A Qualitative Interpretation of Code Clone Detection Techniques

Amandeep Kaur, Sandeep Sharma

Abstract Clone is the software code snippets that are similar to each other with little modifications. There is a 10-20 percent clone mostly present in the software. Many techniques are developed for detection. With the code clone detection, the software developer gets an idea of removing, refactoring the clone. Code clone has both advantages and disadvantages in the particular software. In this paper, we explore the types of code clones, advantages, and disadvantages, the reason for cloning. Typically, this paper describes various techniques by using several parameters. Lastly, we discuss gaps in the research.

Keywords: Code Clone Detection, Data mining, Tokens, Code Clone, Abstract Syntax Trees.

I. INTRODUCTION

Code snippets that are copied and paste in the software code with or without change results into code clone. Different authors provide different definitions of code clones. Various studies provide information about the percentage of duplication in the source code. Commonly, there is a 10-20 percent cloning present in the software code [3][5][22]. Due to advancements in technology, the companies spend more cost on the maintenance of software. For the maintenance of software, code clone is detected and should be removed or refactored according to the requirements. Because if code fragment that contains bug is copied or duplicated then each it is clone also contains that bug and it is difficult to discover the bug in the large software that contains thousands or millions of LOC. Roy[25] and Koschke[16] presents a comprehensive review of the clone found in the software. In the remaining paper, Section 2 presents the background. Section 3 detailed the various techniques of clone detection. Section 4 shows the result analyses. In the last, we conclude this paper.

II. BACKGROUND

In this, we discuss the classification of clones, advantages, and disadvantages, why clone occurs in the software.

A. Classification of Clones

Broadly, the code clone divided into four categories. Fig. 1 shows the classification of clones.

- Type-1: Code snippet similar to other snippets with the only changes in whitespaces, comments leads to this clone. Type-1 can be detected easily with the text-based and token-based technique.
- Type-2: Type-2 clone occurs due to alteration in name of the identifier, literals, and variables, keywords. They can be detected with the token and metric-based technique.
- Type-3: Results from addition, deletion of lines of codes. They can be detected with a tree-based and graph-based approach.
- Type-4: A Graph-based and hybrid approach can be used to detect a type 4 clone. This clone occurs when two code snippets have similar functionality but the difference in their structure.

B. Advantages of code clone

- A better understanding of the program: Code clone helps to better understand the code fragment as a similar code appears many times in the program [24].
- Code reuse: The code fragment is difficult to design but similar functionality is available in another code fragment then the programmer can easily reuse that.
- Bug identification: Bug can be identified in the clone which helps to correct an error in the original code and vice versa.
- Plagiarism detection: Plagiarism can be easily detected through the code cloning techniques. Mostly the plagiarism did at the time of the assignment or project-related work [3][12].
- Copy infringement: Copy infringement can be performed through the code cloning technique as the many software companies use the code of another company's software without their permission [31].
- Clone removal/ refactoring: Code cloning helps to determine which clone should be removed or refactored according to requirements.

C. Disadvantages of code clone

- Bug propagation: The bug occurs in the particular code fragment then all its clone also contains that error which is known as bug propagation [20].
A Qualitative Interpretation of Code Clone Detection Techniques

- **Maintenance cost increases:** Various studies show that about 90 percent of software development cost spends on software maintenance [15].
- **Bad quality design:** The software that consists of too much clone is considered to be bad quality software as it is difficult to understand the program [13].
- **Difficulties in Modifications or Improvements:** Modifications are difficult to make as changes in the original code requires the changes in its entire related clone [11].

D. Why clone occurs

- **Cloning by accident:** Sometimes the programmer writes the codes that are similar to another code fragment as no other solution for the given problem is available [25].
- **Time/resource constraint:** Programmer copy, paste or make some modification in the original code if the resources or time available with him is limited [14].
- **Language constraints:** Sometimes the programming language has limited features, therefore, programmers use the code with or without modifications [14].
- **Programmer's constraints:** The Programmer does not have the knowledge of problem domain sometimes that leads to the use of the already existing code in the program [25].
- **Risk of designing the new code:** Cordy[7] says that in financial software, updations or some modifications are made in the existing code as the risk associated with it related to money. So, financial products are not changed very much frequently.

III. TECHNIQUES FOR DETECTION OF CODE CLONE

Many researchers propose various detection techniques. These techniques are differentiated according to source code representation, comparison method, etc. Fig. 2 shows the classification of techniques.

A. Textual techniques

Text-based clone detection can easily identify the Type-1 clone. Text-based clone detection technique has a very less false positive clone. Johnson [11] proposes the tool which is applied to the source code of length 40 megabytes. Roy and Cordy [26] presents a technique that involves three phases. First, using parsing to remove noise, make the standard format, and break the program statements into parts. Second, using island grammar and parsing to effectively find the potential clone. Third, code normalizations are provided using transformation rules to filter out uninteresting part of a potential clone. Lee and Jeong [19] describe the algorithm that detects exact copy and bit/pieces of similar code.

B. Lexical techniques

In these techniques, tokens are generated from code using lexer. The lexical technique has more false-positive then textural. Li et al. [20] developed a tool named CP-Miner. It represented statements into number and similar statements have similar numbers. The sequence of numbers is stored into a database and frequent mining is performed to find a similar sequence of numbers. The tool focuses on the analysis of code by size, by granularity and by a degree of modifications. The tool finds the clone in two popular operating systems.

Basit et al. [4] describe a technique that finds repeated tokens by implementing the concept of the suffix array. The token-based technique involves the least code transformations. Yuan and Guo [33] depict the approach that using cosine similarity to find the clone. Experiments performed on the JDK7 and Linux kernel.

C. Syntactic techniques

These techniques deal with the program's syntax. It incorporates a tree-based and metric-based approach. Table 1 shows the comparative portrait of these techniques.

![Fig 2. Classification of techniques of code clone detection](Immage)

- **Metric-based approach**

The metric-based technique detects the syntactic clone. Mayrand et al. [22] describe the approach that includes functional metrics for detection. These metrics are incorporate into points for comparison to identify the function clone in the source code. As the level increases the false positive rate also increases. Patenaude et al. [23] describe the technique that considers source code object metrics and interface metrics for the identification of JAVA clone. The experiment performed on the 500KLOC of java source code. Lanubile and Mallardo [18] describe the method for the detection of clone present in web applications. The homonym function refers to potential clone function. The homonym function is represented together by three measures LOC, ELOC, CLOC. Abd-El-Hafiz [1] presents the method that includes the four steps. Firstly, the preprocessing of code performed. In the next step, the function and metric are extracted. In the third step, the fractal clustering algorithm is applied to cluster the functions and in the last, the clone classes are extracted.

- **Tree-based approach**

In this technique, the Abstract syntax tree generated from the code snippet. Yang [32] presents the algorithm. The text-based technique doesn't identify what kind of differences because some program has rigid syntactic structure. The algorithm is based on a dynamic programming scheme. The pretty-printer is used to highlight the difference between the two programs. Baxter et al. [5] describe the approach that uses three algorithms to find clone. The first approach finds the subtree clone. The next algorithm finds the sequenced clone.
The last algorithm detects the near-miss clone. The method also enables the removal of detected clones. This technique applied to the C language software system to detect the clones. Wahler et al. [30] describe the clone detection mechanism that detects type 1, 2 clones. In this, the frequent itemset finding technique is used. XML representations exist for Java, Prolog, and C++. Koschke et al. [17] describe the technique that finds the syntactic clones in linear complexity. This technique is compared with other techniques using Bellon benchmark. The suffix tree detection applied to find the clone pairs. Sager et al. [27] describe the method for clone detection in JAVA language. Tree similarity algorithm which includes the tree edit distance, Top-down maximum common subtree isomorphism, bottom-up maximum common subtree isomorphism is applied to find the common subtree. Bulychev and Minea [6] describe the approach that implemented the anti-unification method. Duplicated code is identified by setting the minimum distance threshold. The tool currently supports Java and Python languages.

<table>
<thead>
<tr>
<th>Author</th>
<th>Tool</th>
<th>Code Representation</th>
<th>Comparison Granularity</th>
<th>Comparison technique</th>
<th>Types of clone detected</th>
<th>The language used for an experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Johnson [11]</td>
<td>Not available</td>
<td>Substring</td>
<td>Chunks of text</td>
<td>Karp and robin fingerprinting based string matching</td>
<td>Type 1, 3</td>
<td>C</td>
</tr>
<tr>
<td>Roy and Cordy [26]</td>
<td>NICAD</td>
<td>Sequence of items</td>
<td>Text</td>
<td>LCS(longest common subsequence)</td>
<td>Type 1, 2, 3</td>
<td>C</td>
</tr>
<tr>
<td>Lee and Jeong [19]</td>
<td>SDD</td>
<td>Text</td>
<td>Not fixed</td>
<td>n-Neighbor distance</td>
<td>Type1, 3</td>
<td>JAVA</td>
</tr>
<tr>
<td>Li et al. [20]</td>
<td>CPMiner</td>
<td>Block represents a sequence of numbers</td>
<td>Token sequence</td>
<td>Frequent subsequence mining</td>
<td>Type1,2,3</td>
<td>C, C++</td>
</tr>
<tr>
<td>Basit et al. [4]</td>
<td>RTF</td>
<td>Token sequence</td>
<td>Token</td>
<td>Suffix array algorithm</td>
<td>Type1,2,3</td>
<td>C</td>
</tr>
<tr>
<td>Yuan and Guo [33]</td>
<td>Boreas</td>
<td>Variable</td>
<td>Count vector</td>
<td>Cosine similarity</td>
<td>Type1,2,3</td>
<td>C, C++, JAVA</td>
</tr>
<tr>
<td>Yang [32]</td>
<td>Not available</td>
<td>Parse tree</td>
<td>AST node as Token</td>
<td>Tree matching using dynamic programming</td>
<td>Visualization only</td>
<td>C</td>
</tr>
<tr>
<td>Baxter et al. [5]</td>
<td>CloneDR</td>
<td>AST</td>
<td>AST node</td>
<td>Hash value based matching</td>
<td>Type 1, 2, 3</td>
<td>C, Java, C++</td>
</tr>
<tr>
<td>Wahler et al. [30]</td>
<td>Not available</td>
<td>An XML representation of AST</td>
<td>Statement</td>
<td>Frequent itemset mining</td>
<td>Type 1, 2</td>
<td>Java, Prolog, C++</td>
</tr>
<tr>
<td>Evans et al. [8]</td>
<td>Not available</td>
<td>An XML representation of AST</td>
<td>AST node</td>
<td>Graph-theoretic approach</td>
<td>Type 1, 2, 3</td>
<td>Java, C++</td>
</tr>
<tr>
<td>Koschke et al. [17]</td>
<td>Not available</td>
<td>AST</td>
<td>Token(AST node)</td>
<td>Suffix tree detection</td>
<td>Type 1, 2</td>
<td>C, Java</td>
</tr>
<tr>
<td>Sager et al. [27]</td>
<td>Google</td>
<td>FAMIX tree</td>
<td>FAMIX node</td>
<td>Tree comparison algorithm</td>
<td>Type 1, 2, 3</td>
<td>Java</td>
</tr>
<tr>
<td>Bulychev and Minea [6]</td>
<td>Clone digger</td>
<td>Anti-Unifier tree</td>
<td>Subtree</td>
<td>Suffix tree detection</td>
<td>Type 1, 2, 3</td>
<td>Python, Java</td>
</tr>
<tr>
<td>Mayrand et al. [22]</td>
<td>CLAN</td>
<td>Metrics</td>
<td>Metrics for function</td>
<td>4 point of comparison</td>
<td>Type 1,2,3</td>
<td>C</td>
</tr>
<tr>
<td>Patenaude et al. [23]</td>
<td>Not available</td>
<td>Metrics</td>
<td>Metrics for method</td>
<td>Distinct Comparison</td>
<td>Type 1, 2,3</td>
<td>Java</td>
</tr>
<tr>
<td>Lanubile and Mallardo [18]</td>
<td>Not available</td>
<td>Homonym function</td>
<td>Function</td>
<td>Visual inspection</td>
<td>Type1,2,3</td>
<td>Web language</td>
</tr>
<tr>
<td>Abd-El-Hafiz [1]</td>
<td>Not available</td>
<td>Metrics for function</td>
<td>Function</td>
<td>Fractal clustering</td>
<td>Type1,2,3</td>
<td>C</td>
</tr>
<tr>
<td>Komondoor and Horwitz [15]</td>
<td>Not available</td>
<td>PDGs</td>
<td>PGD nodes</td>
<td>Isomorphic PDG subgraph matching by Backward slicing</td>
<td>Type 3, 4</td>
<td>C</td>
</tr>
</tbody>
</table>
A Qualitative Interpretation of Code Clone Detection Techniques

<table>
<thead>
<tr>
<th>Method</th>
<th>Method</th>
<th>PDG nodes</th>
<th>PDGs</th>
<th>Isomorphic PDG subgraph matching algorithm</th>
<th>Type 1, 2, 3</th>
<th>C, C++, Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>Liu et al. [21]</td>
<td>GPLAG</td>
<td>PDGs</td>
<td>PGD nodes</td>
<td>Isomorphic PDG subgraph matching algorithm</td>
<td>Type 1, 2, 3</td>
<td>C, C++, Java</td>
</tr>
<tr>
<td>Gabel et al. [9]</td>
<td>Not available</td>
<td>Vector</td>
<td>characteristics</td>
<td>Tree-based Locality-sensitive hashing detection</td>
<td>Type 3, 4</td>
<td>C, C++</td>
</tr>
<tr>
<td>Higo et al. [10]</td>
<td>Not available</td>
<td>PDGs</td>
<td>PGD edges</td>
<td>Code clone detection module</td>
<td>Type 3, 4</td>
<td>Java</td>
</tr>
<tr>
<td>Agrawal and Yadav [2]</td>
<td>Not available</td>
<td>Tokens</td>
<td>Tokens</td>
<td>Line by line comparison</td>
<td>Type 1, 2, 3</td>
<td>C</td>
</tr>
<tr>
<td>Sheneamer and Kalita [28]</td>
<td>Not available</td>
<td>Feature Vector</td>
<td>Method</td>
<td>Classification algorithms</td>
<td>Type 3, 4</td>
<td>Java</td>
</tr>
<tr>
<td>Sheneamer [29]</td>
<td>CCDLC</td>
<td>Feature Vector</td>
<td>Method</td>
<td>Clustering and Deep learning classifier</td>
<td>Type 3, 4</td>
<td>Java</td>
</tr>
</tbody>
</table>

D. Semantic techniques

Semantic clone occurs when the syntax of two fragments is different but they perform the same functionality. These techniques find the semantic clones. The semantic approach mostly detects type 3 clone. It comprises of graph-based approach and a hybrid approach.

- **Graph-based approach**

Program dependency graphs of code snippets are made in this approach. Komondoor and Horwitz [15] describe the tool that identifies clones using backward slicing. After that the potential clone is removed that is not the actual clone and then the actual clone is combined to form the larger groups. Liu et al. [21] present the tool that detects plagiarism through a program dependency graph. Many plagiarism tools are available for academic use. GPLAG makes use of the combination of lossy filter and lossless filter which properly prunes the search space. Gabel et al. [9] describe the method that focuses on the simple representation of a similar graph to a similar tree by mapping the program dependency graph to related structured syntax. The vectors are generated from the abstract syntax trees which are clustered according to the similarity to generate clone pair, clone classes. Higo et al. [10] describe the approach that comprises the analysis and detection phase for finding the clone pair.

- **Hybrid approach**

The hybrid techniques are developed to encounter the limitation of the existing techniques. The hybrid technique includes both semantic and syntactic properties. Agrawal and Yadav [2] describe the approach in which the Textual and token technique are combined to identify the clone of type 1, 2 and 3. Sheneamer [29] describes the method that makes use of abstract syntax tree and program dependency graph for detection of the clone.

IV. RESULT ANALYSES

In this section, we are analyzing these techniques in the form of the gap in the technique or tools that are discussed above. Almost every technique or tool identifies Type 1, 2 clones as they are easy to identify. Type 4 or a functional clone is tough to identify. The efficiency, scalability, time and space requirement of the clone detection tool and technique can be improved by designing the detection algorithm by employing other domains. Table 2 presents gaps in the research of the papers discussed above.

**Table II. Gaps in research**

<table>
<thead>
<tr>
<th>Sr No.</th>
<th>Reference No.</th>
<th>Gaps in Research</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[11]</td>
<td>It does not compute the complexity of the technique.</td>
</tr>
<tr>
<td>2</td>
<td>[26]</td>
<td>NICAD is experimented to find the clones in small and medium-size software. It can detect a clone in only one language.</td>
</tr>
<tr>
<td>3</td>
<td>[19]</td>
<td>It cannot compute the recall and precision of the algorithm.</td>
</tr>
<tr>
<td>4</td>
<td>[20]</td>
<td>The CP-Miner could not detect the bug related to semantic information.</td>
</tr>
<tr>
<td>5</td>
<td>[33]</td>
<td>The complexity of this technique is not computed.</td>
</tr>
<tr>
<td>6</td>
<td>[32]</td>
<td>The technique ignores the semantic information of the program. The pair of non-terminal should not be considered for comparison. This technique requires parser and pretty-printer for language independence.</td>
</tr>
<tr>
<td>7</td>
<td>[5]</td>
<td>This technique does not focus on the removal of the clone</td>
</tr>
<tr>
<td>8</td>
<td>[30]</td>
<td>The clone detection mechanism does not find the Type-3 clone.</td>
</tr>
<tr>
<td>9</td>
<td>[8]</td>
<td>The complexity of this algorithm is not computed.</td>
</tr>
</tbody>
</table>
Software clone results from the copy and paste of the code snippets from one to another software or to same the software. More research is done on the detection than the management of the clone find in the software. In this paper, we discuss the different techniques and tools. Researchers and authors used different algorithms for the identification of clones. From the past two decades, most of the techniques detect the syntactic clone that deals with the syntax. Type 4 or functional clones are hard to find due to their nature. This paper guides the software developer to use techniques that are corresponding to their requirements for the detection of a clone. The researcher also makes changes or updates the existing techniques.

**ACKNOWLEDGMENT**

I am very thankful to Dr. Sandeep Sharma.

**REFERENCES**


A Qualitative Interpretation of Code Clone Detection Techniques


AUTHORS PROFILE

Amandeep Kaur, Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar, India. Amandeep Kaur is pursuing Mtech (CSE) from Guru Nanak Dev University, Amritsar. Her research interests include software engineering and machine learning. She did her Bachelors of Engineering in Computer Science and Technology from Sant baba bhag Singh institute of engineering and technology, Jalandhar, India

Sandeep Sharma, is head Professor in the Department of Computer Engineering and Technology, Guru Nanak Dev University, Amritsar, India. He received his B.E, M.E, and PhD in Computer Science and Engineering His area of research interest is in Big Data, Cloud computing, and parallel processing.