="" level="2" parentelement="info"></element>
    <element tagname="class" name="Nfragnents" mapper="" dbmapper="" level="2" parentelement="info"></element>
    <element tagname="source" name="File" mapper="" dbmapper="" level="3" parentelement="class"></element>
    <element tagname="source" name="Pcid" mapper="" dbmapper="" level="3" parentelement="class"></element>
</tool>

 <tool name="ccFinder" folderName="CCFinder">
    <element tagname="Result" name="" mapper="" dbmapper="" level="0" parentelement=""></element>
    <element tagname="item" name="summary" mapper="processedLines" dbmapper="noOfLine" level="1" parentelement="Result"></element>
    <element tagname="clone" name="cloneType" mapper="CloneType" dbmapper="" level="2" parentelement="item"></element>
    <element tagname="clone" name="commiter" mapper="Owner" dbmapper="" level="2" parentelement="item"></element>
    <element tagname="clone" name="version" mapper="SourceVersion" dbmapper="" level="2" parentelement="item"></element>

</tool>

</root>
```

Figure 6.5: XML Mapper

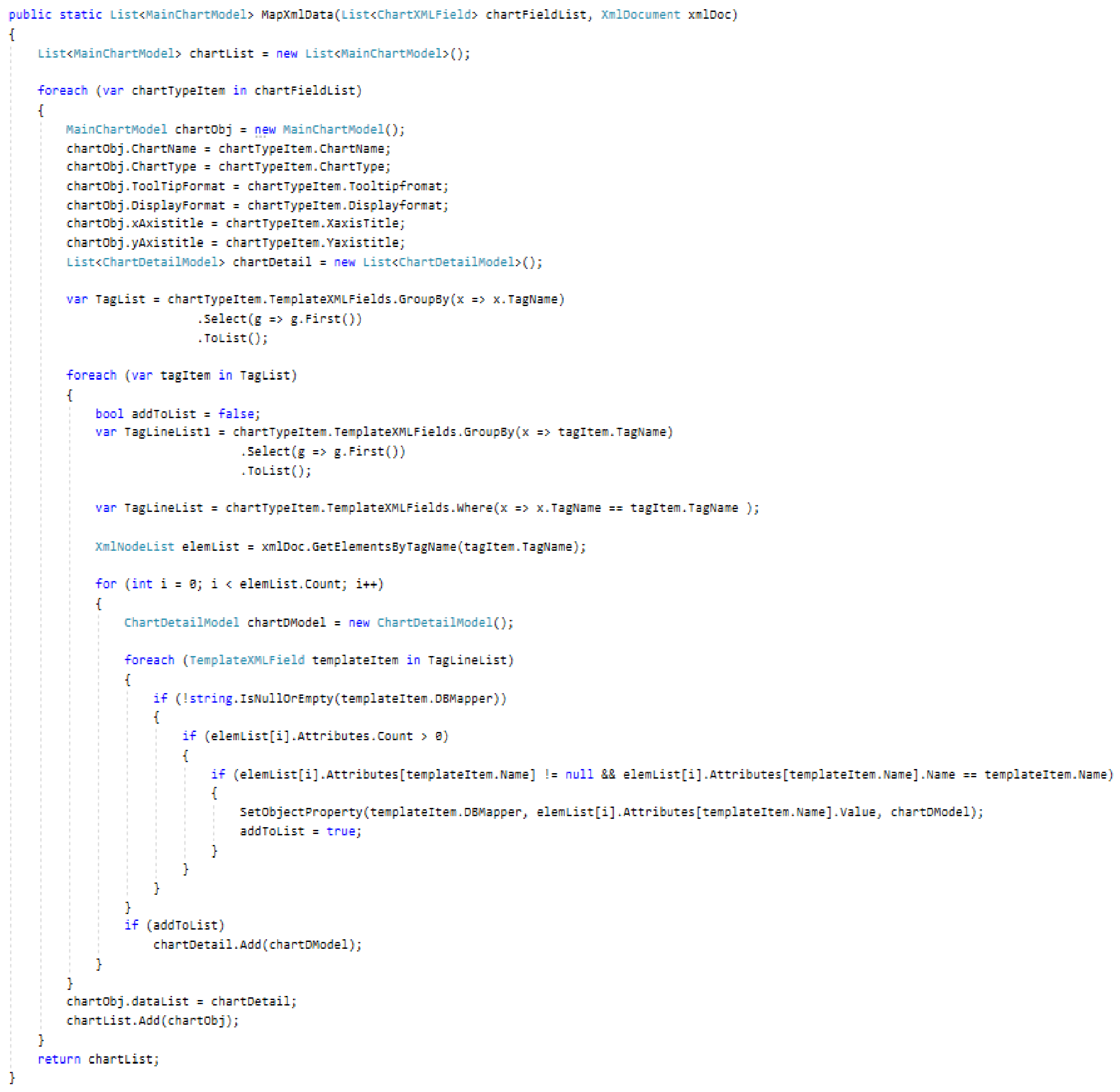
In the proposed solution has given a common model to initialize values from the results file. User can define the values a need to be taken from the results files and based on that model will be filled with the respective data. The current structure is capable of handling up to 3 factors. (Figure 6.6)



```csharp
public class ChartDetailModel
{
    public string XValue { get; set; }
    public string YValue { get; set; }
    public string ZValue { get; set; }
}
```

Figure 6.6: Detail Model Structure

Based on the configuration file application will extract data from the result XML and process to map with the current backend structure. Following figure 6.7 indicates the core logic has used to extract details from the configured XML structure.



```
public static List<MainChartModel> MapXmlData(List<ChartXMLField> chartFieldList, XmlDocument xmlDoc)
{
    List<MainChartModel> chartList = new List<MainChartModel>();

    foreach (var chartTypeItem in chartFieldList)
    {
        MainChartModel chartObj = new MainChartModel();
        chartObj.ChartName = chartTypeItem.ChartName;
        chartObj.ChartType = chartTypeItem.ChartType;
        chartObj.ToolTipFormat = chartTypeItem.Tooltipfromat;
        chartObj.DisplayFormat = chartTypeItem.Displayformat;
        chartObj.xAxistitle = chartTypeItem.XaxisTitle;
        chartObj.yAxistitle = chartTypeItem.Yaxistitle;
        List<ChartDetailModel> chartDetail = new List<ChartDetailModel>();

        var TagList = chartTypeItem.TemplateXMLFields.GroupBy(x => x.TagName)
                           .Select(g => g.First())
                           .ToList();

        foreach (var tagItem in TagList)
        {
            bool addToList = false;
            var TagLineList1 = chartTypeItem.TemplateXMLFields.GroupBy(x => tagItem.TagName)
                                   .Select(g => g.First())
                                   .ToList();

            var TagLineList = chartTypeItem.TemplateXMLFields.Where(x => x.TagName == tagItem.TagName );

            XmlNodeList elemList = xmlDoc.GetElementsByTagName(tagItem.TagName);

            for (int i = 0; i < elemList.Count; i++)
            {
                ChartDetailModel chartDModel = new ChartDetailModel();

                foreach (TemplateXMLField templateItem in TagLineList)
                {
                    if (!string.IsNullOrEmpty(templateItem.DBMapper))
                    {
                        if (elemList[i].Attributes.Count > 0)
                        {
                            if (elemList[i].Attributes[templateItem.Name] != null && elemList[i].Attributes[templateItem.Name].Name == templateItem.Name)
                            {
                                SetObjectProperty(templateItem.DBMapper, elemList[i].Attributes[templateItem.Name].Value, chartDModel);
                                addToList = true;
                            }
                        }
                    }
                }
                if (addToList)
                    chartDetail.Add(chartDModel);
            }
        }
        chartObj.dataList = chartDetail;
        chartList.Add(chartObj);
    }
    return chartList;
}
```

Figure 6.7: Extraction core logic

6.7 Code Clone Detection

Once user request for a code clone review web service will consider the relevant change set the existing codes. By using a code clone detection algorithm will extract the required data. That will cover all the types of code clones.

Among Text-based Approaches, Lexical Approaches (Token-based Approach), Graph-based Approaches, Syntax-Tree based Approaches will be using the required algorithm based on the scenario.

All the results will be saved to the DB which will be able to revise the history based on a particular person and that will help to provide more detailed reports to the relevant personnel. And same time will be able to help to identify how things have improved in a timely manner.

6.8 Summary

This chapter includes the implementation approaches and techniques that were used to implement the proposed solution. Next chapter details the test scenarios and the results.

Chapter 7

7 Testing And Evaluation

7.1 Introduction

The testing phase, though sometimes not given much attention, is one of the most important phases in the software development life cycle. It is through testing that we can ensure the quality of the final outcome of the application. Therefore, this chapter focuses on developer testing and evaluation of the proposed solution against the objectives that we had set initially.

7.2 Testing

Testing was done using different approaches to make sure the quality of the end product is high. Following are the different testing approaches used.

7.2.1 Testing Methods

According to the IEEE (1990, 74) testing is defined as,” The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component”. There are several types of testing strategies available to use. In this solution, mainly two types of testing strategies have been used in order to check the quality and to ensure that project objective and the coding standards are met. Following testing strategies have been used in the application:

Black box testing:- According to Mohan K.K, Verma A.K and Srividya A.(2010), black box testing refers to the technique of testing a system with no knowledge of the internals of the system. Black Box testers do not have access to the source code and are oblivious of the system architecture.

White box testing:- White Box Testing refers to the technique of testing a system with knowledge of the internals of the system. White Box testers have access to the source code and are aware of the system architecture. A White Box tester typically analyses source code, derives test cases from knowledge about the source code, and finally targets specific code paths to achieve a certain level of code coverage.

7.2.2 Testing Levels

Unit Testing - According to the Burback R (1998), unit testing is used to test the individual units of the source code, sets of one or more program modules. Type of the testing will be white box testing.

Functional Testing – It is a form of black box testing that focuses on the specifications or the functions of the application under test. The functional testing is carried out by feeding in inputs to the system and assessing the output.

User Acceptance Testing - UAT which the last step of testing phase performed prior to its delivery is also a type of black box testing carried out to ensure that the requirements of a specification are met. Usually, this is carried out with the business user in order to get their confirmation that what is built is what is required.

Considering 'Code Point’ is a program developed, based on research outcomes, as an opensource product to be used by anyone, only unit and functional testing are applicable here.

7.2.3 Testing Modules

All the modules have been tested as unit testing and have included test data below.

7.2.3.1 Testing Result Analyzer Module

No.	Test Case Name	Description	Inputs	Actual Output	Expected Output
01	Tool Name Dropdown	User gets an option to select the type of tool based on what they have uploaded in the master screen	Click on Tool name selection dropdown list	Displays names of all the tool mappers uploaded through the master screen	Displays names of all the tool mappers uploaded through the master screen
02	Tool Name Selection	Based on the tool selected by the user, relevant mapper is loaded	Select a tool from the dropdown list	Code picks the relevant mapper for analyzing the results	Code picks the relevant mapper for analyzing the results
03	Choose Local Path	User is given an option to give a file path in the local machine for selecting a single file for analysis	Click on 'Choose File' button against 'Local Path' field	Allows to select a single XML file	Allows to select a single XML file
04	Choose Local Folder	User is given an option to give a folder path in the local machine for selecting a folder with multiple files for analysis	Click on 'Choose File' button against 'Local Folder' field	Allows to select a folder	Allows to select a folder
05	Repeat Scheduler	User is given an option to set up a scheduled job run to pick data from a selected local folder and update the database so that whenever the user logs in and view the graph, the user will be able to see the latest results.	Check the 'Repeat' checkbox. Input the time to start the schedule.	Data extraction from folder to database runs at the scheduled time and frequency. The database reflects data	Data extraction from folder to database runs at the scheduled time and frequency. The database reflects data

			Input the frequency in hours	extracted from the latest updated file.	extracted from the latest updated file.
06	Remove repeat scheduler	A user who has initially set up a scheduled job to pick data from a folder is given the option to update the analyzer to run one time.	Select the job submitted to analyze code clone results. Uncheck the 'Repeat' Checkbox and update the job.	When the 'Repeat' checkbox is unchecked, clear the scheduled time and frequency automatically.	When the 'Repeat' checkbox is unchecked, clear the scheduled time and frequency automatically

Table 7.1: Analyzer Module – Testing Results

7.2.3.2 Testing View Dashboard Module

No.	**Test Case Name**	**Description**	**Inputs**	**Actual Output**	**Expected Output**
01	Select Job Dropdown	User gets an option to select the analyze job submitted for which s/he wants to see the dashboard	Click on 'Select Job' dropdown list	Displays names of all the 'Result Analyzer' jobs submitted by the user.	Displays names of all the 'Result Analyzer' jobs submitted by the user.
02	Dashboard Generation	Dashboard will be generated according to the job selected by the user	Select a result analyzer job from the dropdown list	Update the graphs and data tables showing details of the job selected	Update the graphs and data tables showing details of the job selected
03	Dashboard Comparison	Allows the user to view 2 dashboards generated with	Select the split view.	Update the graphs and data tables showing	Update the graphs and data tables showing

		different jobs on the same screen for comparison	Select the 2 result analyzer jobs to be compared from the dropdown lists displayed in both panes	details of the jobs selected in the 2 panes and display table with comparison results	details of the jobs selected in the 2 panes and display table with comparison results

Table 7.2: Dashboard Module - Testing Results

7.3 Evaluation

This research presents an integrated solution for code clone visualization. As the major contribution of the author, this has presented a Metamodel for user-friendly visualization which has major components of clone results, visualizations and user guidance/support.

During the literature review and evaluation of existing tools, the author identified several code clone tools that analyze the code using different techniques and considering different views. However, one thing that most of these tools lack is the dashboard visualization that provides the user with a graphical view of code clone results that the user can easily read and understand. This led the author to focus on the visualization side rather than code clone detection itself.

Further reading and research showed that with time, different researchers and developers have put effort into developing much-improved code clone detection engines. Although most tools cover Type I, II and II clone types and have not yet developed a tool that analyses Type IV clones, it is only fair to think that as per the current trend, it will happen sooner. However, depending on a developer's focus, different tools have their own pros and cons. It is up to the end user to identify the tool that best matches his/her requirements.

Considering the above, “Code Point” was developed with the flexibility of using any engine available in the market for code clone detection. The output from any of these engines can be uploaded to Code Point and it generates a dashboard view that is simple, user-friendly and easy to understand. This will enable the users to remove clones and help to maintain proper coding standards.

7.3.1 Evaluation against objectives

The final outcome of this research was largely influenced by the comprehensive research done by the author before arriving at a viable solution for the problem defined in the first chapter. The literature review given in chapter two provides the outcome of the first two objectives the author had set at the initial stages of the research. These two objectives are, ‘Conduct a study on code clone types and currently available code clone detection algorithms and tools’ and ‘Analyze currently available data visualization techniques and tools.

As described in detail in the previous chapters, the key functionality of Code Point involves analyzing code clone data and visualizing them in a graphical dashboard. Through the unit tests and integration tests conducted after development, the author has verified and validated that the developed tool meets the third and fourth objectives of this project.

Due to time limitations, the author was not able to meet the fifth objective where the intention was to provide clone analysis among solution repository versions as well. Hence this will be considered as future development. The last two objectives set are achieved through this documentation.

7.4 Summary

This chapter discussed in general different testing methods available and used in this project, the testing levels and how the proposed system was tested accordingly using test cases. Further the project as a whole was evaluated in general and against the objectives set at the beginning of the project.

Chapter 8

8. Conclusions and Further Work

8.1 Introduction

The previous chapter emphasizes on the testing methods used and the evaluation of the proposed solution against the project objectives that have been achieved.

This chapter summarizes what was discussed in the previous chapters. Further, through an evaluation of the proposed solution against what is requested by users, this chapter briefs the limitations of the solution and the improvements that can be done as future developments.

8.2 Overview of the Research

The main objective of this research was to address a major concern that most software development organizations and the programmers have. That is the lack of tools that represent the code clone results in an effective manner. Managers in IT projects, often lack the required governance on code quality as they do not get the statistics required to monitor the same. Sometimes, even though they see the statistics, it is not in a graphical format that helps them to easily grasp the current status and immediately focus on the key problem areas.

Due to the above reasons, the author has carried out a comprehensive research on the gestation of code clone visualization, type of code clones, techniques available to detect various types of code clones, and applications and tools available today to detect and visualize code clones. Taking the findings from the research also into consideration, the author has proposed a solution for detecting code clones in different stages of the project and implementing a system to visualize code clone results in a friendly manner.

Chapter three discusses the technologies that the author has considered for the proposed solution. Chapters four to six provide details on the approach used and the architecture of the tool. They explain in detail how the implementation was done. While discussing various methods of testing, chapter seven lists some of the major test cases and their respective results. In summary, this research presents an integrated solution for code clone visualizations.

In addition to the research, the author has presented a review paper “Trends in Code Clone Detection ” based on the past research papers which have presented in conferences previously. It has considered past 3 years major IT/CSE related conferences like European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), International Conference on Software Maintenance and Evolution (ICSME), Mining Software Repositories (MSR), International Conference on Software Engineering (ICSE) and reviewed the work which has been done related to code clone detection. Results of it imply that code clone detection has a major effect on current software engineering.

8.3 Challenges

- Internet bandwidth Limitations – Some analysis files contains a large amount of data. Due to that size of the files are considerably large. If we are dealing with slow internet speeds need to pay attention to reduce the object transactions between the server and clients.
- Lack of resources - For some areas, there are very limited resources and documentation. Especially when comes to code clone detection engines very limited no of resources are there. Few engines have been discontinued as well

8.4 Limitations of the proposed solution

- There are few limitations in the proposed solution. The main limitation is that there is no way to identify Type 4 code clones. That will need a strong and trained code clone detection engine.

- To configure the XML structure there should be at least basic programming knowledge. But it wasn't much of concern since this product is used by the developers and other IT professionals.
- Currently, no code clone detection engine is integrated into the application. Since the main focus is to integrate different types of tools this has been less priority.

8.5 Future Work

- Code point should be able to integrate with IDEs (ex:- Visual Studio)
- Use deep learning for type 4 code clone detection
- Use none relational database for more customizable behavior
- Various other software engineering data analytics tools that have been proposed in the literature [26, 27, 28] could be easily extended to conduct a systematic code clone detection research.

8.6 Summary

This chapter contains a summary of what was discussed in the report and the challenges faced by the author in implementing the proposed solution. Further, it discusses the limitations of ‘Code Point’, the solution proposed and implemented. Building on this discussion, the author also briefs what improvements can be made in order to provide a better and more user-friendly solution to the users.

References

[1] F.-M. Lazar and O. Banias, “Clone detection algorithm based on the Abstract Syntax Tree approach,” in 2014 IEEE 9th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI), Timisoara, Romania, 2014, pp. 73–78. Available: http://ieeexplore.ieee.org/document/6840038/

[2] H. A. Basit and S. Jarzabek, “A Data Mining Approach for Detecting Higher-Level Clones in Software,” IEEE Transactions on Software Engineering, vol. 35, no. 4, pp. 497–514, Jul. 2009. Available: http://ieeexplore.ieee.org/document/4796208/

[3] N. Göde and R. Koschke, “Incremental Clone Detection,” in *2009 13th European Conference on Software Maintenance and Reengineering*, Kaiserslautern, Germany, 2009, pp. 219–228. Available: http://ieeexplore.ieee.org/document/4812755/

[4] F. Deissenboeck, B. Hummel, and E. Juergens, “Code clone detection in practice,” in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE ’10*, Cape Town, South Africa, 2010, vol. 2, p. 499. Available: https://ieeexplore.ieee.org/document/6062267/

[5] P. Gautam and H. Saini, “Various Code Clone Detection Techniques and Tools: A Comprehensive Survey,” in Smart Trends in Information Technology and Computer Communications, vol. 628, A. Unal, M. Nayak, D. K. Mishra, D. Singh, and A. Joshi, Eds. Singapore: Springer Singapore, 2016, pp. 655–667. Available: http://link.springer.com/10.1007/978-981-10-3433-6_79

[6] C. K. Roy and J. R. Cordy, “Scenario-Based Comparison of Clone Detection Techniques,” in 2008 16th IEEE International Conference on Program Comprehension, Amsterdam, 2008, pp. 153–162. Available: http://ieeexplore.ieee.org/document/4556127/

[7] K. Kaur, “A Comprehensive Review of Code Clone Detection Techniques,” p. 5, 2015.

[8] B. S. Baker, "On finding duplication and near-duplication in large software systems," in 2nd Working Conference on Reverse Engineering, 1995.

[9] H. Kaur and R. Kaur, “Clone Detection in Web Application Using Clone Metrics,” *International Journal of Advanced Research in Computer Science and Software Engineering*, p. 9, 2014.

[10] C. M. Kamalpriya and P. Singh, “Enhancing program dependency graph based clone detection using approximate subgraph matching,” in *2017 IEEE 11th International Workshop on Software Clones (IWSC)*, Klagenfurt, Austria, 2017, pp. 1–7. Available: http://ieeexplore.ieee.org/document/7880511/

[11] J. Kaur, R. Kumar, and S. Kaur, “Design Code Clone Detection System uses Optimal and Intelligence Technique based on Software Engineering,” *International Journal of Advanced Research in Computer Science*, p. 7, 2017.

[12] Y. Sabi, Y. Higo, and S. Kusumoto, “Rearranging the order of program statements for code clone detection,” in *2017 IEEE 11th International Workshop on Software Clones (IWSC)*, Klagenfurt, Austria, 2017, pp. 1–7. Available: http://ieeexplore.ieee.org/document/7880503/

[13] H. Kaur and R. Maini, “Performance Evaluation and Comparative Analysis of Code-Clone-Detection Techniques and Tools,” *International Journal of Software Engineering and Its Applications*, vol. 11, no. 3, pp. 31–50, Mar. 2017. Available: http://www.sersc.org/journals/IJSEIA/vol11_no3_2017/4.pdf

[14] L. Li, H. Feng, W. Zhuang, N. Meng, and B. Ryder, “CCLearner: A Deep Learning-Based Clone Detection Approach,” in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Shanghai, 2017, pp. 249–260. Available: http://ieeexplore.ieee.org/document/8094426/

[15] L. Voinea and A. C. Telea, “Visual Clone Analysis with SolidSDD,” in *2014 Second IEEE Working Conference on Software Visualization*, Victoria, BC, Canada, 2014, pp. 79–82. Available: http://ieeexplore.ieee.org/document/6980217/

[16] A. Hanjalic, “ClonEvol: Visualizing software evolution with code clones,” in *2013 First IEEE Working Conference on Software Visualization (VISSOFT)*, Eindhoven, Netherlands, 2013, pp. 1–4. Available: http://ieeexplore.ieee.org/document/6650525/

[17] Green, P., Lane, P. C. R., Rainer, A. and Scholz, S. B. “Unscrambling Code Clones for One-to-One Matching of Duplicated Code”, Technical Report 502, School of Computer Science, University of Hertfordshire, 2010. [Online]. Available: https://core.ac.uk/download/pdf/1641752.pdf

[18] S. Harris, "Simian: Similarity analyser," 2011. [Online]. Available: http://www.harukizaemon.com/simian/.

[19] K. Hotta, Y. Sasaki, Y. Sano, Y. Higo, and S. Kusumoto, “An Empirical Study on the Impact of Duplicate Code,” Advances in Software Engineering, vol. 2012, pp. 1–22, 2012. https://www.hindawi.com/archive/2012/938296/

[20]"The Archive of CCFinder Official Site," 2015. [Online]. Available: http://www.ccfinder.net/doc/10.2/en/whats.html

[21] S. Wagner, A. Abdulkhaleq, K. Kaya, and A. Paar, “On the Relationship of Inconsistent Software Clones and Faults: An Empirical Study,” in 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), Suita, 2016, pp. 79–89. http://ieeexplore.ieee.org/document/7476632/

[22] Asaduzzaman, M. “Visualization and Analysis of Software Clones”, MSc Thesis, University of Saskatchewan, Saskatoon, 2012. [Online]. Available: https://www.cs.usask.ca/~croy/Theses/Thesis_Asaduzzaman_January2012.pdf

[23] Koschke, R., Falke, R., Frenzel, P.: Clone detection using abstract syntax suffix trees. In: Proceedings of the 13th IEEE Working Conference on Reverse Engineering, Italy, pp. 253–262, October 2006

[24] Tairas, R., Gray, J.: Phoenix-based clone detection using suffix trees. In: Proceedings of the 44th ACM Annual Southeast Regional Conference (ACM-SE 2006), Melbourne, pp. 679– 684, March 2006

[25] Wijesiriwardana, C. and Wimalaratne, P., 2017, November. Component-based experimental testbed to faciltiate code clone detection research. In 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS) (pp. 165-168). IEEE.

[26] Wijesiriwardana, C. and Wimalaratne, P., 2018. Fostering Real-Time Software Analysis by Leveraging Heterogeneous and Autonomous Software Repositories. IEICE TRANSACTIONS on Information and Systems, 101(11), pp.2730-2743.

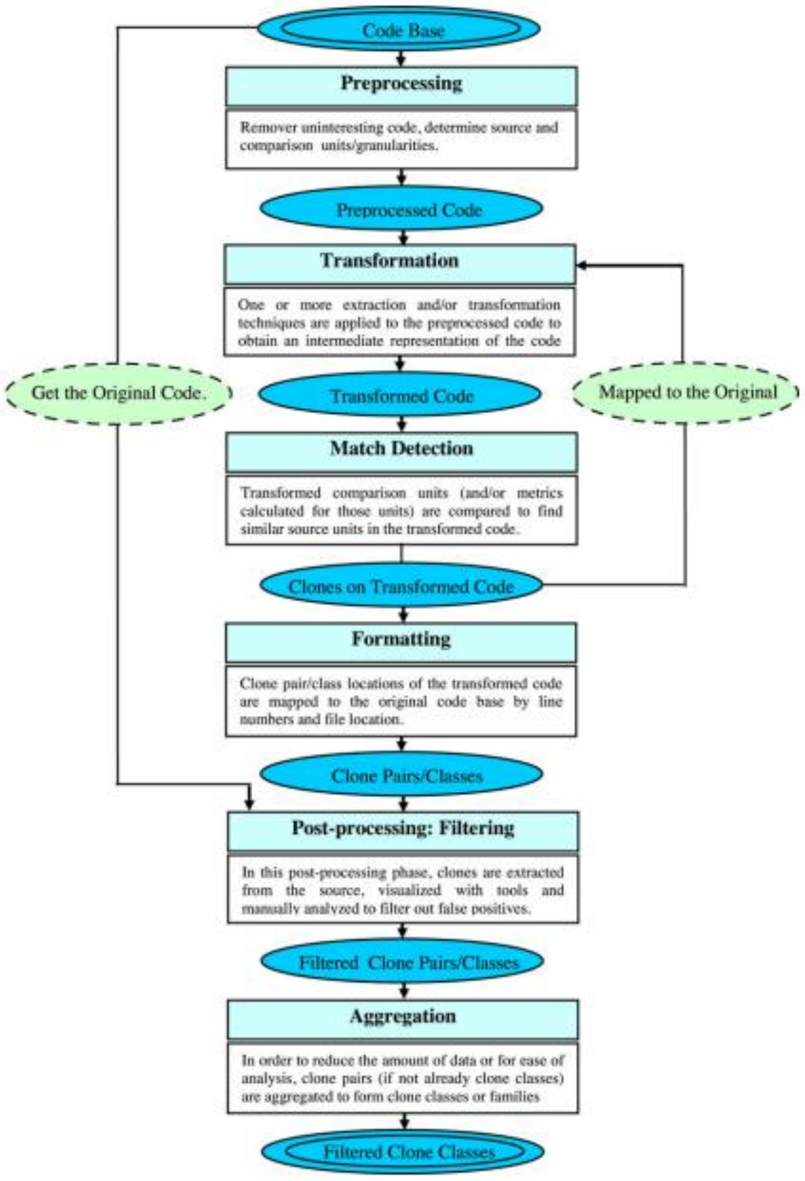
[27] Wijesiriwardana, C., Ghezzi, G. and Gall, H., 2012, December. A guided mashup framework for rapid software analysis service composition. In 2012 19th Asia-Pacific Software Engineering Conference (Vol. 1, pp. 725-728). IEEE.

[28] Wijesiriwardana, C. and Wimalaratne, P., 2019. Software Engineering Data Analytics: A Framework Based on a Multi-Layered Abstraction Mechanism. IEICE Transactions on Information and Systems, 102(3), pp.637-639.

Appendixes

Appendix A

Generic code clone detection model


Code Base
Preprocessing
Remover uninteresting code, determine source and comparison units/granularities.
Preprocessed Code
Transformation
One or more extraction and/or transformation techniques are applied to the preprocessed code to obtain an intermediate representation of the code
Get the Original Code.
Transformed Code
Mapped to the Original
Match Detection
Transformed comparison units (and/or metrics calculated for those units) are compared to find similar source units in the transformed code.
Clones on Transformed Code
Formatting
Clone pair/class locations of the transformed code are mapped to the original code base by line numbers and file location.
Clone Pairs/Classes
Post-processing: Filtering
In this post-processing phase, clones are extracted from the source, visualized with tools and manually analyzed to filter out false positives.
Filtered Clone Pairs/Classes
Aggregation
In order to reduce the amount of data or for ease of analysis, clone pairs (if not already clone classes) are aggregated to form clone classes or families
Filtered Clone Classes