A Comprehensive Study of Finding Copy-and-Paste Clones from Program Source Codes

Kamran Khan,
Shaheed Zulfiqar Ali Bhutto
Institute of Science and Technology
Islamabad, Pakistan
Kamran_3388@yahoo.com

Saif Ur Rehman, Kamran Aziz
CORDE: Center of Research in Data Engineering, MAJU University,
Islamabad, Pakisan
Saifi.ur.rehman@gmail.com,
kamrandik@gmail.com

Simon Fong
Department of Computer and Information Science
University of Macau
Macau SAR
ccfong@umac.mo

Abstract—**In any programming language source code, the code that is repeated is called the clone. The clone detections have got much attention in the recent years. In literature there are a number of clone detection techniques have been proposed. These techniques includes CP-Miner, CC-Finder etc. each of these techniques attempts to detect the clone from the source code of various programming languages. In this study, we will provide comprehensive details of the various clone detection techniques proposed so far. These techniques have been critically evaluated based on a no of efficiency measure parameters. In our future work we will propose our own clone detection technique that will more efficient and accurate in terms of code clone detection from multiple programming languages.**

I. Introduction

In software development, programmer often uses copy-and-paste technique. The aim of copy-and-paste technique is save efforts in manually typing over the codes again in computer program software. It is mandatory to detect and remove such clones. In the past, many techniques have been proposed. For example, [1], [2] use copy-and-paste detection tool for detecting code clones.The main issue associated to clones in programming languages is that copy-and-paste introduces bugs in programming code due to forgetting to change identifiers each time right through the code that is pasted from some somewhere else [2].

There are many issues associated with copy-and-paste source code when the size of the code get bigger, furthermore handling these issues are even greater challenge. A bug in one module is reproduced in every copy [3]. As many of the copy-and-paste codes are not documented which provides that which part of the code repeated in which parts, it is extremely hard to find and fix such programming bugs. These bugs are the main source of issues related to maintenance of existing software and removing such bugs are more complex and costly. Moreover, understanding and reusing such code is also a challenge for programmers which reduce the level of abstraction and adding new functionality to the code [3].

Different research studies have already been done to identify the duplicates in software applications [12]. However, these techniques have limitations regarding the support for certain programming languages. In literature different tests have been performed on known tools and techniques for clone detection but the results reveals that there is none good approach that produce efficient and optimal output.

In software engineering the topic of clone detection has received much attention in last decades. In literature several methods for clone detection have been proposed. These techniques are widely used in software domain[8] the existing clone detection techniques focuses on finding similar codes in source code, known as clone, which resulted in reduced update issues and application size. These gains, however, can be improved by evaluating the level of clone analysis [4]. Previous studies showed that these gains can be detect design level similarities which can aid to the software design in terms of code optimization and understanding the design of software.

The rest of the paper is organized as follows: After presenting some basic definitions about clones in Section 2, we present some summaries and strong points and weak points of clone detection techniques. In Section 3, we present a different comparison of these clone detection techniques have been critically evaluated.

II. Literature Review

In [1] Kamiya *et al.* have proposed a technique for detecting clones in large source codes. Their technique is called CC-Finder. The CC-Finder is based on the following elements: (1)-transformation rules; (2) a token-based comparison and (3) optimization techniques. These three elements attempt to improve the performance and efficiency of clone detection in the source code. The CC-Finder works as follow. In first step, the source code is divided into tokens. Afterwards, all these tokens are concatenated into a single token file. Then clones detection is performed on this single file. During the token analysis process, the white spaces are removed between the tokens and these characters are sent to the formatting step. In the second step, the token sequence is transformed using transformation rules. In this step the special characters, operators and name spaces are discarded. In the third step, following the second step transformation the same pairs are detected as clone pair [1]. In the fourth and last step the proposed method locates the copy-and-paste and converted them into the line numbers on the original source code. Afterword [1] is implemented in C++ language. The results showed that [1] can extract the clone from different languages source code including C, C++, Java and COBOL. Although, CC-Finder is very efficient for clone detection but this technique is failed to detect the clones that come from two different programming language source codes.

In [2] Li *et al.* have proposed another useful tool for clone detection. The method proposed in [2] applies techniques of data mining to identify copy-and-pasted code in a huge source code. The proposed technique is capable to find operating system associated bugs [2]. The proposed approach has two main functionalities (1). Detecting copy-and-paste code segments (2) finding copy-and-paste related bugs. It works as follow: In step reduce the development time. A software developer in his development career frequently uses copy-and-paste and they use the same code again and again. Copy-and-paste method reduces programming effort and time therefore programmer uses copy-and-paste rather than writing new code from scratch. In literature a large number of techniques have been developed for detecting duplicated code in

No.1 the proposed approach first change the problem into frequent subsequence mining problem [2]. The author used CloSpan algorithm to detect the basic copy-and-paste segments. In this step the proposed approach detects copy-and-paste very competently. To further accelerate the process the [2] eliminate unnecessary comparison by using frequent subsequence mining method. In second step, the proposed approach found code clones which were the main source of bugs. Furthermore to evaluate the efficiency of the [2] different experiments were performed using [2] the results showed that the execution time of the proposed technique is optimal, proposed approach took 11-12 minutes to find 101,699-198,605 code clones in a source code segment in a Linux. Further [2] was compared with [1], the comparisons established that execution of [1] is similar to [2] but the [2] is more efficient and detects much more copy-and-paste segments. So there work is supported by very good explanatory examples. The currently developed tool is only work for programs written in C or C++ and their tool detects only simple cases of errors which is not complicated.

In [3] Wahler *et al.* have proposed a new method for detecting clones. Their technique is based on frequent itemset. The proposed technique works as follow; in first step, it takes source code from different compilers as an input, the [3] then creates a consistent code from the input using corresponding parser. In the next step, it uses the popular frequent itemset finding technique to generate most frequent itemset from the XML file. In the last step, it removes or detects the itemset as the duplicate code. By using JDK, the authors have implemented their approach. Although, the experimental results shows that it can efficient and accurate to detect the clones from the source code; but this detection is limited to clone of type 2.

In [4] Basit *et al.* have proposed a method for detecting or finding the clones in a source file. Their technique is based on token-base technique for finding (a) simple clones (b) finds co-occurring clones in a different files by using a frequent itemset mining and it perform file clustering to find clusters that are similar. The execution process of [4] is as follow; In first step, the input file is converted into tokens and the efficient suffix array based algorithm is used to find the repeated tokens. The first step provides a data in a suitable format for second step. In the second step, [4] detect clones that occur together and frequently in a different source by using a frequent itemset technique. In this step they found some un-significant files coverage, for these un-significant files coverage they used third step (clustering highly clone method). The proposed approach is implemented using the C++ compiler for efficiency measure; they used the Java source file as an input to find the clones from it. Although, this technique can find clone but this detection is limited to just one programming language.

In [5] hummel *et al.* have proposed a new technique for finding clones. Their method is known as “Novel Index Based”. Their technique is used for both incremental and scalable to a very large source code. The incremental based method consists of three steps. In a first step the source code is read from a disk and converted into tokens. So the result of this step is the list of normalized statements for each file. In the second step, it finds the global statements list for equal sub strings. The result of this step is cloning information on the level of statement sequences. In the last step this technique creates cloning information on the level of code regions from cloning information on the level of normalized statements. Their clone index approach is similar to the inverted index used in document retrieval system. This method is not only used for the retrieval of all the clones enclosed in a source file but it is also useful for the efficient retrieval of clones from the source file. They use their method in a distributed environment across different machines for the creation of index and retrieval of clones. In a distributed environment they experiment on Java, C and C++ source codes. They have showed the efficiency of their work experimentally.

In [6] Baxter *et al.* proposes the Abstract Syntax Tree (AST) to detect the clones in the source code. Their proposed approach is the simple one as compared to the rest of the clone detecting techniques available in the literature. This technique is efficient and can detect the clones accurately from the code as compare to other clone detection techniques which only focus on either the string matches or near misses only on the body of the underlying functions. The technique proposed in [6] first of all the source code is parsed and from this parsed code an Abstract Syntax tree is produced. Then a set of three algorithms have been applied on Sub-Tree Clones. In this step the similarity is measured between the sub trees. This similarity is measured using the formula given below in Eq.1. This AST is used to find the clones from the source code. The first algorithm is called Finding

$$Similarity = 2 \times S / (2 \times S + L + R) - \quad Eq.1 \quad [1]$$

Where, S is the number of shared nodes [6], L is the number of different nodes in sub-tree 1; R is the number of different nodes in sub-tree 2 [6]. The [6] applied their proposed technique on real world software systems, which confirms that this new technique [6] can generated more accurate clone detection results as compared to the other clone detection techniques. This technique has following strong points. (1) This technique is straightforward for detecting the clone in the source code. (2) It is more efficient as compare to it is proved form the experimental results [1]. (3) It will defiantly open the new direction for the detection of the clone code in the source code of the program. Although [6] has given new dimensions to the clone detection mechanism but the technique it uses, AST uses a thresholds value which eliminates small trees comparisons but the proposed method failed to find the close clones such that in which one clone instance is small, and the other is large [6].

In [7] Falke *et al.* have proposed a method called “Abstract syntax Suffix Tree”. Their method is used to find clones in based on [6]. The approach proposed in [7] is used to find syntactic code clones in an optimal manner. Their method is very efficient especially in token base clone detection as suffix tree token base clone detection is very fast. Basically suffix tree is originally used for efficient string searching. The suffix tree represents a string where every suffix is shown through a path from root to a leaf and the edges are labeled with substrings [7]. Comparison between token base ad AST base shows that suffix-tree-based study offers a lot of advantages over other methods. Their comparison shows that token base clone detector is familiar to a new language in a very small time. The algorithm of linear time is consists of the following steps; In a first step, the source code is parsed and from this parsed code an Abstract Syntax tree is produced. Then in the 2nd step they serialize the AST nodes by a preorder traversal [7]. In the 3rd step each AST node represents a token and suffix tree clone detection is based on token. In this step the actual value of string is not disturbed when they are represented as a node. The output of this step is set of clone classes. The clone classes are consisting of AST node sequences and these sequences may or may not be syntactic clones [7]. Therefore in last and 4th step these sequences are decomposed into syntactic clones. They compare their method with 9 other techniques which is a plus point of their technique. Their method is good to find syntactic clones in a source but their technique is less efficient.

In [8] Jablonski *et al.* have proposed a new approach for detecting a copy-and-paste clones and changing the identifiers name in an integrated development environment. This technique is called a CReN. CReN finds the code clones which is occurred during the copy-and-paste in the integrated development environment (IDE) and the proposed method in [8] uses set of rules which are based on the identifiers relationship in the code fragments. Their tool CReN is implemented as an eclipse plug-in in Java [8]. They have performed experiment on the source code which is written in a Java language. Then they apply the proposed tool to find the copy-and-paste in the input file, the proposed tool is a set of identifiers placed in a code fragment and map the identifiers pairs which are placed in a same location. Their tool uses AST API of eclipse JDK framework [8] and this AST allows the proposed tool is used to create connection in clone code. The [8] tool is also used for renaming the identifiers just within copy-and-paste fragments. Experimentally their tool is so good for clone detection but it is used only for Java source code.

In [9] Uchida *et al.* have presented, the broad analysis of a code clones and for this purpose they use 125 packages of open source which is written in a C language. For analysis they use a CC-Finder technique to determine the code clones and evaluate them statistically. They also use a clone warrior tool for code clone identification for identifying and classifying the code clones and to examine the causes for their production. For token base clone detection they use CC-Finder method which have an industrial potency and applicable to a million lines size of source c ode. For detecting a code clones using a CC-Finder method they follow the following steps; In the first step, the input files are converted into tokens based on the lexical analyzer rules of the programming language [9]. In this step, the proposed method prunes white spaces and remarks or comments. In the next step, data types, variables and constants are replaced by the same respective tokens [9]. This replacement is useful for identifying a code clones as pair of code lines where only the variables names are differ. In the last step, all the substrings which are transferred as token sequences; a pair of identical substring is detected as clone pair. So, for visualization of a code clone they used a clone warrior tool which consists of a graphical user interface (GUI). Using clone warrior tool, first they specify the criteria for clone detection i.e. the smallest amount of a fragments. In next they specify the input files which have to be analyzed and after all the clones are detected from the input file and stored in code clone storage and provide the view in three forms i.e. the file list view, the code clone list view and the source code editor. This work is supported by very good explanatory examples. So their experiment works fine but it is applicable only to a source file which is written in a C Language.

In [10] Jia *et al.* have proposed a new technique for the precise and efficient clone identification. This approach is called Kclone. It combines lexical and local dependencies analysis in order to generate precise clones from the data source without affecting the clone deduction speed. This technique works in three steps; first of all it converts the code into an internal representation. After transformation it identifies those parts that denote clones and finally it combines the clone pairs into clone classes. For further details readers are referred to read [10]. Efficiency of this technique is supported by experiment which performs clones deduction in various programming code including C, C++ and Java. The author have concluded that it is fast and requires less memory as compare to other clone detection techniques. Although it can find type 3 clones from the source code but it is less efficient for type 2 clones.

Shinobi [11] is another useful and novel approach for the clone's detection and then the modification of the code clones. This technique was proposed by Kawaguchi *et al.* in 2oo9. The main features of this technique are to (1). Identify the clones from the source code and (2). Highlight these identified clones segments. This technique is very tightly integrated with Microsoft visual studio. It works in such a simple way that as it detects the code clones, it warns the developers about the clones, so that the developer can get read of it. Shinobi is a token-base detection method. It takes the source code automatically and then detects the clones. The source code may be from a CSV file or source file directory. This approach is very useful for clone detection while a developer is working on the software maintenance. The detail procedure of Shinobi is available in [11]. This work is supported by very strong arguments. These are (1) It can be implemented as add-in in Microsoft visual studio. (2) Its clone detection process is very fast and automatic. (3) It is very useful in clone detection while a developer is working in a software maintenance phase. Although, this technique is simple, fast and accurate it does not provide any kind of compatibility with other development environment like Java Netbeans, Oracle edition etc.

There are two types of code defects that may exist in any of the source code file. These two issues are rule violation defects and copy-and-paste problems. Zhang [12] *et al.* have suggested a model that copes with these two types of source defects. The authors have used the data mining technique for their approach. This technique is frequent pattern mining. It is not an easy task to detect the programming defects. The proposed approach is tested with a C source file having 4 million lines of code. This approach is implemented in C++ and Python. They perform various experiments. The results of these experiments clearly show the efficiency and accuracy of this technique. Since this approach can detects maximum programming defects in one phase that is why testing time is very much optimized with this method. The authors have implemented the data mining techniques effectively to identify and modify the programming defects. Although this technique works fine, they have not compared their experiment results with other techniques, including [2] and [1] etc.

III. Critical Evaluation

This section primarily reflects the comparison and contrast of the above reviewed literature regarding the different clone detection techniques. It identifies the similarities and differences among the various research works on the clone detection algorithms. The critical review is given in the Table-1(A), Table-1(B) and Table-1(C), below. This will help for the future research in the clone identification and deletion of the clone data.

The clone detection technique in [1] provided sufficient experimental details of the methods with implementation in C++ but the method [1] can detect clones only in one language. In contrary the work [2] also provided detailed experiments and with C++ implementation but the method [2] can detect clones in C and C++ source codes. The clone detection approach for java based source code is provided in. The authors in [4] and [8] both have provided with sufficient details but [8] is more accurate as it implements consist renaming approach in detected clone.

Researchers in [6], [7], [8], [9] and [10] proposed approaches for detection clones written in C language, these have provided sufficient details and the results are presented in tabular as well as graphically format except [8] and [9]. The author [6] in his future work planned to provide method which is more efficient and removal of detected clones. In contrary authors from [8] and [9] suggested in their future work to extend their approaches for detecting clones in source code written in more than one language, but the researcher in [10] has not discussed any future work. To detect clones written in .Net language the work [11] provided sufficient details but the future work is not discussed. The work [12] is Python based detection tool.

IV. Conclusion and Future Work

In this study, we have presented the summary information of the different clone detection techniques. These clone detection techniques are based on the CC-Finder, CP-Miner, Abstract Syntax Tree, and Frequent Itemset Technique. In addition, we also have highlighted the research contributions and found out some limitations in different research works. Consequently, this work also depicts the critical evaluation in which comparison and contrast have been taken out to show the similarities and differences among different author's works. The spatiality of this work is that it reveals the literature review of different clone detection techniques and provides a vast amount of information under a single paper. In our future work, we have planned to propose our own technique based clone detection technique, and provide its implementation and compare its results with the different existing clone detection algorithms.

References

[1] T. Kamiya, S. Kusumoto and K. Inoue, "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code", IEEE, 2002.

[2] Z. Li, S. Lu, S. Myagmar and Y. Zhou,"CP-Miner: Finding Copy-and-paste and Related Bugs in Large-Scale Software Code", IEEE, 2006.

[3] V. Wahler, D. Seipel, J. Wolff, V. Gudenberg and G. Fischer,"Clone Detection in Source Code by Frequent Itemset Technique".

[4] H. A. Basit and S. Jarzabek,"Detecting Higher-level Similarity Patterns in Programs", ACM, 2005.

[5] B. Hummel, E. Juergens, L. Heinemann and M. Conradt,"Index-Based Code Clone Detection: Incremental, Distributed, Scalable".

[6] I. Baxter, A.Yahin, L. moura, M. S. Anna and L. Bier,”Clone Detection Using Abstract Syntax Trees”ICSM, 2005, p3.
[7] R. Falke, P. Frenzel and R. Koschke, “Clone Detection Using Abstract Syntax suffix Trees”.
[8] P. Jablonski and D. Hou,” CReN*: A Tool for Tracking Copy-and-Paste Code Clones and Renaming Identifiers Consistently in the IDE”, ACM, 2007.
[9] S. Uchida,T. Kamiya,A. Monden,K. Matsumoto,N. Ohsugi and H. Kudo,”SOFTWARE ANALYSIS BY CODE CLONES IN OPEN SOURCE SOFTWARE”,2005.
[10] Y. Jia, D. Binkley, M. Harman, J. Krinke and Makoto Matsushita, “KClone: A Proposed Approach to Fast Precise Code Clone Detection”, 2009.
[11] S. Kawaguchi, T. Yamashina, H. Uwano, K. Fushida, Y. Kamei, M. Nagura and H. Lida, “SHINOBI: A Tool for Automatic Code Clone Detection in the IDE”, 2009.
[12] Y. Zhang, Y. Liu and L. Zhang, “A data mining based method: Detecting software defects in source code”, IEEE, 2010.

TABLE-1
CRITICAL EVALUATION OF VARIOUS CLONE DETECTION TECHNIQUES

Authors	Features	Experiment Detail	Methodology	Complexity	Future Work
Kamiya [1]	• It combines lexical and dependencies analysis in order to generate precise clones from the data.	• Sufficient experimental details are provided • They used data from multiple domains with various data size	Implementation (in C++)		They are trying to expand their tool to detect clone detection in multiple languages.
Z. Li [2] et al	Their approach uses data mining techniques to competently detect copy-and-pasted code in large software and CP-Miner is very efficient as compare to other clone detection methods like CC-Finder etc.	• They provide sufficient experiment details. • They used data in different operating systems.	• Implementation (in C and C++)		Not discussed
Wahler[3] et al	Their new approach is used to detect clones of types 1 and 2.	• They provide sufficient experiment detail • They used different packages of Java for clone detection.	• Implementation (in Java)		They are tried to extend their algorithm to detect clones of type 3.
Basit[4] *et al.*	• Their technique is based on token-base methodology for finding (a) simple clones (b) Finds co-occurring clones in a different files by using a frequent itemset mining method.	• • They provide experiment results. • They used java source code for analysis.	• Implementation (in C++)	•	• They discuss about a demanding feature that would be able to provide multiple language ability within the same system.

hummel[5] ***et al.***	• In this method source code is read from a disk and converted into tokens. • This method is useful for the efficient retrieval of clones from the source file.	• They provide their results experimentally.	Implementation(in C++)		They plan to develop algorithms that detect the clone of type 3 and also plan to extend their index base tool for plagiarism detection in text documents.
Baxter[6] ***et al***	Their proposed approach AST is the simple one as compared to the rest of the clone detecting techniques and efficient and can detect the accurate clones from the code as compare other clone detection techniques	• Experimental details are given • These experimental details are provided in tabular as well as in graphically	implementation(in C)		To automate the removal of code clones detected in a source file [6]. Increasing performance of the proposed method.
Falke[7] ***et al***	Their method is used to find clones in abstract syntax trees. Their method is very efficient and fast especially in token base clone detection for type-2.	• Experimental details are given • These experimental details are provided in tabular as well as in graphically	Implementation (in C)		To improve and uses a token counts instead of lines during the measurement of clone size.
Jablonski[8] ***et al***	Their proposed tool is implemented as an eclipse plug-in in Java and their tool is also used for renaming the identifiers just within copy-and-paste fragments	• Experimental details are given in tabular form	Implementation (in Java)		To generalize the proposed tool and provide consistent renaming support.
Uchida [9] ***et al***	They use [1] tool for detection of code clones and for identification they use a clone warrior tool.	• Experimental details are given in tabular form.	Implementation (in C)		To perform analysis on multiple languages.

Jia[10] ***et al***	The proposed approach is fast and requires less memory as compare to other code clones detecting tools.	• Experimental details are given • These experimental details are provided in tabular as well as in graphically	Implementation (in C)		Not Discussed
Kawaguchi [11] ***et al***	The proposed tool is token base clone detection and used for identifying and Highlight these clones. This method integrated in Microsoft visual studio.	Experimental details are provided graphically	Implementation (in .net)		Not Discussed
Zhang [12] ***et al***	The proposed approach is used to detect maximum programming defects in one phase that is why testing time is very much optimized with this method.	• Experimental details are given • These experimental details are provided in tabular as well as in graphically	Implementation (in C++ and Python)		To compare the proposed approach with other code clones detection tools like [1] and [2] etc.