A Study of Code Clone Detection in Software Systems

Jai Bhagwan
Assistant Professor
Department of Computer Science & Engineering,
Guru Jambheshwar University of Science & Technology, Hisar, India

Abstract: Software engineering is a wider area of research. Various hot topics are picked for research like quality prediction, bug detection, clone detection, efforts estimation etc. Software clone detection can save the development efforts and helpful in developing efficient software. The clones are useful for code reuse but, the copy-paste technical may increase the problem of code reliability and robustness. Unnecessary duplicate code not only affects the software itself but entire system performance is affected. So, this research covers the software clone detection techniques. The scientists have introduced various tools and techniques for clone detection such as token based code representation, abstract syntax tree based model, CDLH, NiCad, LWH model, PMD, CodePro etc. In this paper, various researches are studied and compared.

IndexTerms **– Abstract Syntax Tree, Code Clones, CCLearner, NiCad, SDLC.**

I. INTRODUCTION

People are living in the digital era; as a result, the demand of software is rapidly growing. In software development life cycle (SDLC) various phases are included such as requirements and documentation, designing, coding, testing, implementation and maintenance. Coding part is a major part in software development as it is the backbone of the entire development process. There are various forms of redundancy or replication in software programs. This type of coding redundancy is typically referred to as clones; many taxonomies and definitions of clones have been put forth [1]. In software development, it's common practice to copy preexisting code pieces and paste them with or without alterations into other code sections. Software clones are code copies, and the method of creating them is known as software cloning [2]. It is discovered that code cloning poses a greater risk to industrial software systems. The system's regular operation might not be impacted by clones, but if the maintenance department doesn't take preventative action, future development might become unaffordable. Clones are thought to be detrimental to growth. Code clones may have a negative impact on the quality of software systems, particularly on their readability and maintainability. Cloning, for instance, raises the likelihood of update irregularities [6]. The Big Data era has brought new clone detecting applications. Clone detection has been used to locate code examples, categorize code snippets effectively, find comparable mobile applications, and more. Numerous clone detection technologies have been offered A few of such methods are just text-based, while others are token-based, tree-based, metrics-based, graph-based, and hybrid—for instance, parser-based but with text comparison, etc [4].

Refactoring is a method that facilitates the processing of code clones. According to Fowler (1999), refactoring is the process of "restructuring an existing body of code, altering its internal structure without changing its external behavior." Refactoring the identified clones may enhance the system's interpretability, maintainability, and extensibility while lowering its complexity (Fowler 1999) [7]. It makes sense to quantify the functional similarity if the code fragments' functional behaviors could be meticulously characterized. By simultaneously taking into account the lexical data and syntactical patterns of code fragments, latent properties that define the functionalities of the code might be acquired in order to accomplish this goal [8].

1.1 Clone Detection Techniques

There are various clone detection techniques. A few of them are explain below in short [5].

- **Token-Based -** Token-based approaches generate a stream of tokens by applying complex changes to the source code, such as lexical analyzers. To find out if two code fragments are clones or not, these algorithms compare tokens of the two fragments.

- **Metrics-Based -** Metrics-based approaches calculate and contrast various software metrics of two portions of code to ascertain whether or not they are clones. These techniques compare these metrics rather than the codes themselves. When two code fragments have comparable related metrics values, they are deemed to be clones.

- **Learning-Based -** Using learning-based approaches, the code fragments are used to extract features. A machine learning model is then trained using these features in order to identify clones. These methods of learning can be unsupervised or supervised. These methods have problems with scalability, much like metrics-based methods.

In this paper, we have compared various research papers. These are research as well as literature based papers. We have presented a detailed related work, comparison of various clone detection approaches. Further the paper is organized as: Section II represents the related work. In section III, we have compared various techniques in tabular form for easy understanding. Finally, section IV concludes this research.

II. RELATED WORK

We have reviewed various research papers relating to software clone detection. The scientists in [1] shared their experiences to detect clones in software. The authors used three different tools for this purpose. Various taxonomies and definitions are given

about software clones. In this research, the impact of clones is mapped for software quality metrics. In [2], different types of clones and their factors are explained. Various matching techniques are detected and reported. Also, drawbacks and benefits are summarized in this research.

The authors [3] discuss the main sources of code clones and reason behind it. Also, the negative effects behind the code duplication are highlighted. Various existing code clones techniques are analyzed. This research summarized the open problems about software code clones. A basic introduction to clone detection is given in [4] along with benchmark terminology. In [6], authors give introduction about software clones, review clone taxonomies, detection techniques and evaluation of various tools available for clone detection. The authors discuss various applications of clone detection techniques and their effects in other domains of software engineering. The scientist also identified open problems concerning to clone detection approaches.

In [7], scientists said that research on software code clones has been executed over a decade. The research shows that software systems may have 9-17% duplicate code. In this research, various tools have been analyzed and discussed. Different techniques such as token-based, Abstract Syntax Tree-based and text-based approaches are used. In this paper, a hybrid approach of textual and a metrics-based concept has been proposed and the results and effects are explored.

The paper [8] proposed a supervised learning model called CDLH, to detect functional code clones. CDLH learns hash codes by following lexical analysis and syntactical information. The approach uses deep learning features to detection functional based code clones. A deep learning based concept to detect clones is introduced in [9] and the research is followed by token-based approach. The model has been given a name as CCLearner. The tree-matching algorithm was also explored. The paper [10] proposes a method by following Cyclomatic complexity and Halstead measures. For exact match refactoring, slicing techniques were used. Code smell detected and analyzed using specific metrics. The experiments were carried out on various modules of GanttProject.

In [11], refactoring, design pattern, and clones classification schemes are explained. This research also discusses the aim behind cloning patterns instead of abstractions. Here, various code duplication and usage patterns are observed. In [12], an abstract syntax tree-based model was designed to detect near miss clones. The proposed approach is capable to detect exact and near miss clones patterns. The proposed method is successful to detect in complex language constructs with the help of ASTs. The proposed method is better than previous one.

A token-based approach was proposed in [13] and it is named as RFT (Repeated Tokens Finder). This technique incorporates a suffix array based algorithm for clone detection. This approach is customizable and locates methods boundaries. The proposed RTF method is integrated into Clone-Miner tool for similarity patterns. A new algorithm is presented in [14] for detecting similarities in software systems. This method is relied on the notion of the function. The proposed method produces effective results for insertion, detection of code blocks.

III. Comparison and Analysis

The analysis of various researches is also shown in Table 1.

Table 1. Comparison of Various Researches

Ref. No.	Methods Used	Limitations	Practical Implications
[1]	Exact and Replace Method, Exact Class	Refactoring has more disadvantages, Extracting Classes increases dependencies, Removal of code and new code smells not detected, different tools find different types of clones.	Various clone detection tools yield varying outcomes as a result of employing distinct detection methodologies. The elimination of redundant code leads to enhancements in the stability, cohesion, and complexity of the system. The act of refactoring proves beneficial in enhancing the quality of particular classes. The examination of the existence of fresh code smells subsequent to the refactoring of clones. In certain instances, refactoring may not be advantageous, particularly when it engenders an increase in dependencies.
[2]	Not Mentioned	Complexity of large systems leads to code copying. Some programming languages have less support for reusability of code. Forking and Templating are short-term reused mechanisms.	The identification of research gaps in the field of software cloning and clone detection is crucial. It is imperative to raise awareness about the numerous benefits that come with effective software clone management. Additionally, there is a pressing need to emphasize the utilization of semantic and model-based techniques in clone detection. Instead of simply removing clones, it is recommended to establish proper clone management facilities. Furthermore, this study offers valuable recommendations for future research endeavors in the realm of clone detection.
[3]		None of the clone detection methods currently in use are suitable for use in industrial settings, and their accuracy and thoroughness are both lacking.	Code replication poses a significant challenge within the realm of software development. The presence of identical code fragments exacerbates the complexity

	Not Mentioned		of software maintenance and support. Current methodologies for detecting code clones display certain limitations in relation to their precision and comprehensiveness. Unanswered inquiries persist concerning the detection of code clones and its effect on the quality of software.
[4]	Pre-Validated Clones	Issues with developing a clone benchmark that has been verified for comparison. Several hundred prospective clones must be manually checked. Validated clones that were made artificially might not accurately represent clone detecting tools' recall. Difficulties in precisely assessing the recall and precision of clone detection instruments.	It is now feasible to conduct an impartial assessment of clone detection tools. The difficulties encountered in constructing benchmarks for clone detection have been explicated. Upcoming endeavors ought to prioritize the establishment of a firm objective foundation for evaluating clone detectors.
[6]	Not Mentioned	Restrictions brought about by developers and programming languages.	Provides a comprehensive survey on software clone detection research. Describes clone terms, taxonomies, detection approaches, and experimental evaluations. Points out open problems for further research in clone detection. Assists potential users in selecting the right tool or technique. Identifies avenues for future research and interesting combinations of techniques.
[7]	LWH Approach	Limitations in locating either functional or structural information. Human mistake may occur in the Clones Manual evaluation.	The proposed approach known as LWH exhibits enhanced precision and reduced comparison cost. It possesses the capability to accurately identify method-level clones with a high degree of precision and recall. In the future, efforts will be made to augment the technique specifically for web static pages and clone elimination. Furthermore, the tool has the ability to preserve prior clone detection outcomes in order to decrease processing time.
[8]	CDLH	32-bit hash code with limited room for experimental results. The value of the threshold is not stated. No mention of restrictions in the context at hand.	The present study considers the issue of identifying software functional clones. It introduces a novel supervised deep feature learning framework, referred to as CDLH. CDLH leverages both lexical and syntactical data in order to quantify the functional resemblance among code fragments. Remarkably, CDLH surpasses current cutting-edge methods in software functional clone detection.
[9]	Token Based, Deep Learning, AST	Not Mentioned	CCLEARNER is an approach for clone detection that is based on deep learning and does not rely on specialized algorithms. In comparison to existing token-based approaches, CCLEARNER is able to detect a wide range of clones with a high level of precision and recall. Additionally, when compared to tree-based approaches, CCLEARNER is able to efficiently detect clones with a high level of precision. The effectiveness of CCLEARNER is influenced by factors such as the number of hidden layers and iterations in the deep neural network (DNN). Indicators of code clones include the similar usage of reserved words, markers, type identifiers, and method identifiers. Code clones are more likely to utilize divergent operators, literals, qualified names, and variable identifiers. In

			future research, more advanced methods will be explored to match similar terms without incurring runtime overhead. Furthermore, other machine learning techniques will be investigated for the purpose of clone detection. It is also worth noting that deep learning can be employed to automatically extract features from known code clones.
[10]	Cyclomatic Complexity, Halstead Measures	Code smell definitions provided by Fowler are too vague to be put into practice. The lack of clarity in code smell definitions affects how well they are identified. It is a low-probability event when a code element has a code smell. There is a significant skew in the distribution of "smelly" and "non-smelly" code items. The experiments' limited size makes them less statistically significant.	Different software engineering tools for detecting code smells do not reach a consensus in their findings. Tools for identifying code smells have the capability to pinpoint problematic sections of code. These tools serve a valuable purpose in evaluating the evolution of software. Among the various tools considered, only one possesses the ability to perform refactoring. In order to effectively eliminate code smells, these detection tools must possess the capability to support their removal. Additional data is necessary to further refine experiments and establish a benchmark dataset. Manual validations are imperative in order to facilitate a more thorough comparison of results. The definitions of code smells can be enhanced to optimize the techniques employed for detection. The relationships between code smells and other structural elements within the code base should be thoroughly analyzed.
[11]	Cloning Patterns	Not Mentioned	Code duplication can be employed in a constructive manner within software systems. The utilization of duplication patterns can be regarded as a logical design choice. It is imperative to devise tools that assist in making informed decisions pertaining to maintenance and refactoring. The incorporation of cloning patterns necessitates the contemplation of long-term consequences. It is of utmost significance to identify additional cloning patterns and ascertain their rates of success.
[12]	Abstract Syntax Trees	Prior research was restricted to identifying precise textual matches or close calls on entire function bodies. As size grows, the quantity of unintentional code fragments decreases significantly.	Software maintenance costs can be reduced by identifying and eliminating code clones. The technology given has the ability to identify clones on a big scale and detect near-misses. The technique can identify clones in any language construct and is very easy to apply. To get rid of the clones without interfering with program functionality, macros can be calculated. Tools for detecting clones have the potential to support domain analysis.
[13]	Repeated Tokens Finder	One problem is the detection of very short clones. It's possible that language-level approaches won't be enough to bring all clones together.	Offers a unique, versatile tokenization method for token-based clone detection. Adds clone detecting functionality to the Clone Miner tool. Assures code reuse and program comprehension that is superior to code clones. Gives consumers the option to specify the token-based minimum size for clones. Offers a lower memory use than CCFinder.
[14]	Notion of Functions, Pattern matching	False-positive results because of how long the leaf functions are. There is no indication of a lower bound on the	Opposition to obfuscator techniques such code shifting, in-lining, and outlining. If the function calls are not taken into account,

		length of leaf functions.	precision is decreased. A maximum threshold for leaf function lengths needs to be established because false positives can occur. Taking function call order into account while calculating new function comparison metrics. To cut down on false positives, data-flow analysis can be done. Function calls with multiple function calls as parameters are not yet handled by the method. A preliminary method for unfolding constructed calls by introducing temporary local variables. Partial outlining graphs provide a visual representation of the outcomes of outline procedures. It's necessary to research more user-friendly result rendering tools.
[15]	Active Testing using Rule Evaluation	Not Mentioned	Uses clone detection to locate bug pattern matching code snippets. Uses interactions between code fragments to identify high- and low-potential issues. It is imperative that testing efforts be localized to possible concurrency problems. Minimizes search space for software testing that is done concurrently. To finish the active testing process, further labour and experimentation are needed.

IV. CONCLUSION

The field of software engineering research is more broader. Research is conducted on a variety of trending subjects, such as efforts estimates, clone detection, bug detection, and quality prediction. Software clone detection can reduce development time and aid in the creation of effective software. Although the clones are helpful for reusing code, the copy-paste technique could make the code less stable and reliable. The performance of the entire system is impacted by superfluous duplicate code in addition to the software itself. Thus, the methods for detecting software clones are included in this research. Token-based code representation, abstract syntax tree-based models, CDLH, NiCad, LWH models, PMD, CodePro, and other methods have been introduced by scientists as tools and strategies for clone detection. Many studies are examined and contrasted in this paper. The limitations and practical implications are also mentioned in this research.

In future, it is a need to develop a machine learning model to detect software clones as the accuracy is not up-to the mark in case of existing models.

References

[1] Fontana, F. A., Zanoni M., Ranchetti, A. and Ranchetti, D. 2013. Software Clone Detection and Refactoring. ISRN Software Engineering, 2013: 1-9.

[2] Rattan, D., Bhatia, R., and Singh, M. 2013. Software Clone Detection: A Systematic Review. Information and Software Technology, 55: 1165-1199.

[3] Akhin, M., and Itsykson, V. 2010. Clone Detection: Why, What and How?. IEEE, 36-42.

[4] Roy, C. K., and Cordy, J. R. 2018. Benchmark for Software Clone Detection: A Ten-Year Retrospective. IEEE, 26-37.

[5] Saini, V. P. S. 2018. Towards Accurate and Scalable Clone Detection using Software Metrics. PhD Thesis, University of California, Irvine.

[6] Roy, C. K., and Cordy J. R. 2007. A survey on Software Clone Detection Research. School of Computing, Queen's University at Kingston, Ontario, Canada.

[7] Kodhai, E., and Kanmani, S. 2014. Method-Level Code Clone Detection through LWH (Light Weight Hybrid) Approach. Journal of Software Engineering Research and Development, 2(12): 1-29.

[8] Wei, H. H., and Li, M. 2017. Supervised Deep Features for Software Functional Clone Detection by Exploiting Lexical and Syntactical Information in Source Code. Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, 3034-3040.

[9] Li, L., Feng, H., Zhuang, W., Meng, N., and Ryder, B. 2017. CCLearner: A Deep Learning-Based Clone Detection Approach. International Conference on Software Maintenance and Evolution, IEEE, 249-260.

[10] Fontana, F. A., Braione, P., and Zanoni, M. 2011. Automatic detection of bad smells in code: An experimental assessment. Journal of Object Technology, 1-38.

[11] Kasper, C., and Godfrey, M. W. 2006. "Cloning Considered Harmful" Considered Harmful. 13th Working Conference on Reverse Engineering, IEEE.

[12] Baxter, I. D., and Anna, M. S., and Bier, L. 1998. Clone Detection Using Abstract Syntax Trees. International Conference on Software Maintenance.

[13] Basit, H. A., Puglisi, S. J., and Smyth, W. F. 2007. Efficient Token Based Clone Detection with Flexible Tokenization. Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 513-516.

[14] Chilowicz, M., Duris, E., and Roussel, G. 2009. Finding Similarities in Source Code through Factorization. Electronic Notes in Theoretical Computer Science, 238: 47-62.
[15] Jelbert, K., Bradbury, J. S. 2010. Using Clone Detection to Identify Bugs in Concurrent Software. International Conference on Software Maintenance.