A REVIEW ON AN EFFECTIVE AND EFFICIENT APPROACH OF SOFTWARE CLONE DETECTION METHOD

Megha Jha[1], Mr. Akash Sarswat[2], Ms. Arundhati Walia[3]

[1,2,3]Department of Computer Science And Engineering,
H.R.I.T. Group of Institutions, Ghaziabad(U.P)

***Abstract*: Making copy existing code portion and pasting them without modifications or with modifications into other sections of code is very common process in software development. The code which is copied is called a cloning of the software and the process is called software cloning. Software developers tend to copy the existing functionality from the source code and paste it somewhere else either intentionally or unintentionally. The increasing use of open source software and its variants increased code reuse, too. The existing codes can be modified according to new requirements thereby facilitating and advancing open source development. Clone management tools should be similar and integrated into development environments, to help designers understand the behavior of integration patterns. Combine a management tool with integrated discovery and friendly clones recognized by the developer that will help engineers identify clones as they grow over time. The discovery of software clones will help raise awareness of clones that exist within systems and are aimed at preventing the creation of new clones. We believe that the results of our systematic review will be useful in any case for a researcher who wants to continue research in any field related to software clones such as clone management, clone detection, clone analysis, software quality clones, etc. This paper presents a systematic review of existing literature performed to analyze and report the findings of a clone-based study**

***Keywords*: cloning, software clone, model clone, clone detection etc.**

Introduction: Software clones exist in many different software artifacts. After examining the state of the art in the software clone study, we noticed a lack of systematic review of the literature. We therefore summarized existing research based on extensive and systematic search of information and reported research opportunities for further investigation. Systematic reviews provide important evaluation and integration of existing research, a way to identify and interpret all available research related to a specific research question, or topic area, or item of interest. Software may be integrated with retractable engineering or officially restarted from documents or other sources, or by looking at the appearance and behavior of the program. Reasons to integrate software may include skipping unnecessary license fees, obtaining information about system features or creating an unsupported platform. GNU, a copy of UNIX, was inspired by the need for a Free Software Movement operating system built entirely free software. Since 2010 clone computing, in the sense of duplicating a session on a computer host in a cloud-based environment, has been introduced. This allows the user to access the desktop copy of his PC on any other computer device such as desktop computer, personal computer using any operating system, WebOS, smartphones, etc. Clone computer duplicates, operates, and is regularly available in a series of cloud servers. Unlike remote management software, clone computing does not have to depend on a host computer.

The review protocol includes a research questionnaire, a search site, methods used to identify and evaluate the evidence. The current review includes the identification of basic subjects, the application of exclusion and exclusion criteria and the integration of outcomes. Electronic database is highly searched and its subjects are reported. Extensive view is needed to cover a wide range of publications. Before starting a search, a suitable set of information should be selected to increase the chances of finding the most relevant articles. In almost every search, the keyword "clone" is included in the abstract. It is a complex and time-consuming process.

Ekram et al. [1] used the Knuth-Morris-Pratt algorithm for character comparisons to update clone information from a server continuously to save time in client server setup. Later, only related clones were downloaded by individual developers. The process is available faster than the Simian based series and ASS-based AST. However, as it uses a series-based strategy, it fails to detect clones with small and large changes.

Yamashina et al. [2] proposed a novel acquisition / modification tool to support the software maintenance process. The study reported significant differences between beginners and programmers who are knowledgeable about motivation and behavior in the management of clones. They are then added to the suffix list for immediate retrieval. Clone detection and leveling is done by SHINOBI server. Tests show that SHINOBI is fast and accurate. In a previous study, a large number of program planners were interviewed to analyze their behavior. SHINOBI still needs to be upgraded to a standard algorithm. However, it can be expanded to support more useful information in addition to coding.

Yang et al. [3] suggested one of the first ways to detect structural differences between two versions of the same system. The process was based on grammar and created a variation of the analysis tree in both versions. The findings are applied equally to both trees and are based on the long-term general tracking of the dynamic system. The limitation of this approach is that its divisive part can only apply to appropriate grammatical programs.

Deckard by Jiang et al. [4] is based on the calculation of the vectors with AST signals and the adjacent vector connecting in the Euclidean space with sensitive area hashing. Deckard has been used to represent clone groups and has been used to obtain the same code of conduct. Another use of the deck is to check the effect of code encryption errors on source code.

Hummel et al. [5] developed an integrated reference clone detector that takes input in the form of a token sequence. The index is used to view all clones in a single file and allows merging, deleting or modification. The tool is used in ConQAT and works fast thus growing, growing and providing excellent working time performance.
Koschke et al. [6] test their AST-suffix cpdetector tool based on accuracy, memory and timing versus existing tools using the Bellon test process and additional tools with cpdetector, ccdiml, clones and cscope. As their advanced tool is only suitable for system C, so they only work with Bellon 4 C test systems and provide only detailed system results. The results show that the AST ccdiml-based tool shows good memory (53%). However, the recall rate for memory-based tools is almost double (54%) than for AST-based tools (30%). Their test result shows that clones received 71% more clones than CCFinder. The CLAN-based tool is good and the duplix is very bad in terms of operating time.
Falke et al. [6] officially compared ccdiml, cpdetector, clast, clast-ba, clones-uk, clones-ba and cscope using theme systems in C and Java. The results show that tools based on tokens produce a large number of clones. AST comparisons show a lower level of rejection, but also lower memory. The result shows that tree-based detection is faster than tree-based comparisons. Tree-based tools show a lower level of rejection than AST-based tools for C programs but a higher level for Java programs. And tree-based tools for analysis are faster than AST-based tools for Java applications. Contrary to previous tests, this study does not show high recall of token-based tools. According to this study the advantage of token-based strategies is that lexer is easieruse than the parser and requires less space than AST. On the other hand, AST-based techniques are good for finding artificial clones and helping to filter out artificial structures that have little interest.
Balazinska et al. [9] developed a CloRT naming tool that works with any AST-based clone detector. Their analysis focuses on two aspects of clones which means the definition of their differences in the programmers' perspective and contextual analysis that helps to transform things. The study concluded that clones are good candidates for reconstruction. Gemini clone analysis tool captures CCFinder output and displays results in the form of a scatter structure and metric graph. The tool provides users with useful functions for analyzing, storing and rearranging code.
Tiarks et al. [7] found a large number of false positives i.e. height of 75%. This has a detrimental effect on clone recognition. It also prevents the use of clone detection for practical purposes. Subsequent studies were performed to detect and remove code clones in Erlang / OTP and Haskell programs. Their research used the metrics and Levenshtein range to reflect changes in co-operation to the code to the developer.
Jurgens et al. [8] performed clone detection to obtain copying and pasting functions in documentation of specified requirements. Larger transcripts are used to detect any loss of function, thus assessing the level of clarity. Extensive research was done on all 28 documents with 9000 pages, adjusting the clone length to 20 words. Clones were not found to be particularly fond of mistakes. The tool for obtaining Clone, Deckard and extraction of data from version control and bug fixes was used for research. His research has significant results in terms of software maintenance. With common theme systems such as inputs and a growing clone detector tool, research identified the extraction process as the most commonly used recycling method.
We reviewed many articles in the collection and provided categorization and general overview. Unlike previous surveys, we place emphasis on clone management, clone models and semantic clones and classify literature into different key areas. In addition to clone detection tools and methods, we also deal with other clone software research issues such as clone analysis, clone emergence, impact of clones on software quality.

Proposed work : The purpose of our work is to test UML models for the presence of clones. In our paper, we have developed a strategy where UML models are made using any a standard modeling tool. These models are then exported to XMI representation. XMI file is fragmented to extract various features of the UML category model such as class name, Field name with data type, path name and return type, etc. Our core technology is the construction of a labeled tree, so limited that its lower trees represent the field and its data type and signature method. The existing algorithm identifies a variety duplicate small trees of different sizes from the constructed tree. We used a systematic approach to create a clone management map that shows how clone management papers interact with clone detection method sheets and clone detection tool sheets. We examined model-based and semantic clones in detail and compared the status of artistic techniques. By integration and integration in the recurring footnotes, the proposed method is able to identify clones that are model different granularity. The novel classification of model models provides useful information on software modeling processes. Another highlight of the proposed strategy is its ability to detect clones in three different levels of granularity in the UML class model which is complete class, attributes with their data types and methods with signatures and a collection of those repetition of attributes and methods.
To understand big systems, one needs to see the big picture. So the second process of our paper focuses on identifying high-level similarities in source code and UML models. We called these clones as clumsy as the concept clones. Since, users can choose different implementations in the form of class models of the same concept of the real world. Therefore, we have expanded our approach to finding similarities between the two different categories of UML models. The discovery of concept clones finds applications in different domains such as software product lines, clone emergence, software crime detection, etc. The proposed method is able to identify conceptual clones through the principal partial analysis and hidden semantic index. The distance between the two pieces of source code or UML models calculated using cosine similarity. It is important to identify the concept clones at the level of the invisible model rather than the lower levels. So do we test the tool by using it so that a sample of UML class models goes down the path level within the same source code file.

Conclusion

There is recent research showing that clones can be used more often as a means of regeneration, and can be beneficial in many ways. Also, it is not easy to replicate all clones because of the cost / risk associated with repetition. It is therefore suggested that

instead of removing clones, we should have proper clone handling areas. In order to improve the state of the art in clone management, one needs to know advances in clone research itself. We have tried to do this by finding the right books and summarizing them in the form of a systematic map. The main goal of this systematic review was to identify and classify the existing literature focusing on clone detection, clone management, semantic clone detections and model based clone detection techniques. Also to investigate the available literature as per the research questions mentioned above and propose our work as an alternate.

REFERENCES

1. R. Al-Ekram, C. Kapser, R. Holt, M. Godfrey, Cloning by accident: An empirical study of source code cloning across software systems, in: Proceedings of International Symposium on Empirical Software Engineering (ISESE'05), Noosa Heads, Australia, 2015, pp. 376-385.
2. T. Yamashina, H. Uwano, K. Fushida, Y. Kamei, M. Nagura, S. Kawaguchi, H. Iida, SHINOBI: A tool for automatic code clone detection in the IDE, in: Proceedings of the 16th Working Conference on Reverse Engineering (WCRE'09), Lille, France, 2019, pp. 313-314.
3. W. Yang, Identifying syntactic differences between two programs, Software Practice and Experience 21 (7) 739-755.
4. L. Jiang, G. Misherghi, Z. Su, S. Glondu, DECKARD: Scalable and accurate treebased detection of code clones, in: Proceedings of 29th International Conference on Software Engineering (ICSE'07), Minneapolis, MN, USA, 2017, pp. 96-105.
5. B. Hummel, E. Juergens, L. Heinemann, M. Conradt, Index-based code clone detection: Incremental, distributed, scalable, in: Proceedings of the 26th IEEE International Conference on Software Maintenance (ICSM '10), Timisoara, Romania, 2010, pp. 1-9.
6. R. Falke, P. Frenzel, R. Koschke, Empirical evaluation of clone detection using syntax suffix trees, Empirical Software Engineering 13 (6) (2018) 601-643.
7. R. Tiarks, R. Koschke, R. Falke, An assessment of type -3 clones as detected by state-of-the-art tools, in: Proceedings of the 9th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM"09), Edmonton, Canada, 2019, pp. 67-76
8. E. Juergens, F. Deissenboeck, M. Feilkas, B. Hummel, B. Schaetz, S. Wagner, C. Domann J, , Streit, Can clone detection support quality assessments of requirement specifications? in: Proceedings of 32nd International Conference on Software Engineering (ICSE'10), Cape Town, South Africa, 2010, pp. 79-88.
9. M. Balazinska, E. Merlo, M. Dagenais, B. Laguë, K. Kontogiannis, Advanced clone-analysis to support object-oriented system refactoring, in: Proceedings of the 7th Working Conference on Reverse Engineering (WCRE'00), Brisbane, Queensland, Australia, 2000, pp. 98–107.