Study of Clone Code Detection Method

Yafang Wang [a], Dongsheng Liu [b], Min Hou [c]

College of Computer Science and Technology Inner Mongolia Normal University Hohhot, China

[a]15540931820@163.com, [b]lds@imnu.edu.cn, [c]houmin920@163.com

Abstract. In the process of software development and maintenance, developers often use "copy-paste" or use the development framework, so that a large number of clone codes appear in the software system. In order to eliminate clone code and reduce the negative effects of clone code, researchers have proposed many excellent methods for clone code detection. This paper first introduces the significance of the research on clone code detection, and sorts out the related concepts of clone code detection. Then, according to the clone code detection technology, the existing techniques can be categorized to Text-based, Token-based, Tree-based, Metric-based and Graph-based categories. The five categories, respectively introduce the corresponding existing methods or tools, and summarize their advantages and disadvantages. Finally summarize the key problems in the current clone code detection research.

Keywords: clone code, clone detection, detection method.

1. Introduction

Copying code fragments and then reusing them by pasting and adjusting (such as adding, deleting, or modifying statements) is a common practice in software development, and this reuse mechanism can result in a large number of identical or similar code fragments in the code repository. This type of code segment is called Clone Code, also known as Duplicate Code. In addition, using a particular development framework or reusing design patterns can also result in clone code. Empirical studies have shown that a software system may have 20% ~ 30% clone codes[1][2][3][4], and sometimes even up to 50%[5]. A large number of clone codes can have a negative impact on the software system[6]. For example, reusing code fragments containing unknown errors may lead to error propagation and reduce the reliability of the software system. In addition, without good clone management of the software system, the code repository will continue to expand, increasing the cost of maintenance. Therefore, it is necessary to perform regular clone detection, management and maintenance of the software system, thereby reducing the extra labor and reducing the maintenance cost of the software.

The code clone classification standard widely accepted by researchers is a four-category standard proposed by Bellon et al. according to the degree of code cloning[7], that is, the identical code fragment (Type-1), renamed code fragment (Type-2), almost the same code fragment (Type-3) and semantically similar code fragment (Type-4), from Type-1 to Type-4, the similarity of code fragments is inversely proportional to the detection difficulty, that is, the similarity between code fragments gradually decreases and the detection difficulty gradually increases. Scholars have put forward many excellent clone detection methods and tools[8][9][10], but the complex clone situation cannot be effectively detected. This paper in view of the current code clone detection research progress combing the existing methods, according to the method used, it is divided into Text-based method, Token-based method, Tree-based method, Metric-based method and Graph-based method, etc. Summarize the existing clone code detection technology and evaluation methods, conclusion the current key problems in the field, and discuss the solution ideas and research trends.

2. Clone Detection Related Concepts

2.1 Basic Concepts of Code Clone

Code clone is also called duplicate code (referred to as clone). For the convenience of description, this paper defines several basic concepts as follows:

- Clone Fragement (CF): clone fragement is a special part of the source code that consists of a number of statements, which can be a classe, a function or a method, a sequence of declarations, and so on.
- Clone Pair: when a code fragment (CF1) and another code fragment (CF2) are identical or similar to each other exceeds a certain threshold(using a similarity calculation method), the two code fragments are called clone pair (CF1, CF2). Clone relationship are generally reflexive and symmetrical.
- Clone Class: In a series of code fragments, any two code fragments are clone pair, which are called a clone group.

2.2 Clone Code Classification

There is no uniform clone code classification standard in the field of clone code. The most widely used is the four-category standard proposed by Bellon et al. according to the degree of clone code[7]:

Type-1(identical code fragment): a code pair that removes spaces and comments exactly the same.

Type-2 (renamed/parameterized code): code pairs that are identical except for Type names, identifiers, and constants.

Type-3 (almost the same code): a pair of code where some statements are added, deleted, modified, or where identifiers and types are replaced, but the syntactic structure is basically the same.

Type-4 (semanticly similar code): a pair of code that is semantically similar but has a different syntactic structure.

Although this classification reflects the degree of similarity between code fragements, there is no strict definition of type-3 and type-4 clone code. For example, how many statements can a type-3 clone delete? Therefore, the standard is not rigorous. In addition, scholars have also studied other types of clone code, such as structural clone [11][12], function clone[13] and model clone[14].

2.3 General Process of Clone Code Detection

There are a lot of differences between the existing clone code detection methods, but their basic processes are the same, including four steps: source code preprocessing, intermediate representation of code, code similarity comparison, and report detection results. Source code preprocessing mainly includes removing spaces, comments and redundant statements, and selecting appropriate source code units according to requirements. These units generally have files, classes, functions, or methods, etc. The intermediate representation of the code is to abstract the text, symbols, or convert the source code into an abstract syntax tree, etc. Code similarity comparison refers to the similarity detection between each intermediate code fragment and other code fragments to find the code clone fragments. Reporting detection results means storing detected clone codes in an appropriate manner and providing detection reports to researchers.

2.4 Evaluation Indicator of Clone Code Detection Method

Different code cloning detection methods are applicable to software systems with different scales, programming languages and structures. In order to evaluate the detection methods, the following evaluation indicators are generally adopted[15][16]:

Recall: the ratio of all detected clone codes to the total number of clones.

$$Recall = TP / (TP + FP) \tag{1}$$

Precision: refers to the percentage of candidate clone codes detected by the clone detection algorithm that are true code clones.

$$Precision = TP / (TP + FN) \tag{2}$$

Where: TP represents the intersection of the cloned fragment detected by a certain clone code detection method and the real code clone fragment. FP represents the collection of clone code

fragments that are detected to be clone fragments by using the clone detection method, but are not actually clone. FN represents a collection of real clone code fragments that are not detected by the detection method.

Clone type: the ability to detect type-1 to type-4 code clones is an important indicator of evaluation code clone detection tools or methods.

Results presentation: generally, the detection results are presented by clone pair or clone class. Since clone class contains the information of clone pair, the clone group can provide more detailed information of detection results[17].

2.5 Clone Code Detection Method

The method used in clone code detection directly determines the efficiency of the method, the type of clone that can be detected and the accuracy of the final results. For example, if the source code is cloned and detected in the form of text, the required space-time complexity is low. Theoretically, only the clone codes of type-1 and part of type-2 can be detected. However, if the source code is cloned and detected in a graph, although the semantically similar code clone can be detected, the graph-based graph matching technique requires a huge computational overhead. Therefore, using different ways to represent source code for detection is the key to clone code detection.

2.6 Research on Text-based Detection Method

The Text-based code clone detection method directly detects the code clone by using the text similarity detection algorithm by the source code of the pre-processing (delete spaces, comments, etc.). Since the source code itself has meaning, directly processing it as text will lose a lot of information, so the Text-based clone code detection method can only detect type-1 and type-2 clone codes. At present, mainstream Text-based clone code detection methods mainly include NiCad[10], SDD[18], Dup[19], Duploc[20] and so on.

NiCad is a clone code detection tool proposed by Roy's team, which is widely used in academic circles. NiCad first preprocesses the source code, removes whitespace and comments, splits the code fragement into lines using a parser. Then converts the code according to the established rules. Finally uses the dynamic matching pattern to find the longest common subsequence (LCS). NiCad uses percentage of unique strings (PUS) to control the detection slack, so that type-1, type-2, and type-3 clone codes can be detected. NiCad can detect clone code in block and function granularity, with high recall and accuracy[15], and the final clone information is fed back in the form of clone pair and clone class.

SDD is a method proposed by Lee et al. for efficient detection of clone code of large software systems, which has been packaged as an Eclipse plug-in. SDD directly establishes an inverted index on the source code, and uses n-nearest neighbor algorithm to detection, it can detect type-1 to type-3 clone code. Finally, the clone code pair are displayed in a visual form.

Dup first removes the useless information such as whitespace and comments in the source code. Then converts all variables in the source code into strings, that is, parameterized strings. Finally, uses the pattern matching algorithm based on suffix tree to detect the parameterized strings. This text method needs to be implemented by means of a lexical analyzer, and only type-1 and type-2 clone code can be detected, and the final result is presented in the form of a clone pair.

Duploc first removes the comments and whitespace in the code, characterizes the lines of code in the text, hashes the string of each line, and visualizes the difference using a dot plot. Then use a simple string method to detect the longest common subsequence. Duploc can detect type-1 and type-3 clone code, and display clone code information in the form of clone pair.

The Text-based clone code detection method directly compares the source code text, and generally does not detect clone code with large text differences, so it has a high accuracy. Since only the text structure of the source code is considered, there is no need to use a parser, so this detection method can be easily extended to multiple programming languages. Compared with the matching algorithm based on graph or tree, this method has lower time-space complexity, so it is suitable for the detection

of large-scale software systems. the Text-based detection method ignores the syntactic and semantic information of the code and can only detect the relatively simple clone code.

2.7 Research on Token-based Detection Method

The Token-based code clone detection method uses a lexical analyzer to divide the source code into Token sequences and then find similar subsequences in the Token sequence. The Token-based detection method performs lexical analysis on the source code. The symbol sequence is more consistent with the compilation principle, and the source code information is more fully utilized. However, it lacks the analysis of the syntax and semantics of the code, and the detection effect of type-3 and type-4 cloning is not ideal. At present, the Token-based code clone detection method mainly include CCFinder[4], CP-Miner[21], CCAligner[22], and CCLearner[23].

CCFinder was proposed by Kamiya et al., and on this basis, D-CCFinder[24] was proposed. CCFinder first uses the lexical analyzer to preprocess the source code, including removing comments and whitespace. Then performing parameter conversion according to special rules, representing the source code as parameterized symbols. Finally using the suffix tree matching algorithm for code cloning detection, that can detecte type-1 and type-2 clone code. The final result can be displayed in both clone pair and cloned class.

CP-Miner is a Token-based clone code detection method proposed by Li et al., which is suitable for detecte large software systems. CP-Miner first parses the source code into a sequence set, and characterizes the source code in code block manner. Then using frequent sub-item mining technique to get similar code fragment information. Finally, the end result is obtained by checking and filtering. CP-Miner can detect type-1 and type-2 clone with higher accuracy than CCFinder.

CCAligner first removes extraneous information such as comments and spaces from the source code. Then use a sliding window to hash the source code. Finally, clone code information is obtained by hash comparison algorithm and setting appropriate threshold. CCAligner can detect type-1 to type-3 clone, especially when the line number between the two clone fragments of type-3 is greatly different.

CCLearner is a Token-based code clone detection method that uses deep learning. Li et al. used BigCloneBench as a training set, used a fully connected neural network to train the marked clone pair of data sets, and uses the learned feature to detecte software. CCLearner can detect type-1, type-2, type-3, and type-4 clone, and present the final result with clone pair.

Compared with the Text-based detection method, the Token-based detection method also serializes the source code, but it makes better use of code and performs better in detect type-2 clone. The Token-based detection method is sensitive to the occurrence of situation of adding, deleting, changing and changing the order of code lines in the source code, and is prone to missed detection.

2.8 Research on Tree-based Detection Method

The tree-based code clone detection method is to represent the source code as an abstract syntax tree (AST) or a code parse tree, and then use a tree matching algorithm to find similar or identical subtrees, so as to detect clone code. This method performs syntax analysis on the source code, and the utilization of the source code information is further improved, which can better detect type-3 clone and improve the detection accuracy. CloneDR[8], Deckard[25], CDLH[26], CloneDigger[27] and other Tree-based clone code detection methods are widely used at present.

CloneDR uses the parser to parse the source code into an abstract syntax tree. The tree matching technique and hash algorithm are used to divide the tree into several subtrees and search for similar subtrees. The code fragments corresponding to the founded subtrees are clone code. It can detect type-1 and type-2 clone, and the detection results were displayed in the form of clone pair. CloneDR not only detects clone code in software, but also can refactors source code with an abstract syntax tree.

Deckard first parses the source code into a code parse tree. Then converts it into a vector, clusters it using the local sensitive hash matching algorithm (LSH), and calculates the similarity between certain vectors using Euclidean distance. Finally, the similar vector is reconverted into the detected

code clone fragment, and display the result in a text format. Deckard can detect type-1, type-2, and type-3 clone, including changing the code line, but the detection speed is not ideal.

CDLH is a method proposed by Wei et al. that use deep learning techniques to detect clone code. CDLH first converts the source code into an abstract syntax tree and hashes it. Then uses the hashed code as training data and inputs it into the convolutional neural network for training. Finally, using the learned feature to detected software system and show the test results in the form of clone pair. This method can detect type-1 to type-4 clone code with high recall and accuracy.

CloneDigger detects clone code by representing XML as an abstract syntax tree and then looking for the same or similar subtree. This method can detect type-1, type-2, and type-3 clone code, but the effect of clone detection for type-2 is not satisfactory.

The tree-based clone detection method considers the syntax structure of the code, and makes more use of the code information than the text-based. Token-based detection method, which improves the accuracy of code detection and can better support the detection of type-3 clone code. However, the tree matching algorithm is complex and requires a lot of time, so the speed of the algorithm is lower than that of the first two methods and cannot be used on large software.

2.9 Research on Metric -based Detection Method

The Metric-based clone code detection method extracts the source code specific index metrics (such as the number of lines of code, the number of variables, the number of loops), abstracts them into feature vectors, and then determines clone based on the distance between the feature vectors, this method has a great advantage in speed. Currently, metric-based clone code detection methods include the work of Mayrand et al.[28], the work of Kontogiannis et al.[29], the work of Raheja et al.[30], and the like.

The work of Mayrand et al. first parses the source code into an abstract syntax tree, and extracts some indicator metrics (function names, expressions, hierarchies, etc.) from the source code. If the index metrics of the two code fragments are similar, they are considered to be clone of each other. This method can detect type-1, type-2, and type-3 clone, and the final results are displayed in the form of clone pair and clonal class.

The work of Kontogiannis et al. first converts the indicators of the source code into feature vectors. Then uses the dynamic program and the edit distance to calculate the similarity between the two codes. Finally, the preliminary test results are given, and users can further screen the results according to their own needs. The code clone information detected by this method is not necessarily correct and needs to be further checked manually.

Raheja et al. work to detect clone code that exist in the JAVA projects. The method uses bytecode to detect and map with the source code to find semantically similar code clone fragments, which is expensive to deploy. The method proposed by Raheja et al. can detect type-1, type-2, and type-3 clone code, and the final clone information is presented in an Excel table.

The metric-based clone detection method has high detection efficiency, but is limited to clone detection with only fixed granularity. Since the method is highly dependent on the extracted feature values, the semantic information possessed by part of the source code will be lost. For example, in some cases, although the two code segments have the same indicator, they are not clone.

2.10 Research on Graph -based Detection Method

The Graph-based clone code detection method converts the source code into a program dependency graph (PDG) composed of data flow graph and control flow graph, and implements clone detection by finding a homogeneous subgraph. The graph-based detection method exploits the semantic information of the code, so it can type-4 clone. At present, the code-based clone detection tools mainly include Duplix[9], Komondoor et al.[31] and GPLAG[32].

Duplix is a Graph-based clone code detection method proposed by Krinke. This method first converts the source code into a program dependency graph, and uses the K-length patch algorithm to search similar subgraph, which can detect the type-1, type-2, type-3 and type-4 clone, and finally displays the detection results in the form of clone class.

The main idea of Komondoor's work is to use program slicing to find isomorphic subgraphs in program dependency graphs. It first uses CodeSurfer to generate a program dependency graph. Then uses the isomorphic program dependency graph subgraph matching method to detect clone code. Finally, all clone pairs are clustered to obtain the final result, and the detection results are presented in the form of clone pair and clone class. This method takes into account the semantic characteristics of the code, so type-1 to type-4 clone code can be detected.

The GPLAG method also uses the isomorphic program dependency graph subgraph matching method. Since the time-space complexity required for graph matching detection is high, this method is only applicable to small data sets, and type-1, type-2, type-3, and type-4 clone can be detected. Finally, display the detection results in the form of clone pairs.

The graph-based clone code detection method not only utilizes the syntax structure of source code, but also considers the semantic information of the source code to a certain extent, so this method can detect type-4 clone. However, due to the high space-time complexity of the program dependency graph generation algorithm and the isomorphism program dependency graph subgraph matching method, the graph-based code clone detection method cannot be applied to the clone code detection of large software systems.

3. Evaluation Method

Different clone code detection methods have their own advantages and disadvantages. For example, the Text-based clone detection method is fast, but it can only detect type-1 and type-2 clone code. The Graph-based clone detection method can detect type-1, type-2, type-3 and type-4 clone, but the complexity of time and space is high, and the computational overhead is large. Therefore, a scientific and systematic method is needed to evaluate these detection methods. The collection of objective and effective data sets is the key to the evaluation of detection methods. At present, scholars use annotation data and data generation to evaluate.

3.1 Annotation Data Method

The method of label data collection means that the researcher collects several software codes[7][33][34][35], and uses the code modification log, context and manual inspection to evaluate the detection method proposed by themselves. However, this method has some defects. When collecting code sets, the data collected by different researchers are not the same, the scale and structure of different software systems are also different, and the code base is small, so it is difficult to form an objective and unified evaluation standard. In order to solve this problem, Svajlenko et al. built a JAVA code set called BigCloneBench[36]. A total of 8 experts participated in the construction of this data set, consuming 216 hours, manually marking 6 million pairs of type-1 to type-4 clone code, and 260,000 pairs of negative samples, which contained 10 functions. In the following work, many researchers used this data set to evaluate their proposed methods[22][23][26], and promoted the research on clone detection. However, BigCloneBench, with its ten features, does not measure up to reality, and it is only evaluated as a method for detecting clone code in JAVA software systems, and does not meet the unified standard.

3.2 Data Generation Method

Roy's team proposed a variation insertion method[15] to evaluate the clone code detection method. This method is based on the variability test in software testing[37]. The main idea is to artificially generate type-1, type-2, type-3 clone code by inserting a piece of manually written code into the source code fragment. Then use these clone pairs as a data set to evaluate the effectiveness of the clone code detection tool. The data set using the data generation method can obtain an accurate recall rate, such as Zhang Jiujie et al. used this method in evaluating the FClones[38] detection effect. However, this method has some defects. The artificially constructed data set is different from the real data. It is not scientific to evaluate the detection tool, so it can only be used as an auxiliary test method.

4. Key Problems and Solutions

4.1 Key Problem

Clone code has been studied for more than 20 years, and the development of clone detection technology is relatively mature, but there are still some key problems that need to be solved urgently.

(1) Lack of Type-4 Clone Detection Methods

Now, there are many excellent clone code detection methods, which are mainly divided into five types of detection methods based on Text, Token, Tree, Metric and Graph, which can effectively detect type-1, type-2, and type-3 clone code and have been widely recognized by researchers. However, there are few scholars research the type-4 clone method, and there is a lack of corresponding detection tools. Type-4 clone code requires semantic similarity between two code snippets. Existing Text-based and Token-based clone detection methods can only detect textual and lexical similar clone pairs. Tree-based clone detection method can detect syntax similar clone pairs. The Graph-based clone code detection method considers part of the semantic information of the source code, and can detect partial semantically similar clone pairs, but the calculation cost of this method is large and the technology is complicated. It is not suitable for testing needs. Therefore, how to better detect type-4 clone needs further research.

(2) The Definition of Clone Code is Blurred

The currently widely accepted definition of clone code is proposed by Bellon et al., with strict definitions for type-1 and type-2, but only conceptual definitions for type-3 and type-4. Due to the ambiguity of clone code definitions, it is impossible to accurately label cloned fragments, which limits the subsequent development of clone code research and makes it impossible to uniformly evaluate the performance of clone detection methods or tools.

Study the Practicality of Clone Code

Existing clone code detection tools or methods mainly gives the detection result in the form of clone pair or clone class, and reports the location where the clone code appears, does not sort or evaluate the detection result according to the specific application target, nor the clone code test results applied to the subsequent maintenance and management process. Moreover, only a few tools, such as CloneDetective[39] and SHINOBI[40], were combined with the actual software development environment. Therefore, it is necessary to carry out in-depth research in combination with the actual development process, and combine clone code with reality.

4.2 Solution

In view of the key problems summarized above, this paper expounds the possible solutions and future development trends of the above problems from two aspects: clone code detection technology and engineering practice application.

(1) Combine Deep Learning and Achieve Technological Breakthroughs

In recent years, deep learning has become the most popular method of machine learning, which has been widely used in computer vision, speech recognition, natural language processing, and other commercial fields, and has achieved remarkable results. The principle of deep learning is to form a more abstract high-level representation of attribute categories or features by combining low-level features, so as to discover the distributed representation of data features. If deep learning is applied to clone code detection technology, it may solve the problem that detecte type-4 clone is difficult, and the computational cost of using tree or graph detection technology is large, and the performance of code cloning research is further improved. So far, some scholars have combined deep learning technology into code clone detection methods, which is effective for detecting type-4. However, how to better integrate these technologies into detection technology still needs further exploration and analysis.

(2) Contact the Actual Situation and Promote the Application of Clone Code

The current clone code detection technology has been able to detect type-1, type-2 and type-3 clone, but it is not fully applicable to the industry. Most testing tools or methods have their own limitations, and the results of clone testing are only presented in the form of clone pair or clone class,

which cannot meet the needs of the industry. Therefore, in the following work, it is necessary to consider the needs of the actual developer and the design in an interactive mode. In addition, we should communicate with industry insiders to keep abreast of their needs, establish effective cooperative relations, and actively seek out areas where products can be expanded.

5. Conclusion

Clone code has important implications for the development, maintenance, and management of software. Based on the current research status of clone code, this paper introduces the related concepts of clone detection. From the five aspects of text, Token, tree, metric and graph, this paper sorts and summarizes the current clone code detection technology, analyzes it. Discussed the key problem facing the current clone code research and proposed possible solutions and future trends.

Acknowledgements

This work is partially supported by the National Natural Science Foundation of China (61363017).

References

[1]. Chen, Wen Ke, B. Li and R. Gupta . “Code Compaction of Matching Single-Entry Multiple-Exit Regions.” Lecture Notes in Computer Science 2694, 2003, pp.401--417.

[2]. M. Kim, V. Sazawal and D. Notkin. “An empirical study of code clone genealogies.” In ACM SIGSOFT Software Engineering Notes, 2005, pp.187.

[3]. J-F Patenaude, Ettore Merlo, Michel Dagenais and Bruno Lague¨. “Extending software quality assessment techniques to Java systems.” International Workshop on Program Comprehension IEEE, 1999, pp. 49-56.

[4]. T. Kamiya, S. Kusumoto and K. Inoue. “CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code.” IEEE Transactions on Software Engineering 28.7, 2002, pp.654-670.

[5]. M. Rieger, Stéphane Ducasse and M. Lanza . “Insights into System-Wide Code Duplication.” Conference on Reverse Engineering IEEE, 2005, pp.100-109.

[6]. Chanchal Kumar Roy and James R Cordy. “A survey on software clone detection research.” Queen's School of Computing TR, 541(115), 2007, pp.64-68,

[7]. S. Bellon. "Comparison and evaluation of clone detection tools." IEEE Trans. Softw. Eng. 33, 2007, pp.9.

[8]. I D.Baxter, A. Yahin, L. Moura and S. Marcelo. "Clone detection using abstract syntax trees." Conference on Reverse Engineering IEEE, 2006.

[9]. J. Krinke. "Identifying similar code with program dependence graphs." Proc. of 8th Working Conference on Reverse Engineering, 2001 IEEE, 2001, pp. 301-309.

[10]. C. K. Roy and J. R. Cordy. "NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization." Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on IEEE, 2008.

[11]. H. A. Basit and S. Jarzabek . "A Data Mining Approach for Detecting Higher-Level Clones in Software." IEEE Transactions on Software Engineering 35.4, 2009, pp.497-514.

[12]. K. Verena, S. Wagner and R. Koschke . "Are There Functionally Similar Code Clones in Practice?." 2018.

[13]. C. K. Roy and J. R. Cordy . "An Empirical Study of Function Clones in Open Source Software." WCRE 2008, Proceedings of the 15th Working Conference on Reverse Engineering, Antwerp, Belgium, October 15-18, 2008 IEEE Computer Society, 2008, pp.81-90.

[14]. M. H. Alalfi, J. R. Cordy, T. R. Dean and S. Matthew. "Models are Code too: Near-miss Clone Detection for Simulink Models." IEEE International Conference on Software Maintenance IEEE, 2012, pp.295-304.

[15]. C. K. Roy and J. R. Cordy. "A Mutation/Injection-Based Automatic Framework for Evaluating Code Clone Detection Tools." C3S2E Conference ACM, 2008, pp.157-166.

[16]. Xiaohong SU, Fanlong ZHANG. A Survey for Management-Oriented Code Clone Research[J]. Chinese Journal of Computers, 2018, 41(03), pp.628-651.

[17]. S. Abdullah and K. Jugal. “A Survey of Software Clone Detection Techniques.” International Journal of Computer Applications, 2016: 137(10), pp.1-21.

[18]. S. Lee. "SDD : High Performance Code Clone Detection System for Large Scale Source Code." Companion to the Acm Sigplan Conference on Object-oriented Programming ACM, 2005, pp.140-141.

[19]. B. S. Baker. “On Finding Duplication and Near-Duplication in Large Software Systems.” On finding duplication and near-duplication in large softwaresystems. 1995, pp.86-95.

[20]. S. Ducasse, M. Rieger and S. Demeyer. "A Language Independent Approach for Detecting Duplicated Code." IEEE International Conference on Software Maintenance IEEE, 1999, pp. 109-118.

[21]. Z. Li, S. Lu, S. Myagmar and Yuanyuan Zhou. "CP-Miner: finding copy-paste and related bugs in large-scale software code." IEEE Transactions on Software Engineering 32.3, 2006, pp.176-192.

[22]. Pengcheng Wang. “CCAligner: A Token Based Large-Gap Clone Detector.” IEEE International Conference on Software Engineering. IEEE, 2018, pp.12.

[23]. L. Li, H. Feng, W. Zhuang N. Meng and R. Barbara. "CCLearner: A Deep Learning-Based Clone Detection Approach." 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME) IEEE Computer Society, 2017, pp. 249-260.

[24]. S. Livieri, Y. Higo, M. Matushita and K. Inoue. "Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder." Proc International Conference on Software Engineering 2007, pp.106-115.

[25]. L. Jiang, G. Misherghi, Z. Su and S. Glondu. "DECKARD: Scalable and Accurate Tree-based Detection of Code Clones*." null IEEE Computer Society, 2007, pp. 96-105.

[26]. Huihui Wei, Ming Li. “Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code” International Joint Conferences on Artificial Intelligence Organization. 2017, pp.3034-3040.

[27]. P. Bulychev and M. Minea. "Duplicate code detection using anti-unification." Proceedings of the Spring/Summer Young Researchers’ Colloquium on Software Engineering 2, 2008, pp.51-54.

[28]. J. Mayland. “Experiment on the automatic detection of function clones in a software system using metrics” Proc. 12th International Conference on Software Maintenance. IEEE, 1996, pp.244.

[29]. K. A. Kontogiannis, R. Demori, E. Merlo M. Galler and M. Bernstein. "Pattern matching for clone and concept detection." Automated Software Engineering 3.1-2, 1996, pp.77-108.

[30]. K. Raheja, R. Tekchandani. “An Emerging Approach towards Code Clone Detection: Metric Based Approach on Byte Code.”

[31]. R. Komondoor and S. Horwitz. "Using Slicing to Identify Duplication in Source Code." Proc of Sas Paris France, 2001, pp.40-56.

[32]. C. Liu, F. Chen, J. W. Han and Philip S. Yu. "GPLAG: Detection of software plagiarism by program dependence graph analysis." Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006 ACM, 2006.

[33]. F. V. Rysselberghe and S. Demeyer. "Evaluating clone detection techniques from a refactoring perspective." Proceedings. 19th International Conference on Automated Software Engineering, 2004. IEEE, 2004, pp.336-339.

[34]. S. Shafieian, Y. Zou. “Comparison of clone detection techniques” Technical report, Technical report, Queen, 2012.

[35]. C. K. Roy and J. R. Cordy. "Scenario-Based Comparison of Clone Detection Techniques." IEEE International Conference on Program Comprehension IEEE Computer Society, 2008.

[36]. J. Svajlenko and C. K. Roy. "Evaluating clone detection tools with BigCloneBench." 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME) IEEE Computer Society, 2015, pp.131-140.

[37]. J. H. Andrews and L. C. Briand and Y. Labiche. “Is Mutation an Appropriate Tool for Testing Experiments?” Is mutation an appropriate tool for testing experiments?. 2005.

[38]. J. J. Zhang, C. H. Wang and L. P. Zhang. “Clone code detection based on Levenshtein distance of token” Journal of Computer Applications, 2015, 35(12), pp.3536-3543.

[39]. E. Juergens, F. Deissenboeck and B. Hummel. "CloneDetective - A workbench for clone detection research." IEEE International Conference on Software Engineering IEEE, 2009, pp. 603-606.

[40]. S. Kawaguchi, T. Yamashina, H. Uwano and K. Fushida. "SHINOBI: A tool for automatic code clone detection in the IDE." 16th Working Conference on Reverse Engineering, WCRE 2009, 13-16 October 2009, Lille, France IEEE, 2009, pp. 313-314.