Literature Survey of Clone Detection Techniques

Sonam Gupta
Research Scholar,
Suresh Gyan Vihar University
Jaipur (Rajasthan), India

P. C Gupta, Ph.D
Department of Computer Science & Engineering
University of Kota
Kota (Rajasthan), India

ABSTRACT
Code clones are the codes which have same code in the system and so it is difficult to locate all the same codes in the system when any change is to be done. Researchers have proved that almost 70% of the effort done during maintenance is just because of the occurrence the clones in the system. A number of approaches had been given earlier to detect various types of clones [39]. This paper presents the systematic literature review of all the detection approaches researched so far. Along with it this paper also gives the advantages to implement them and also all the defects due to which they were not able to completely detect the clones. It also gives a novel approach to automatically detect the clones irrespective of the matter that whether the code is in same order or any statement has been inserted, deleted or modified in the code fragment.

Keywords
Clones maintenance, Program dependence graph, tree-based approach, false positives, and hybrid approach

1. INTRODUCTION
Maintenance effort of the system increases with the increase in complexity of the system. There are four types of maintenance namely corrective maintenance, adaptive maintenance, perfective maintenance and preventive maintenance [36]. Corrective maintenance is the reactive modification for a software product performed after delivery to correct discovered problems. Adaptive maintenance is the modification of a software product, performed after delivery, to keep a software product usable in a changed or changing environment. Perfective maintenance is the modification of a software product after delivery to improve performance or maintainability. Preventive maintenance is the modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults. Arthur states that only one-fourth or one-third of all life-cycle costs are attributed to software development and that some 67% of life-cycle costs are expended in the operation-maintenance phase of the life cycle [35,37]. The major challenge during maintenance is the difficulty to trace the product or the process that created the product, changes are not adequately documented, lack of change stability and ripple effect when making changes. There are many approaches that had given by many researchers to detect such types of changes primarily known as clones. In this paper we have presented a systematic literature survey for the detection of clones. The rest of the paper is organized as follows.
Section II consists of the literature survey of past 10 years in the field of clone detection along with it the section also gave a novel approach to overcome the flaws of previous approaches. Section III consists of conclusion and future work and section IV consists of all the references from where the survey has been done.

2. LITERATURE SURVEY
This section includes the literature survey of past 10 years in the clone detection techniques. This will help to find the various merits and demerits of various approaches developed so far so that a new and better approach of clone detection can be developed.

Text-Based Approach: The code fragments are considered as sequence of text and are compared with each other as the basis of various transformations like removing comments, whitespace, newlines etc. There are various researchers who found various techniques to detect clones on the basis of text. Baker [2,3] used line-based string matching algorithm by making the tokens of each sequence/line of the text by help of a tool named as Dup. This technique consists of all the basic properties of text-based technique, along with this it also replaced identifiers, variables and types with a special parameter so that even if the name of variable is different the clone can be identified. But this tool was not able to support exploration and navigation between the duplicated codes and moreover it cannot detect clones if the code is written in different style. This problem was overcome by Koshke et al.by [23] by finding the clones on the basis of tokens rather than lines but this was not able to keep track that whether the identifiers had been renamed consistently or not after transformation. Johnson used [19] used Karp-Rabin fingerprinting algorithm to detect clones on purely text basis. In this technique each character is included in atleast one substring and then the matching of those substrings is done. The disadvantage with this technique is that it had the restriction of keeping 50 lines match resulting in more number of false positives. Cordy et al. [9] used this text-based approach to detect the near miss clones for HTML web pages. In this technique firstly syntactic constructs are identified and then used as smallest comparison but this did not normalize any code. Ducasse et al. [12,13] presented an approach using dynamic pattern matching which is language independent but this was not able to identify meaningful clones because the cohesiveness of the code gets effected. Marcuss[28] gavean approach which used latent semantic indexing [14] and does not compare the whole code rather than that it identify clones by creating certain domains of comments and identifier matching. But this cannot detect clones if the structure is same but the name of identifiers is different. All the above detected techniques clearly show that although the cost of applying the approach is very less but the code having line break, change in variable name, type, change of parenthesis etc. cannot be identified and tested that whether it is a cloned code or not.

Token-based approach: This technique parse the whole source code into sequence of tokens thereby overcoming the problems faced in text-based approach likechange in space, identifier name etc. Kamiya et al.[6]developed a tool named as CCFinder in which each line is divided into tokens and then added together to form a single token, so that even if the

name of identifier etc. is changed then it will not effect the detection of clone even if the structure of the code is same but there are some minor avoidable changes in the code. Even though Baker [2,4]also used the token scheme to detect the clone but it did not use any transformation technique resulting in detection of false positives. For more flexible tokenization RTF[7] used suffix array rather than suffix tree so that unnecessary tokens can be removed so as to reduce the false detection but this technique is more complex to implement. To overcome the problem of CCFinder and Dup i.e. cannot detect clones if there are minor changes in the code, CPMiner [26,27] was introduced which can ignore insertion/deletion or any modifications of code upto 1-2 statements only. Juergens et. al [40] gave a plug-in in visual studio which can detect the clones in Java and C# but the approach was not able to handle the defects at programmer side itself. Almost same approach was given by Kawaguchi et. al[41] but it was developed for C++ and C# but it did not overcome the problem as in [40].

Tree-based approach: Rather than creating tokens for each statement this technique creates sub trees of a fragment of the code and the code is said to be a cloned code if the corresponding sub trees match.. this is done by creating the AST of the code. CloneDR [8] is a well known tool which uses this technique. It generates the parse tree and then by the help of hash functions the subtrees are matched. But this technique was not able to identify similar clones. This problem was overcome by CCdiml[31] tool given by Bauhaus but this was not able to identify renamed identifiers. Yang[34] also proposed an approach based on the same technique which finds the syntactic difference between the versions of the system and creates their parse tree. Nahler et. al [33] gave the approach which convert the AST into XML and then by using data mining technique[1] it extract the clones. This approach was further refined by Evas & Fraser [15] to find near miss clones by using only AST leaves rather than the tree, but again it was not able to detect much of the exact clones. Duala Ekoko et. al[38] hd developed a tool named Clone Tracker in Java but again the number of false positives are much more in it and it is not able to detect post programming. Hoan Anh [43] developed a clone management tool in Java but it increases tht time to find the clones. All the above mentioned researches clearly show that AST is not able to find the gapped clones along with it the cost of search space also increases. It does not follow the data flow and also cannot detect the clones is the statements are reordered. All the drawbacks of AST can be easily overcome by the use of PDG-based technique.

PDG-based technique: Program Dependency Graph (PDG) [20,24,25] overcome the problems faced in AST and also maintain the data and control flow [80] and therefore it become easier to clones semantically as well as syntactically. Komondoor and Horwitz [20,21] gave an approach known as PDG-DUP which used program slicing method to identify the clone groups without changing its semantics. Gallagher and Lucas [16] extended the work of Komondoor et. al by applying program slices on all the variables of a code but could not come to any conclusion. PDG technique was used as an iterative approach by Krinke[24] for finding the maximal similar subgraph but it was not able to give a formula that can be used on any type of system to find the clone. All the researchers are using the PDG technique came to the conclusion that although PDG-based techniques can find non-contiguous clones but it cannot be applied to large systems.

Metric-based technique: This technique does not compare the code directly instead it calculated different metric (like number of source lines, number of functions etc) for the code and then these metrics are compared. Mayrand et. al[29] used this technique and calculated the metrics from names, layouts, expression and control flow but it was not able to identify segment based copy-paste operation. Kontogiannis et. al [22] used markov model but it can only measure the similarity between the codes rather than finding the exact clones. They modified the approach by calculating the metrics on the basis of begin-end block and the code is said to be cloned code only if their metrics is approximately same. Di Ducca et. al[11] also used this approach to find the clones in static HTML pages by calculating their degree of similarity. This was done by calculating the Levenshtein distance of the code [67]. Lanubile Calefato [161,46] used eMetrics tool to identify the clones and then check the extracted clones manually to find that whether the extracted clone is a true positive or not but this was not feasible to be implemented for large systems. So, the metric-based approach is able to extract the clones from the code but the metrics may vary from system to system resulting in different cloned codes for the same system.

Hybrid approaches: These approaches are the combination of various approaches discussed above. Koschke et. al [23] used tree based and token based technique to identify the clones. From the tree-based technique firstly a suffix tree is created and then it is compared by using token-based approach. This is used only to find exact and type-II clones. Almost similar approach was given in Microsoft's new phoenix framework [32] but it can only identify clones with change in name of identifier not of change in its type. Greenan[17] gave the same approach using sequence matching algorithm. Jiang et al. [18] used the AST in Euclidean space and then Locating Sesitive Hasing (LSH) [10] is used to cluster the vectors based on similarity. Dynamic Pattern Matching technique was proposed by Balazinska [6] in which the characteristic metrics are computed for the code and then the clusters are identified using Patenaude's [30] metric-based approach. De Wit [43] gave an approach based on dynamic change tracking and resolution in Java language but it failed in detecting the data flows and examining the clones at the programmer's level.

3. PROPOSED APPROACH

All the advantages and disadvantages of various approaches discussed in section II clearly show that although many techniques but still none is able to find the clones correctly. So we are proposing a hybrid technique which is able to find more number of true positives. This approach will find all the clones in the system irrespective of their place and will show the same to the programmer so that after or during the development of the code the programmer itself can identify the chunks which contain the clone and can decide whether to remove the clone or it is a good smell. In the proposed work one or two fragments can be compared. It will give the result in form of chucks so it will be called as clone-chuck extraction algorithm. Firstly, the statements of the code will be examined serially if the statements are found to be reordered then the semantic-preserving transformation will be applied to the code so that reordering of the code does not affect the procedure of identifying the clone. This approach will be a bit faster removing the sluggish behavior of many approaches discussed above and also will handle all types of clones [39].

4. CONCLUSION AND FUTURE WORK

In this paper we have surveyed papers of past 10 years and found that there many approaches given by various researchers to detect the clones which primarily includes text based, token based, tree based, PDG based, metric based and hybrid technique. Although many algorithms had been developed based on these approaches but still none is able to find the clone with accuracy and efficiency. Some algorithms can detect only a particular type of clone and some are so slow that whole system comes to a bottleneck if large system is to be compared. So we have also proposed an algorithm which will find all types of clones that too with accurate clones and more efficiency. Moreover by using the clone-chunk extraction algorithm the developer can decide whether to remove it or not and then can mention the same in the documentation so that the maintenance team do not waste their time in resolving the issues regarding the clones.

Our future work will be to implement the clone-chunk extraction algorithm and check it on various systems.

5. REFERENCES

[1] Brenda S. Baker. Finding Clones with Dup: Analysis of an Experiment. IEEE Transactions on Software Engineering, Vol. 33(9): 608-621, September 2007.

[2] Brenda S. Baker. A Program for Identifying Duplicated Code. In Proceedings of Computing Science and Statistics: 24th Symposium on the Interface, Vol. 24:4957, March 1992.

[3] Brenda S. Baker. Parameterized diff. In Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms (SODA'99), pp. 854-855, Baltimore, Maryland, USA, January 1999.

[4] Brenda S. Baker. On Finding Duplication in Strings and Software. Journal of Algorithms, 1993.

[5] Brenda Baker. On Finding Duplication and Near-Duplication in Large Software Systems. In Proceedings of the Second Working Conference on Reverse Engineering (WCRE'95), pp. 86-95, Toronto, Ontario, Canada, July 1995.

[6] Magdalena Balazinska, Ettore Merlo, Michel Dagenais, Bruno Lague, Kostas Kontogiannis. Measuring Clone Based Reengineering Opportunities. In Proceedings of the 6th International Software Metrics Symposium (METRICS'99), pp. 292-303, Boca Raton, Florida, USA, November 1999.

[7] Hamid Basit, Simon Pugliesi, William Smyth, Andrei Turpin, and Stan Jarzabek. Efficient Token Based Clone Detection with Flexible Tokenization. In Proceedings of the Joint Meeting of the European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE'07), pp. 513-515, Dubrovnik, Croatia, September 2007.

[8] Ira Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna. Clone Detection Using Abstract Syntax Trees. In Proceedings of the 14th International Conference on Software Maintenance (ICSM'98), pp. 368-377, Bethesda, Maryland, November 1998.

[9] James Cordy, Thomas Dean, Nikita Synytskyy. Practical Language-Independent Detection of Near-Miss. In Proceedings of the 14th IBM Centre for Advanced Studies Conference (CASCON'04), pp. 1 - 12, Toronto, Ontario, Canada, October 2004.

[10] M. Datar, N. Immorlica, P. Indyk and V. S.Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In Proceedings of the 20th annual symposium on Computational geometry (SoGG'04), pp. 253-262, Brooklyn, New York, USA, June 2004.

[11] G.A. Di Lucca, M. Di Penta, and A.R. Fasolino and P. Granato. Clone Analysis in the Web Era: an Approach to Identify Cloned Web Pages. In Proceedings of the 7th IEEE Workshop on Empirical Studies of Software Maintenance (WESS'99), pp. 107-113, Florence, Italy, November 2001.

[12] St´ephane Ducasse, Oscar Nierstrasz, and Matthias Rieger. Lightweight detection of duplicated codea language-independent approach. Technical report, University of Bern, Institute of Computer Science and Applied Mathematics, Bern, Switzerland, February 2004.

[13] St´ephane Ducasse, Matthias Rieger, Serge Demeyer. A Language Independent Approach for Detecting Duplicated Code. In Proceedings of the 15th International Conference on Software Maintenance (ICSM'99), pp. 109-118, Oxford, England, September 1999.

[14] Susan T. Dumais. Latent Semantic Indexing (LSI) and TREC-2. In Proceedings of the 2nd Text Retrieval Conference (TREC'94), pp. 105-115, Gaithersburg, Maryland, March 1994.

[15] Williams Evans, and Christopher Fraser. Clone Detection via Structural Abstraction. In Proceedings of the 14th Conference on Reverse Engineering (WCRE'07), Vancouver, BC, Canada, October 2007(to appear, available as Technical Report since August 2005).

[16] Keith Gallagher, Lucas Layman. Are Decomposition Slices Clones? In Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC'03), pp.251-256 Portland, Oregon, USA, May 2003.

[17] Kevin Greenan. Method-Level Code Clone Detection on Transformed Abstract Syntax Trees using Sequence Matching Algorithms. Student Report, University of California -Santa Cruz, Winter 2005.

[18] Lingxiao Jiang, GhassanMisherghi, Zhendong Su, and Stephane Glondu. DECKARD: Scalable and Accurate Tree-based Detection of Code Clones. In Proceedings of the 29th International Conference on Software Engineering (ICSE'07), pp. 96-105, Minnesota, USA, May 2007.

[19] J Howard Johnson. Identifying Redundancy in Source Code Using Fingerprints. In Proceeding of the 1993 Conference of the Centre for Advanced Studies Conference (CASCON'93), pp. 171-183, Toronto, Canada, October 1993.

[20] Raghavan Komondoor and Susan Horwitz. Using Slicing to Identify Duplication in Source Code. In Proceedings of the 8th International Symposium on Static Analysis (SAS'01), Vol. LNCS 2126, pp. 40-56, Paris, France, July 2001.

[21] Raghavan Komondoor. Automated Duplicated-Code Detection and Procedure Extraction. Ph.D. Thesis, 2003.

[22] K. Kontogiannis, M. Galler, and R. DeMori. Detecting code similarity using patterns. In Working Notes of 3rd Workshop on AI and Software Engineering, 6pp., Montreal, Canada, August 1995.

[23] Rainer Koschke, Raimar Falke and Pierre Frenzel. Clone Detection Using Abstract Syntax Suffix Trees. In Proceedings of the 13th Working Conference on Reverse Engineering (WCRE'06), pp. 253-262, Benevento, Italy, October 2006.

[24] Jens Krinke. Identifying Similar Code with Program Dependence Graphs. In Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01), pp. 301-309, Stuttgart, Germany, October 2001.

[25] Chao Liu, Chen Chen, Jiawei Han and Philip S. Yu. GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis. In the Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06), pp. 872-881, Philadelphia, USA, August 2006.

[26] Zhenmin Li, Shan Lu, Suvda Myagmar, Yuanyuan Zhou. CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code. In Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI'04), pp. 289-302, San Francisco, CA, USA, December 2004.

[27] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code. In IEEE Transactions on Software Engineering, Vol. 32(3): 176-192, March 2006.

[28] Andrian Marcus and Jonathan I. Maletic. Identification of high-level concept clones in source code.In Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE'01), pp. 107-114, San Diego, CA, USA, November 2001.

[29] Jean Mayrand, Claude Leblanc, Ettore Merlo. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In Proceedings of the 12th International Conference on Software Maintenance (ICSM'96), pp. 244-253, Monterey, CA, USA, November 1996.

[30] J.-F. Patenaude, E. Merlo, M. Dagenais, and B. Lague. Extending software quality assessment techniques to java systems. In Proceedings of the 7th International Workshop on Program Comprehension (IWPC'99), pp. 4956, Pittsburgh, PA, USA, May 1999.

[31] Aoun Raza, Gunther Vogel, Erhard Pl¨odereder. Bauhaus–A Tool Suite for Program Analysis and Reverse Engineering. In Proceedings of the 11th Ada-Europe International Conference on Reliable Software Technologies , LNCS 4006, pp. 71-82, Porto, Portugal, June 2006.

[32] Robert Tairas, Jeff Gray. Phoenix-Based Clone Detection Using Suffix Trees. In Proceedings of the 44th annual Southeast regional conference (ACM-SE'06), pp. 679-684, Melbourne, Florida, USA, March 2006.

[33] V. Wahler, D. Seipel, Jurgen Wolff von Gudenberg, and G. Fischer. Clone detection in source code by frequent itemset techniques. In Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation (SCAM'04), pp. 128135, Chicago, IL, USA, September 2004.

[34] Wuu Yang. Identifying syntactic differences between two programs. In Software Practice and Experience, 21(7):739755, July 1991.

[35] S. W. L. Yip and T. Lam. A software maintenance survey. In *Proc. of the 1st Asia-Pacific Software Engineering Conference*, pages 70–79, Dec 1994.

[36] ISO/IEC. *Software Engineering - Software Maintenance*. ISO/IEC 14764, 1999.

[37] L. Arthur. *Software Evolution: The Software Maintenance Challenge*. Wiley 1988.

[38]Duala-Ekoko, Ekwa, and Martin P. Robillard. "Clonetracker: tool support for code clone management." *Proceedings of the 30th international conference on Software engineering*. ACM, 2008.

[39] Sonam Gupta, Dr. P.C. Gupta, " Clones : A Survey", International Journal of Computer Scinece and Technology Vol. 3, Issue 3, July - Sept 2012.

[40]Juergens, Elmar, Florian Deissenboeck, and Benjamin Hummel. "CloneDetective-A workbench for clone detection research." *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 2009.

[41] Kawaguchi, Shinji, et al. "Shinobi: A tool for automatic code clone detection in the ide." *Reverse Engineering, 2009. WCRE'09. 16th Working Conference on*. IEEE, 2009.

[42] De Wit, Michiel, Andy Zaidman, and Arie Van Deursen. "Managing code clones using dynamic change tracking and resolution." *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. IEEE, 2009.

[43] Nguyen, Hoan Anh, et al. "Clone management for evolving software." *Software Engineering, IEEE Transactions on* 38.5 (2012): 1008-1026.