Volume 8, No. 5, May-June 2017
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

Design Code Clone Detection System uses Optimal and Intelligence Technique based on Software Engineering

Jasmandeep Kaur
Research (Scholar)
Department of CSE, SECG, Gharuan, India

Rakesh Kumar
Principal and Associate Professor
SECG, Gharuan, India

Sukhjot Kaur
Assistant Professor
Department of CSE, SECG, Gharuan, India

Abstract: Code clones are the main source of cloned software. Now, a day's redundancy in initial code is called clones or duplicate code caused by copy and paste, could search consistently using code clone detection software tools. The redundancy could arise also individually, although, not produced by copy and paste. Recently, it is not clear to define how the only Metric approach (functionally same clones) dissimilar from duplicates made by copy and paste. In this paper, our idea is to understand and classify the syntactical dissimilates in (function same clones) Metric based technique used with the help of swarm and artificial intelligence techniques that described them from copy and paste code clones in a path that helps clone detection research. In this method, we discussed it by functionally using the same program in Java, C++, and MATLAB for coding challenges. We studied syntactic correspondence with new detection software tools and discovered whether code clone detection can perform outside other structure. We implement the metric based approach extract the code properties i.e. LOC, Function Overloading, Function Repetition, Total number of functions, Global and Local Variable with the help of PDG and AST tree code clone techniques. The Classification with Neural Network approach to classified the code clone and calculates percentage of the code clone as compared to original code. We executed all tools on 100 programs and manually classified the dissimilarity in a random code sample of 60 programs files. We search non-metric function similar codes, where complete records were syntax and syntactically same code. The major difference between metric and crossbreed algorithm code clone detection techniques are beyond the recent code clone detection approaches.

Keywords: Code Clone, Detection Software Tool, Metric Tool, Abstract structure tree, swarm and artificial intelligence approach.

1. INTRODUCTION

Code clone have been verified as a main source of defects, which means that duplicating the code could be a sustainable issue during implementation and maintenance. The result, a huge part of the research has been implemented how to prevent or spot and change code clones. The main issue with code clones is associated only with their similar code that is indirectly rather than directly which creates it problematic to identify them. Although, modifies like updates or covers that are often meant to affect every clone in the same path, are normally not functional to all of them consistent. The code quality declines and modification becomes more expensive and error-prone [1]. A code part normally defines due to replication from one place and then re-writes them into the additional section of code without and with modifications/changes is software cloning and same code copied are called clones. Several analyses have conveyed more than 20 to 59 percent code duplication [2]. An issue with such same copied code is that an exception detected in the real or original must be verified in each copy for the similar error [3]. The copied code developed the efforts to be done with the parameter code. The quality of code analysis, virus recognize, facet mining and error exposure are the other tasks which need the knowledge of syntactically verified code part to facilitate code detection importance for software detection tool analysis.
In this research paper, we have provided a comparative survey on recently accessible clone detection techniques and software tools. We will initialize with the introduction of code clones after that categories and compare the approaches and tools in binary dissimilar paths. The classification of code clone types, techniques and categorization of the code detection tools [4].
The section 1 presents the introduction and issues found in the code clone detection approach. Section 2 is related to several clone detection techniques. After that classification approach is explained to classify the different types of clones. The types of clones are based on the texture similarity and functional similarity. In the next section proposed work is discussed to Analyse code clone software methodology. In this section algorithms are used to detect the clone from programs or files. In Section 5 Results are shown with the help of tables and also compare the existing and proposed work and in last section conclusion and future scope is discussed.

2. RELATED WORK

There are numerous techniques for detecting code clones which are discussed in the literature survey. Each technique has its own benefits and limitations. While the text-based techniques provide the easiest way of detecting code clones, but they can detect only type-1 clones. Though the token-based techniques can detect both type-1 and type-2 clones, but these techniques need a lexical analyzer to transform the code into tokens. A significant amount of time is consumed in tokenization. In AST based techniques, it is required to parse the source code which is a time and space consuming process. PDG based techniques can find near-miss clones,

but these techniques take a huge amount of time and are very complex. To convert a program into its PDG representation, both its data flow graph and control flow graph are required. Metrics based techniques are complex because they only require comparison of some numerical data, i.e. metrics values of program units to find code clones. But these techniques may give false positives and result in less precision value [5]. Code clone describes of two segments of code that are same according to the description of similarity. The normally, code clone detection approaches define for extract clone is called Type-1 and copied code with individual changes such as re-name is called Type -2. This kind of clones is detectable in recent years an effective and efficient way.

Table 1. Detect kinds of Code Clones

Types	**Description**
Type -1 Clone	Whitespaces added
Type -2 Clone	Variable and literals changes
Type -3 clone	Use functions
Type -4 clone	Different logic and output will same.
Functional Same Clone	Functionality same
Solution set	Set of solution records all resolving the similar program issues
Solution Records	Individual program in single experimenting the solution to a programming issue.

Even the clones with extra modification could be found by numerous detection techniques and tools. The survey of a systematic approach and analyzed in single type 3 clones and their dissimilar. The main focus, however on the difference in code metrics, variable and hided them only type substitution.

Komondoor et al. (2001) [6] author investigates the duplicate code from a software system with slicing technique. Duplicate modules in a software system are a normal thing. But it increases the software maintenance cost and efforts for stable a software system in production mode. The proposed approach detects all the similar clones and converted into a single module. That single module called for all the places to reduce duplicated code from the modules. This approach working with some graphs technique which helps to represent clone from a software system with the help of similar sub-graphs. Toshihiro et al. (2002) [7] developed a novel code clone detection method, which contains the transformation of textual information as input on source end and token to token compare. Various types of optimization techniques implemented or developed the tool, namely CC Finder, which detects code cloned in C++, JAVA, C and other source files. The design of a metric based technique in code clones**.** Roy et al. (2009) [8] proposed SOTA in code clone detection approaches and tools and manage the huge amount of information into a comprehensible conceptual structure. They initialize with related concepts, a unique code detection procedure, and overall current tools and methods. Then classify, compare and evaluate the methods and tools in binary dissimilar dimensions. Garg et al. (2013) [9] code obfuscation for security enhancement is the main objective of this research. An example of this is Java byte codes which are a form of processed code but with the use of reverse engineering, this code can be recovered**.** Kodhai et al. (2013) [10] in this research the Clone Manager is used to detect the clones from the software modules. The author used some kind of unified approach to enhance the working of Clone manager tools**.** Rattan et al. (2013) [11] there are dissimilar situations where clone detection is vital. Some of the capacities are: feature mining, to find the cross cutting code, plagiarism detection, software product-lines, clones in websites, origin analysis, quality assessment, detecting licensing violations. Though these areas are autonomous research areas, yet these areas and clone detection can get promoted from each other. By representing the code in an abstract depiction like PDG, existing code clone detection tools may be modified to detect hidden variations in the code. Clone detection helps in sensing shared and common set of features in software invention lines**.** Bansal et al. (2014) [12] author did their research in clone detection from the large coding modules. Here the main problem discussed in this work is time-consuming and understanding and working complexity of detection tools. The clones are copied code pasted around the large modules of software without any change so that they cause high maintenance cost and software faults. Rao et al. (2014) [13] explain the process of recycling of software components for faster development of large scale software systems. In the large scale system the code development is depends upon various languages to handle front middle and backend views. Wagner et al. (2016) [14] conducted a result using known functionally same programs in Java and C from coding matches. They studied syntactic similarity with traditional detection tools and searched whether con-colic clone detection could go beyond syntax.

3. CLASSIFICATION OF CODE CLONES

It could be classified on the basis of tri-aspects which are described below. Classify the clones, used for expansion re-engineering and detect approaches. We have re-iterated on the major prominent kind of clone, which prevents at the quality of time interval re-engineering. Following are the various code clones based on tri-aspects i.e. a) Similarities b/w binary code parts. b) Object code location in program. c) Re-factor chances with the simulated code [15].

The similarity-based fragments are the majority of binary kinds i.e., i) Binary code part could be verified on the basis

of the same code of their execute program data [16] ii) It could be same in their functionalities without being texture verification. However, texture similarity based clones are of four kinds as type-1, type-2, type-3, and type-4. An instance section the methods which are similar except the name and the techniques which are verified for the kinds of performance parameters integrated with larger similarity code clones. The type-4 code clone is based on the same functionalities, same output but different logics designed. The comparisons [17] between binary methods are of three types which are based on four key points of compared such as method name, deign of code, a method in lexis and flow control of the methods. The methods of clones described the dissimilar classification which recognizes each group of code clone functions on the basis of previous dissimilarity between them. The classification define the quality of methods of content has been copied same and also what kind of syntax tree elements have been changed.

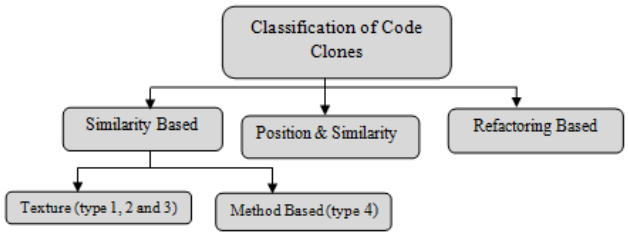

Figure 1. Classification of Code Clones

The second select instance, the literal variations and function aspects based on three types. The third stage is based on the important if the single literal or token in the method block and moreover the 4th stage is defined that the token sequence description in method blocks. The three types of clones such as extract fragments, argument clones, and clones which have other pervasive characteristics [18].

Several research pro-types were not available or cannot be brought to execute. Numerous simulation tools were not added in the analyses due to their lower performance and scalability / their decrease in support for some clone-types [19, 20]. Clone-DR and CPMiner have less performance and scalability evaluated to Deckard. CCFinder has less performance than Deckard and doesn't provision type-3 clones. At last, they select binary clone detection simulation tools that together could study JAVA and C programs: ConQAT and Deckard. Con-QAT is discussed in as newest, useful and speedily open-source clone detector structure or framework. In the analyses, they determined above, Deckard has defined to have better performance and scalability. They both are well described and have been used in prior studies, especially. At the time of the analyses, those were binary tools which were both freely available and possible to create them work for us.

Con-QAT is a steady, free, open-source dash-board tool-kit also used in industry. It is normal aim simulation tool for several kinds of code measurement and analysis study. Con-QAT, gives various specific code clone detection configurations for several programming languages, adding JAVA, C/C++, and COBOL. It has divide detection methods for Type-1 or Type-2 clones and Type-3 clones. They employed the previous method. Con-QAT has been described in various analyses in clone detection adding the study, they construct on [21].

Deckard uses an effective method for verifying same sub-trees, and applies it to tree re-presentations of source code. It normally generates a parse-tree constructor to construct parse-trees required by its method. By a same parameter, it is possible to control whether only Type-1, Type-2 clones and Type-3 clones are detected. Deckard is a suitable tool described in-other analyses in adding the study, we construct on [22].

Table 2. Tool of Code Clones

Tools Compared	**Techniques**
CC Finder [19]	Suffix Tree and Token
Clone DR	Abstract Syntax Tree, Hashing
Covet	Abstract Syntax Tree and Metrics
Duploc	Texture, Substring Matching
Dup [23]	Text, Token and suffix Tree

Table 3. Comparative study of the Code Clone detection techniques

Transformation /Normalizations	**Source Code Representation**	**Clone Matching Technique**
Program Dependence Graph[24]	Find Grained	n-length patch matching
Suffer code to PDG	PDG	Dependence Graph sub-graph comparing using program slicing
Token and Comment Removal	Text	Vector defined in LSI [25]

4. PROPOSED MODEL

In this section, we discussed the proposed model with key point and phases.

4.1 Proposed Key-points

- To study and analysis various techniques and algorithms of code clone detection and calculate vulnerabilities.
- To develop an embedded technique for BFO algorithm based on reduction lies.
- To implement a proposed algorithm for classification using back propagation neural network Model to do classification of code clones.
- To evaluate the performance parameters like false acceptance rate, false rejection rate, precision, recall and enhance the accuracy.

4.2 Explanation of Code Clone New Approach

Step 1: We search the data of the Java, C++ and MATLAB programming code files. We used the open access code files. We create a three programming files. Some data access in UCI machine learning repository dataset.

- Upload the dataset form the single language and show that data in list box UI-control tool used.

Step 2: Performing the Pre-processing and Feature extraction. We apply the metric based approach to calculate the Lines of Code, Number of repeating Function, method overloading, global and local variable. This process is known as Feature extraction which means to found the unique property of the programming files.

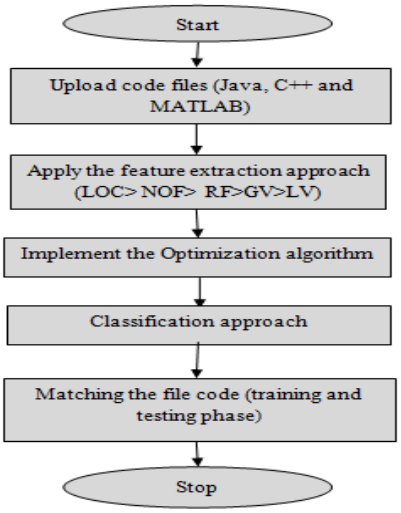

Figure 2. Proposed Flow chart

Step 3: Apply optimization technique to reduce the relevant features of the code files. We applied the Bacteria Foraging Optimization approach to reduce the properties of the program files. In this approach data set is generated based on the input information. The information move in two forms i.e. swimming and tumbling form i.e., faster and slower speed data moved.

Steps

a) Elimination and Dispersal

b) Reproduction of the information.

With the help of fit value generate the best output or given the result optimize as compared to other techniques.

Step 4: Classification approach i.e. to add the matching process. Back propagation Neural Network (BPNN) is used classified the programming files. In BPNN speed is faster as compared to other approaches as BPNN can handle the information one too many forms. It generates the two sections: i) Training Section ii) Testing Section. We create the training set, to train the MATLAB code clone files based on target. After that input is passing to testing phase, the simulation model used to compare the code file of training and testing section. After that match the LOC training set and testing set if match the Lines of code in same file then found the clone detect and calculate the percentage of the clone code. Calculate the performance parameters i.e. false acceptance rate, false rejection rate and accuracy. Compare with the existing performance parameters i.e. precision and recall.

5. SIMULATION MODEL

In this section, we discussed with the MATLAB 2013a in the simulation tool. We implement the code clone detection tool in graphical user interface. We add in the user interface in the GUI tool design a code clone interface and calculate the performance parameters i.e. FAR, FRR, Accuracy and Percentage Rate of the code clone. If similarity exists in code file then calculates the performance parameters and compared with it.

Table 4. Performance NN in Proposed Work

Number of Iterations	Hidden Neurons	Time	Best Performance (MSE)
27	10	0.1	39.83
24	8	0.02	34.56
20	5	0.001	25.67
21	5	0.001	24.32

Table 4 describes the number of iterations or epochs, hidden neuron used, time and base validation performance based on back propagation neural network.

Table 5. Performance Parameters in Proposed Work

FAR	FRR	Accuracy
0.00203	0.0078	99
0.00211	0.0067	99.1
0.00224	0.0056	98.6

Table 5 described the false acceptance rate, false rejection rate and accuracy performance parameters in the proposed work. If wrong information is acceptable then is called FAR and correct information is reject by mistake then is known as FRR. When FAR and FRR Value is decreasing then increase the accuracy.

Table 6. Percentage Rate with Precision and Recall

Percentage Rate	Precision	Recall
17.15	0.997	0.0991
56	0.995	0.992
66	0.987	0.983

Table 6 describes the percentage rate, precision and recall performance parameters in the proposed work. If the similarity code presents in training and testing phase, then calculate the 100% percentage rate.

Table 7. Comparison between Existing and Proposed Work

Accuracy in existing Approach	Accuracy in proposed Approach
83.8	97.8
93.18	98.5
98.78	99.8

Table 7 describes the comparison between existing and proposed approach. We improve the performance parameters with the BFOA and BPNN (Crossbreed Approach) and exiting one (NN).

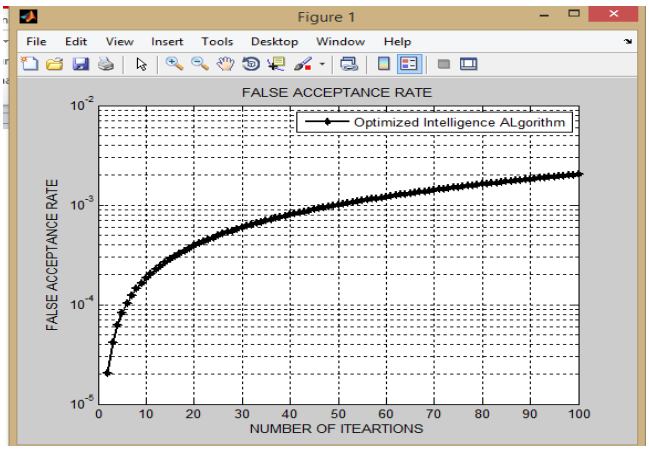

Figure 3. False Acceptance Rate (Proposed Work)

Figure 3 defined the false acceptance rate (FAR) means that incorrectly acceptable an access code from an unauthorized user. A Far Typically is defined that the ratio of the number of FAR divided by the number of verification attempts.

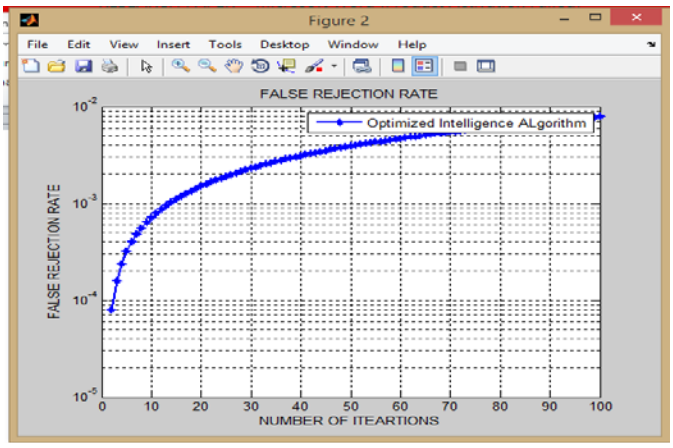

Figure. 4. False Rejection Rate

Figure 4 false rejection rate (FRR) shows that will incorrectly reject an access attempt by an authorized user/consumer. The systems FRR normally is stated as the ratio of the number of FRR divided by the number of identification attempts.

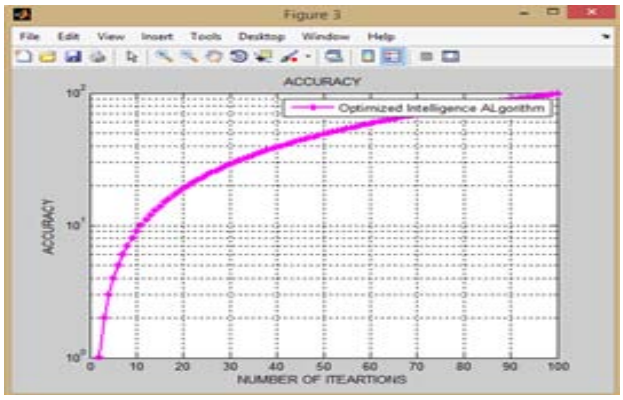

Figure 5. Accuracy

Figure 5 shows the best accuracy result. As False Acceptance Rate and False Rejection Rate decreases, Accuracy increases.

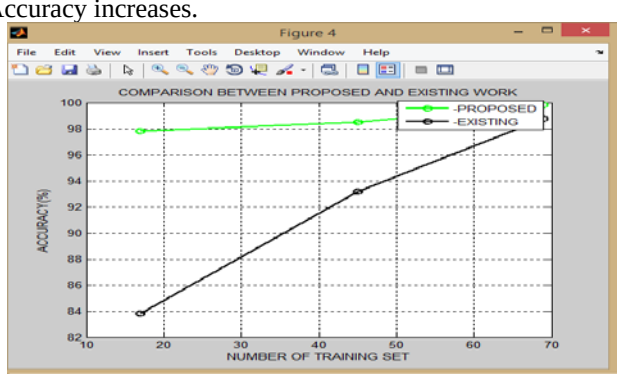

Figure 6. Comparison of Existing and Proposed Work (Accuracy)

Figure 6 defines the comparison between the existing and proposed performance parameters i.e. accuracy. We improve the performance parameters with the help of the new approach (ACO+BPNN)**.** The improvement with back propagation neural network accuracy for 30 instance number was calculated as 98.8%. In Artificial Neural Network accuracy for 30 instance number was evaluated as 90%. We improved the performance of the accuracy with back Propagation Neural Network.

6. CONCLUSION AND FUTURE SCOPE

The existence of code clones in a program enhancement is conservation cost as their existence makes the execution program complex and generates the issue of redundancy. The study of prior research work suggests the major focus of their research work on implementation approaches for detection of identified clones. In the current research study, the main focus is on the development of a noble approach to detect if same code blocks exist in any other file. The code clones have been clone detected in two phases:

- In the initial phase we implement the metric based approach to extract the features of LOC, Repeated Function and Function Overloading etc. After that, the BFOA algorithm is applied to optimize the feature set. BFOA algorithm is used two phase rotations
  - Tumble
  - Swim.

Tumble rotation means slowly reduce the feature set data and swim rotation means fastly reduce the extracted features. In BFOA, we can use the cost best solution function to identify best output in the form of 0's and 1's.

- In the second phase the classification approach (BPNN) is introduced to detect the clone in the code files based on training and testing section. The consequence of the second phase is to improve accuracy with the enhancement of several objects.

We increases accuracy with back propagation neural network for 30 instance number is from 90% to 98.8%. Using JAVA, C and C++ that gives graphical user interface, the complete tool is recommended for the classification of the input file as interchanging and non-interchanging targets. The title of record could be used for other executable programs to search the presence of clones.

In further research work, the proposed field would give a more generalized introduction of the code clone detection in an executable program.

REFERENCES

[1] Roy, C., Cordy, J., (2007), "A Survey on Software Clone Detection Research", Technical Report 2007-541, Queen's University at Kingston Ontario, Canada, p. 115.

[2] Koschke, R., (2007), "Survey of research on software clones, in: Duplication, Redundancy, and Similarity in Software," Dagstuhl Seminar Proceedings, p. 24.

[3] Kitchenham, B., Charters, S., (2007), " Guidelines for performing systematic literature reviews in software engineering." Technical Report EBSE-2007-01, School of Computer Science and Mathematics, Keele University, Keele and Department of Computer Science, University of Durham, Durham, UK, p. 65.

[4] Antoniol, G., Cassaza, G., Penta, M., Merlo (2001), Modeling clones evolution through time series, in: Proceedings of the 17th International Conference on Software Maintenance (ICSM '01), pp. 273–280.

[5] Bellon, S., et.al. (2007), Comparison and evaluation of clone detection tools. IEEE Transactions on Software Engineering vol33(9),pp.577-591.

[6] Komondoor, R., and Horwitz, S.,(2001), "Using slicing to identify duplication in source code."In International Static Analysis Symposium, pp. 40-56.Springer Berlin Heidelberg.

[7] Toshihiro, K., Kusumoto, S., and Inoue, K., (2002), "CCFinder: a multilinguistic token-based code clone detection system for large scale source code." IEEE Transactions on Software Engineering 28, no. 7 ,pp. 654-670.

[8] Roy, C.K., Cordy, J.,and Koschke, R., (2009), "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach." Science of Computer Programming 74, no. 7 ,pp. 470-495.

[9] Garg, V., Srivastava, A., and Mishra, A., (2013) , "Obscuring Mobile Agents by Source Code Obfuscation." International Journal of Computer Applications 61, no. 9.

[10] Kodhai, E., and Kanmani, S., (2013), "Method-Level code clone modification using refactoring techniques for clone maintenance." Advanced Computing 4, no. 2 (2013): 7.

[11] Rattan, D., Bhatia, R., and Singh, M., (2013), "Software clone detection: A systematic review." Information and Software Technology 55, no. 7 ,pp. 1165-1199.

[12] Bansal, G., and Tekchandani, R., (2014), "Selecting a set of appropriate metrics for detecting code clones." In Contemporary Computing (IC3), 2014 Seventh International Conference on, pp. 484-488. IEEE.

[13] Goda, G., and Damodaram, A., (2014), "An efficient software clone detection system based on the textual comparison of dynamic methods and metrics computation." International Journal of Computer Applications 86, no. 6.

[14] Wagner, S., Abdulkhaleq, A., Bogicevic, I., Ostberg, J., and Ramadani, J., (2016), "How are functionally similar code clones syntactically different? An empirical study and a benchmark." PeerJ Computer Science 2: e49.

[15] Göde, N., and Koschke, R., (2009), "Incremental clone detection."In Software Maintenance and Reengineering, 2009.CSMR'09. 13th European Conference on, pp. 219-228. IEEE.

[16] Zhenmin, L., Lu, S., Myagmar, S., and Zhou, Y., (2006), "CP-Miner: Finding copy-paste and related bugs in large-scale software code." IEEE Transactions on software Engineering 32, no. 3 ,pp. 176-192.

[17] Koschke, Rainer, RaimarFalke, and Pierre Frenzel. Clone detection using abstract syntax suffix trees. In Reverse Engineering, 2006.WCRE'06. 13th Working Conference on, pp. 253-262. IEEE, 2006.

[18] Yang, Y., and Guo, Y., (2012), "Boreas: an accurate and scalable token-based approach to code clone detection." In Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on, pp. 286-289. IEEE.

[19] Gabel, M., Jiang, L., and Su, Z., (2008), "Scalable detection of semantic clones."In Software Engineering, 2008.ICSE'08. ACM/IEEE 30th International Conference on, pp. 321-330. IEEE.

[20] Cory, K., and Godfrey, M.,. (2006), "" Cloning considered harmful" considered harmful." In Reverse Engineering, 2006.WCRE'06. 13th Working Conference on, pp. 19-28. IEEE.

[21] Denis, K., Koskinen, J., Sakkinen, M., and Markkula, J., (2010), "Exploratory analysis of the relations between code cloning and open source software quality." In Quality of Information and Communications Technology (QUATIC), 2010 Seventh International Conference on the, pp. 358-363. IEEE.

[22] Kim, M., Sazawal, V., Notkin, D., and Murphy, M., (2005), "An empirical study of code clone genealogies." In ACM SIGSOFT Software Engineering Notes, vol. 30, no. 5, pp. 187-196.ACM.

[23] Kim, H., Jung, Y., Kim, S., and Yi, K., (2011), "MeCC: memory comparison-based clone detector." In Software Engineering (ICSE), 2011 33rd International Conference on, pp. 301-310.IEEE.

[24] Mubarak, A., and Sulaiman, S., (2014), "A hybrid technique in pre-processing and transformation process for code clone detection." In Software Engineering Conference (MySEC), 2014 8th Malaysian, pp. 102-107.IEEE.

[25] Bari, M.A., and Ahamad, S., (2011), "Code Cloning: The Analysis, Detection and Removal." International Journal of Computer Applications 20, no. 7 : 34-38.

