IJCSN International Journal of Computer Science and Network, Volume 3, Issue 2, April 2014
ISSN (Online) : 2277-5420 www.IJCSN.org
Impact Factor: 0.274

Review of Code Clone Detection

[1] Jyoti Khanna, [2] Rajvir Singh, [3] Ritu Garg

[1, 2, 3] CSE Department, DeenBandhuChhotu Ram University of Science and Technology,
Murthal, Haryana, India

Abstract - Now a days Copy and Paste of code fragments has been regularly practiced in development of software. Because of limitations of time and lack of knowledge programmers use this code strategy. This strategy is known as code cloning. Clones may cause many problems. Probability of errors and the maintenance cost is increased. Modification would be difficult because of clones. So it needs to detect clones and remove them.

Keywords - *Clone, Tokens, Abstract Syntax Tree, Count Matrix, Count Vector.*

1. Introduction

Many techniques are used in clone detection. Some techniques are discussed below:

the maintenance cost and probability of bugs. So clones needs to be detected and refactored. Many techniques have been purposed Generally redundancy is found out in software system. This redundancy is because of duplicate fragments **[1]**. And these fragments are termed as clones. Clones increase

for clone detection and are discussed below.

1.1 Types of clone **[2]**:

Code clones are divided into four types:

Type 1 or Exact clones: One fragment is exact copy of another fragment except of white spaces and comments.

Type 2 or Renamed clones: one fragment is exact copy of another fragment except of identifier names, spaces and comments.

Type 3 or Modified clones: One fragment is copied to another fragment except that some statements are added or deleted.

Type 4 or semantic clones: One fragment is not syntactical similar to another fragment but behavior of two fragments are similar.

2. Techniques of Code Clones Detection

Table 1: Techniques of clone detection

Text Based	Token Based	Tree Based	PDG Based
In text based technique program is considered as a sequence of statements. Two fragments are declared as clones if they have same sequence for particular statements.	In this approach programs are transformed into sequence of tokens and clones are declared on the basis of matching subsequence of tokens.	In this techniqueprograms are converted into syntax tree or parse tree with the help of parser of the specific language. Clones are detected using different techniques applying on tree.	In this technique program dependency graph is built which represent data flow & control flow of the program. Isomorphic matching techniques are applied for clone detection.
These techniques detect less false clones, low recall.	These techniques detect less false clones and low recall but higher as compared to text based techniques.	These techniques have low precision and low recall.	These techniques detect less false clones and high recall.
Scalability depends upon the method applied for these techniques.	Scalability is high	Scalability depends upon algorithm applied.	Scalability is low because graph matching techniques are costly.
Easy to implement, light weight technique.	Fewer Complexes.	To build tree is very complex.	Matching techniques are complex.
This technique is independent of language.	This technique is independent of language.	Parser used for transformation is language dependent.	This technique is independent of language.

3. Literature Review:

Ira et al. [3] has used tree based technique to find the clones in the code. Three subprocesses are used to detect clones. First algorithm which is the basic algorithm detects the subtree clones. Every subtree will be compared to another subtree. And clones will be accepted on the basis of similarity:

2*S/ (2*S+L+R)
S-no of Shared clones
L-no of different nodes in subtree1
R-no of different nodes in subtree2

The second algorithm is used to find out sequence of clones. e.g. if subtrees are detected as clones in a sequence by basic algorithm then rather than declaring them as individual clones they would be considered as single clone.

Third algorithm is used to find out more complex near-miss clones. This method visits the parents of the already detected clones and finds out whether parent is it near-miss clones **[3]**.

Using this technique clones are not only detected but refactored also. Limitation of this technique is this is unable to detect semantic clones.

Yong Yuan et al. [4] used token based approach to detect the clones named as Boreas. Boreas purposes a novel counting based method for characteristics matrices which describes the program segments distinctively in an effective manner to detect the clones. Boreas has introduced three terms counting environments (CE), count vectors (CV) and count matrix (CM) **[5]**.

CE describes the pattern of variable CE is divided into three stages. These stages are naïve counting stage, in statement counting stage and inter statement counting stage. Naive stage consists of used variables. In in-statement counting stage CE is used for the variables where variables are used as if-predicates, array subscript and where any operation is applied to the variables and where it is defined by expression with constants. In inter statement stage CE is used as the variable is in first level loop, second level loop or deeper level loop.

CV with m dimensions is formed by using m CE. The ith dimension of CV consists of no of counts of that variable in ith CE. CV is also known as characteristics vector.

For n variable using m-dimensional CV m*n Count Matrix is formed. These CM are the abstract form of code fragments. Similarity of two fragments is measured by production of their CM's similarity and similarity of their CVs of the keyword and punctuation marks. Similarity of two CVs is measured with the formula:

Prosim = 1/ (a + 1) + b / (a + 1)

Boreas is language independent and high scalable with high speed means with less set up time and with less comparison time.

Rochelle et al. [6] detects semantic clone using IOE behavior. Semantic clones are also called by behavioral clones **[7]**. To detect clones input output is includedas in the **[8]** and their effects are also considered i.e. change in heap state is also included. This approach works at java code. In input value of parameters passed to function and heap's state at invocation time of method. In output, behavior is determined with the returned value of the function, possible and persistent change into heap. This Method includes four subprocesses: abstraction, filtering, testing and collection.

In abstraction AST are formed to obtain list of two method properties: method types and effects. In Filtering two filters are used. First filters take method type i.e. syntactical information as input and returns methods having equivalent return type and parameters as candidate clones. Second filter take semantic information as i/p and refine candidate clones by eliminating the function with different effects. Now after filtering, methods in any equivalence class having same type and effects. In testing phase dynamic behavior of the method is checked. The dynamic behavior is checked using test files. In collection phase Test driver for running the files is executed.

After all these process equivalent fragments or clones are obtained.

Thierry Lavoie et al. [9] used levenshtein distance **[10]** to find out the clones. This technique is combination of token based **[11]** and metric based **[12]** clone detection techniques. Metric trees and Manhattan distance are used for accurate estimation of levenshtein distance. Levenshtein distance is measure of similarity between strings. It calculates no of insertion, deletion and swapping of characters to transform string s1 into s2. In the first step tokens are extracted from source code with lexical analyzer. In the second step frequency vectors are built. A unique id is provided to each token. Id is provided dynamically, if any new item is discovered an available id is provided to that token else corresponding vector of token is incremented by one. Hashing Tables are used to store vectors. In the next step metric tree are built using frequency vectors. L1 metric i.e. Manhattan distance is

used for this purpose. Manhattan distance is chosen because of the following reason:

- L1 covers less space.
- L1 is fast and high precision.
-

In the fourth step Metric tree is built with all the vectors. Metric tree separate the search space and increase the speed of insertion, range queries and nearest neighbors. The last step is tree query step. In this step Manhattan distance between two metric trees is measured. If two trees have Manhattan distance less than threshold, then they would be clones.

Dandan Kong et al.[13] used k-nearest algorithm to find clones. The advantage of AST, control dependency graph and k-nearest neighbor clustering algorithm **[14]** is taken in this algorithm. Two fragments are defined as functional equivalent if there exist two permutations p1, p2 such that for

$$OC1\ (p1\ (I)) = OC2\ (p2\ (I))$$

where OC represents the output set of C1 and C2 code fragments.

Firstly code is passed to lexical analyzer and syntax analysis to obtain the control dependency information by generating AST and PDG. Then functionally cohesive code is obtained by nearest neighbor clustering algorithm. Input and Output variables are identified **[13]**. Then uncompilable code is transformed into executable functions. The automated generated input is given to function and dynamically tested. Then according to output results generated on inputs equivalent functions are grouped into same cluster.

4. Comparisons:

Table 2: Comparison of Techniques of different authors

	Techniques Used	**Scalability**	**Portability**	**Complexity**
Thierry Lavoie et al. [9]	Combination of Token based and metric based	Less	This technique is applied on 'c' language	More complex
Ira et al. [3]	Tree Based	Less	Parser used for transformation, is language dependent	More complex
Yong Yuan et al. [4]	Token Based	More	Language Independent	Less complex
Dandan Kong et al.[13]	Combination of tree based and Graph based	Less	Parser is language dependent	More complex
Rochelle et al. [6]	Based on behavior of fragments	More	This Technique is applied for java language	Less complex

5. Conclusion

Clones have many drawbacks. They increase the maintenance cost and the probability of bugs. So clones need to be removed. Clones can't be removed completely. Many techniques have been used for clone detection till now. Each technique has its own advantages and limitations. Techniques are applied according to requirements. Sometimes hybrid techniques are used. More techniques can be proposed with less complexity and high scalability.

6. Future Scope:

It will be tried to find out more and more clones to decrease the no of false clones.There are many techniques for clone detection, but very few are to refactor the clones. So our future objective is to purpose a novel for clone management having less complexity.

References

[1] Z. Li, S. Lu, S. Myagmar and Y. Zhou, CP-miner: Finding copy-paste and related bugs in large-scale software code, IEEE, 2006.

[2] C. K. Roy, J. R. Cordy, and R. Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, Sci. Comput. Program, 2009.

[3] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, Lorraine Bier, Clone Detection Using Abstract Syntax Trees, IEEE, 2011.

[4] Yang Yuan,YaoGuo, Boreas: an accurate and scalable token-based approach to code clone detection, In

Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering - ASE,IEEE 2012.

[5] Y. Yuan and Y. Guo. CMCD: Count Matrix based Code Clone Detection, APSEC.Thierry Lavoie, EttoreMerlo, An accurate estimation of the Levenshtein distance using metric trees and Manhattan distance, In 6th International Workshop on Software Clones (IWSC), IEEE, 2012.

[6] Rochelle Elva, Gary T. Leavens, Semantic Clone Detection Using Method IOE-Behavior,InIWSC, Zurich, Switzerland,IEEE, 2012.

[7] E. Juergens, F. Deissenboeck, and B. Hummel, Clone detection beyond copy & paste, in Proc. of the 3rd International Workshop on Software Clones, 2009.

[8] R. Elva and G. Leavens, Jsctracker: A semantic clonedetection tool for Java code, University of Central Florida, Department of EECS,University of Central Florida,4000 Central Florida Blvd,Orlando, Florida, 32816, USA, Research paper CS-TR-12-04,March 2012, http://www.eecs.ucf.edu/ leavens/tech-reports/UCF/CS-TR-12-04/TR.pdf,2012.

[9] Thierry Lavoie, Ettore Merlo, An accurate estimation of the Levenshtein distance using metric trees and Manhattan distance, In 6th International Workshop on Software Clones (IWSC) ,IEEE,2012.

[10] Lavoie and E. Merlo. Automated type-3 clone oracle using levenshtein metric, 2011, ACM, 2011.

[11] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: A multilinguistic token-based code clone detection system for large scale source code, IEEE, 2002

[12] E. Merlo, G. Antoniol, M. D. Penta, and F. Rollo. Linear complexity object-oriented similarity for clone detection and software evolution analysis. In Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press, IEEE, 2004

[13] Dandan Kong, Xiaohong Su, Shitang Wu, Tiantian Wang and Peijun Ma, Detect Functionally Equivalent Code Fragments via K-nearest, IEEE fifth International Conference on Advanced Computational Intelligence(ICACI) October 18-20,2012 Nanjing, Jiangsu, China, 2012.

[14] C. Lung, X. Xu, M. Zaman, A. Srinivasan, Program restructuring using clustering techniques, The Journal of Systems and Software, 2006.