Review on Software Clone Management

Prof. Chavi Ralhan[1], Sumit Kumar Kushwaha[2], Gurpreet Singh[3], Saurabh Kumawat[4], Asif Ali[5], Midde Sai Kumar Reddy[6]

*[1,2,3,4,5,6]Department of Computer Science and Engineering, Lovely Professional University, Phagwara, Punjab, India

Abstract— **Copy and pasting of existing code or a fragment of code from one software to another instead of writing from scratch is known as software cloning or code cloning. Software developers find it time consuming and easy but with shortcuts their comes hurdles in long- run. Things becomes more complex when the development reaches advance stages and as a result it gets difficult to maintain the software. Maintenance which is among the most expensive part in software development, studies have been done in the past and recent which speaks of the pros and cons of cloning, this paper will be summarizing different aspects of code cloning.**

Keywords-Software cloning, code cloning, maintenance.

I. Introduction

Cloning is a common practice in both the software and research industries. Known code is commonly reused from publicly accessible tools, which encourages code cloning. It creates a difficult software maintenance issue. It takes a significant amount of time and effort, which raises the cost of developing any tool, software, or other product. Code cloning refers to duplicate or nearly duplicate code segments. The concept of code clones is used for a variety of reasons, including reusing code through 'code-and-paste' and others that make it difficult to modify source files consistently. If a bug is discovered in a single code block, the entire cloned block must be modified, making maintenance more difficult as the system grows larger. Fowler defined clone as a duplicate code which is an example of bad smell [11].

The primary cause of software maintenance difficulties is code clones, which does not imply that code clones are only harmful to software development. For software development, code clones can be both beneficial and disastrous. As a result, before refactoring or removing clones, they should be investigated. Clones are classified into two types: syntactic clones and semantic clones. Type-1, Type-2, and Type-3 clones are included in syntactic clone [12] whereas Type-4 clones are included in semantic clone. Type-1 clones are code fragments that are identical except for differences in white spaces, comments, layout, and so on. Type-2 clones are code segments that are similar to each other but have minor differences in their names, such as renaming of identifiers, variables, and so on. As a result, Type-2 clones are also referred to as renamed clones. Type-3 clones are near-miss clones that have similar code fragments but with some additional statement addition and deletion. Semantic clones (or Type-4 clones) are clones that perform a similar task but have different syntactic structures.

II. LITERATURE REVIEW

There have been various proposed and review papers advocated by various researchers who worked in this area of cloning and implemented their proposed methodologies using various techniques and tools in the literature survey.

- Beginning in 2006, Rachel Edita Roxas [13] proposed a technique that automatically detects cloning in programmes with the help of a tool called Jplag.
- Stefan Bellon [14] published a comparison and evaluation of clone detection tools in 2007, stating that token and text-based techniques work similarly, and Merlo and Baxter's AST tool has higher precision but higher costs. They also claimed that Krinke's PDG-based tool does not detect type 3 clones, also known as Near Miss clones, efficiently.
- In the year 2008, Cory J. Kapser [15] emphasised negative clone characteristics known as code smells. They claimed that these code odours have a negative impact on programme design and maintenance. A well-known review paper published in 2009 by James R. Cordy and Chanchal K. Roy [16] was solely focused on clone detection techniques and tools. They also provided reviews on clone detection methodology or processes involved in clone detection.
  Tung Thanh Nguyens [17] presented another technique in the same year, based on scalable and incremental clone detection using the Clemanx tool, which represents code fragments as sub trees of ASTs and detects similar clones using a distance-based clustering problem.
- Many techniques were presented by various researchers in 2010. Perumal. A [18] proposed one technique out of those listed. This technique made use of metrics to extract similarity in previously detected software clones. Following clone detection, two functions were applied to the detected clones: first, it could create clusters in which the detected clones were grouped together, and then cluster ranking was used on them by arranging them in ascending order. Another review paper emphasised the comparison of various plagiarism detection tools such as Jplag.
  Finally, it was determined that these tools did not reveal the presence or absence of plagiarism, but rather defined the degree of similarity among the code.Moss, Marble, Gplag, Sim, and others on the basis of performance and sensitivity.
- G.Anil Kumar proposed a technique in 2011 that used a light weight clone detection technique, which is also known as a hybrid technique because it combines two techniques, metrics and text-based, with refactoring support.
- Saif Ur Rehman and Kamran Khan [19] proposed a technique to detect clones using the LSC-Miner tool in 2012. LSC-Miner detects clones in multiple languages using a token-based technique. It is a useful tool that can also be used on large software systems. Another technique that came into play the same year was semantic clone detection, which was

based on the JSC Tracker [20] tool and only worked on java code. It also made use of the open source database software DSpace and the reference management software JabRef. In the same year, Deepak Sethi [21] improved clone detection. He applied a data mining technique to data sets. This proposed method makes use of the Solid SDD tool, which provides a better visualisation of clone detection and has been enhanced with the addition of a graphical user interface.

In the same year, Tahira Khatoon [22] proposed a technique for detecting clones in Java using AST. A hash function was used in this technique to partition ASTs into smaller sub trees. Subtrees were also compared to see how similar they were, and subtrees with values greater than the threshold value were considered clones. Another technique presented by Priyanka Batta [23] was dubbed the Hybrid technique because it combined metric and text-based techniques and could detect clones in C or C++. This application runs on the Windows platform. This work can be improved further by creating a tool that detects clones in multiple languages. In the same year, Rupinder Kaur and Prabhjot Kaur [24] attempted to improve performance and efficiency by comparing two token-based techniques known as CC-Finder and PMD, which were compared using three types of metrics, namely file metrics, line metrics, and clone set metrics. They concluded that there is no such tool that detects all clones efficiently; rather, each tool has its own set of strengths and weaknesses that determine its usefulness in detecting clones.

- Dhavleesh Rattan [25] published a review paper in 2013 that was solely about the cloning background. It was about the various clone detection techniques and methodology that must be used when locating clones, as well as the benefits and drawbacks of doing so. Furthermore, it was about clone detection in other key areas such as clone analysis, cloning management, and so on. Kanika Raheja [26] proposed the other technique the same year. The technique used here was a metric-based technique that only worked with the Java programming language. The source code was not applied directly to java source code (.java).
- Another technique proposed by Harpreet Kaur and Rupinder Kaur [27] in 2014 was metric-based. This method detects cloning not only in programming languages but also in web applications. This also returned higher level clones, referred to as directory clones in Java. In one of the other clone detection methods, neural networks and SIMCAD were used to detect clones. This technique was one of the hybrid techniques because it combined two techniques, text-based and data-mining-based.
- Manpreet Kaur and Madan Lal [28] presented one of the most recent proposals in 2015, which was a hybrid technique combining metric-based and text-based techniques. Their proposal focuses on detecting type-1, type-2, and type-3 clones. They also improved their technique with the clone removal strategy. In the future, they can improve their tool to make it more efficient and capable of locating type-4 clones. Shashank Prabhakar [29] proposed a clone detection technique in 2016 that focuses on Type-1 and Type-2 clones.
- Li et al. [30] used deep learning to create CCleaner, a token-based clone detection tool. CCleaner first extracts from the source code a set of features that can be used to easily identify clone pairs by generating tokens with a lexical parser. These tokens are divided into eight categories to generate eight distinct features, which serve as input to the deep neural network and produce output in the form of a clone and non-clone pair. This tool can detect syntactic clones with high precision and recall, but it cannot detect semantic clones with the same level of accuracy.
- In 2018, Ghosh and Kuttal [31] proposed using source code comments to detect semantic clones. This method combines the concept of comments with LDA (Latent Dirichlet Allocation) to produce a good result with precision of 84% and recall of 94%. Almori and Stephan [32] presented an introductory method for detecting semantic clones using the slicing concept. To detect semantic clones, the authors used SrcSlice, a slicing tool that determines the slice profile of each variable. The clone pairs are identified based on the similarity of each variable's slice profile.
- In 2019, Jiang et al. proposed the tool EqMiner [33], which is based on input-output behaviour. If two code fragments have the same input-output behaviour, they are returned as a clone pair. Because this tool is scalable, it is also suitable for large projects. However, it only works with the C programming language.
- In 2020, Using a probabilistic software model (PSM), Thaller et al. [34] proposed a semantic clone detection technique. In comparison to other tools such as Oreo, precision, and recall, the proposed approach yields superior results. The authors use PSM to determine equivalence behaviour, also known as similarity evaluation. This similarity assessment is then used in the software to detect semantic clones.

III. Code Clone Analysis

a.) **<u>Definitions on Code Clone</u>**

Investigates in over a significant time span tells that there is no exact definition for code clone, various explores give distinctive results.it relies upon boundaries, strategies that are being utilized for discovery gives various outcomes.

In general, cloning is the result of different processes used to achieve an identical copy of something which is already existing. In software industry fragments of codes are copied to achieve target software. According to Ira Baxter ‘‘Clones are segments of code that are similar according to some definition of similarity” [2]. Duplication of code or code cloning is a software reuse form as defined by Roy and Cordy [5].

Baker after experimenting on some sample programs presumed that dependent on exact and parametrized matches code contracts by 14% and 61% individually [1].

In extensive software programming frameworks 20-30% comprises of cloned code [1].

There are essentially two sorts of similitudes between two code sections. Two code fragments can be comparable

dependent on the closeness of their program text or they can be comparable in
their functionalities without being literarily comparable. The code fragments can have functional and textual similarity.

Textual Similarity have following types of clones:

- **Type I:** Indistinguishable code parts aside from varieties in whitespace (might be moreover varieties in format) and comments.

- **Type II:** Fragments that are Structurally or syntactically indistinguishable with the exception of varieties in identifiers, literals, types, design and comments.

- **Type III:** Duplicated parts with additional modifications. Proclamations can be changed, added or eliminated notwithstanding varieties in identifiers, literals, types, design and comments.

Functional Similarity:

- **Type IV:** These clones are the consequence of same semantic execution between at least two codes, yet that doesn't show that the specific piece is duplicated from a current issue arrangement, potential outcomes expresses that they may be carried out utilizing diverse syntactic variations.

b.) Advantages and Applications of Detecting Code Clones

Sometimes software development requires taking references from other existing similar or comparable software, in such cases reusing code fragments for achieving similar requirement is a productive way.
Developers sometimes have restricted information hence, codes or solutions of pre-existing problems helps in significant way in accessing legitimate knowledge about the problem requirement like what data structure is required to be used, about instances variables, language, syntax etc.
To distinguish pernicious programming clone location methods can assume an imperative part. By contrasting one noxious programming with another, it is feasible to discover the proof where parts of the one programming framework match portions of another [6].
Reusing pieces of codes additionally helps in decreasing the size of the program, in a large portion of the cases designers are assigned with cutoff times so this code clones helps the engineer from multiple points of view as it upgrades understandability.

c.) Drawbacks of Code Duplication

- Increase in maintenance effort is an unfavorable effect impact of cloning as we need to monitor all the replicated parts. In the event that any change is made in one reused area, it should be reflected altogether the reused areas to eliminate irregularities [4].
- On the off chance that a code portion contains a bug and that section is reused by adapting and pasting without or with minor variations, the bug of the first fragment may stay taking all things together the pasted fragments in the framework and along these lines, the likelihood of bug may rise significantly in the framework [7,8].
- Cloning may result in poor architecture, a lack of inheritance structure, or abstraction, as a result, reusing parts of the implementation in future objects becomes difficult, it also has a detrimental effect on the maintainability of the system of the computer application [9].
- It is very common for the developer who created the original system to not be the one keeps it up to date. Furthermore, replicated codes not only makes the design more complicated, but it also makes it more difficult to understand, making changes and modification more difficult. Over time the program can become so complex that even small improvements would be difficult to implement [1,3].
- The size of the software system grows as a result of code cloning, placing a strain on system resources [3,10], in terms of compilation/execution time and space requirements, it degrades overall performance.

IV. CODE DETECTION PROCESS

A clone detector must look for code fragments with a high degree of similarity in the source text of a system. The main issue is that it is unknown which code fragments can be found multiple times. As a result, the detector must essentially compare every possible fragment with every other possible fragment. Because such a comparison is computationally expensive, several steps are taken prior to performing the actual comparison to reduce the domain of comparison. Furthermore, after identifying potential cloned fragments, additional analysis and/or tool support are required to detect actual clones. This section attempts to provide an overview of the clone detection process.

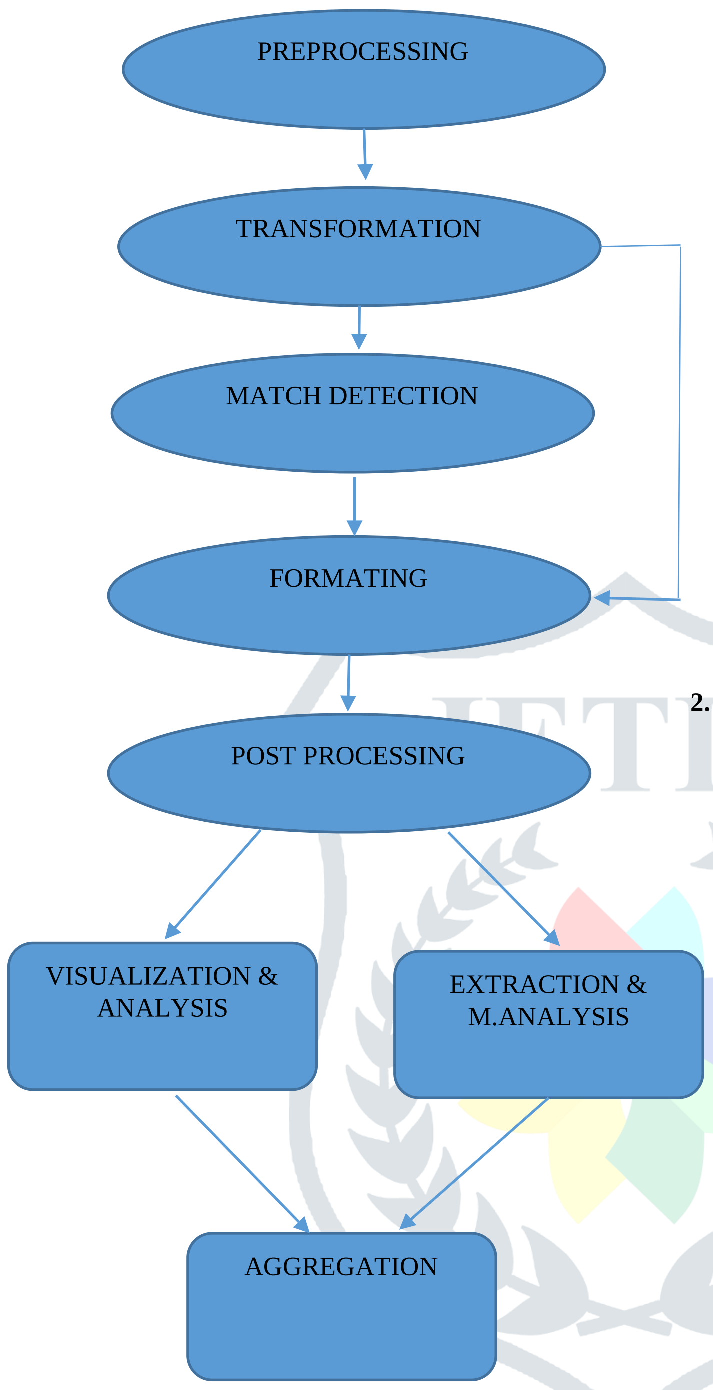

Figure 1. Process of detecting clones

1. **Preprocessing**: At the start of any clone detection method, the targeted source code is partitioned and the domain of the comparison is determined. This phase's main goals are as follows:
   - **Remove any uninteresting elements**: This phase filters out all source code that is uninteresting to the comparison phase. Partitioning is used to separate different languages in embedded code (for example, SQL embedded in Java code or Assembler embedded in C code). Similarly, generated code (e.g., LEX- and YACC-generated code). Before proceeding to the next phase, parts of the source code that are likely to generate a large number of false positives (e.g., table initialization) can be removed.
   - **Determine source unit:** After the uninteresting code is removed, the remaining source code is partitioned into a set of disjoint fragments known as source units. These are the largest source fragments that have direct clone relationships with each other. Because such units do not maintain any order in the source code, matching units cannot be aggregated beyond the border of such source units.
   - **Determine comparison units**: Depending on the comparison function of a method, source units may need to be further partitioned into smaller units. For comparison, source units, for example, can be subdivided into lines or even tokens. The syntactic structure of the source unit can also be used to derive comparison units. An if-statement, for example, can be further divided into conditional expression, then, and else blocks. The comparison units are arranged within the context of their corresponding source units. The comparison function relies on this ordering.

2. **Transformation**: The source code's comparison units are transformed to another intermediate internal representation for ease of comparison or extraction of comparable properties. This transformation can range from very simple, such as simply removing whitespace and comments, to extremely complex, such as generating PDG representation and/or extensive source code transformations. From such intermediate representations, metrics-based methods typically compute an attribute vector for each comparison unit.

There are two ways to accomplish this transformation: extraction and normalization.

I. **Extraction:** It is further subdivided into three subcategories: tokenization, parsing, control, and data flow analysis [35].
   - **Tokenization:** In this method, source units are converted into tokens using lexical protocols or procedures, and these tokens are arranged in token sequences for comparison after blank spaces and comments are removed.
   - **Parsing:** In this case, the entire source code is parsed to generate an AST (Abstract Syntax Tree), and then the source units from the ASTs that are required are displayed in the form of sub trees. These sub trees must be compared in order to identify clones.
   - **Producing PDG**: PDG (Program Dependency Graph) generated tools create PDG graphs in which nodes represent statements and edges represent data and control dependency in this approach. Subgraphs of PDGs are compared to lay out a comparison.

II. **Normalization:** This is an optional step for removing differences caused by comments, whitespaces, and so on. This can be accomplished in a variety of ways, including normalizing identifiers, which involves replacing all identifiers in source code with a single identifier, pretty-printing, and so on [35].

3. **Match Detection**: The transformed code is then fed into a suitable comparison algorithm, which compares transformed comparison units to find a match. The order of the comparison units is used to sum adjacent similar units to form larger units. All comparison units belonging to a source unit are aggregated for fixed granularity clones. Aggregation, on the other hand, is continued for free granularity clones as long as the aggregated sum exceeds a given threshold for the number of aggregated comparison units. This ensures that the aggregation process is carried out until the largest possible group of comparison units is discovered.

The output includes a list of matches in the transformed code that represent clone relations as clone pairs, clone classes, and clone families. Hashing, suffix trees, and other comparison methods are examples.

4. **Formatting:** The clone pair list obtained with respect to the transformed code is converted to a clone pair list with respect to the original code base during this phase. Normally, each location of the previous phase's clone pair is converted into a line number on the original source files. A nested-tuple can be used to represent a clone pair in general.

5. **Post-processing**: During this stage, false positive clones are removed using manual analysis and/or a visualization tool.

- **Manual analysis**: Following the extraction of the original source code, the raw code of the clones of the clone pairs is subjected to manual analysis. False positive clones are filtered out during this stage.
- **Visualization**: A visualization tool can be used to visualize the clones using the obtained clone pair list. A visualization tool can help to accelerate the manual analysis process for removing false positives and other related analysis.

6. **Aggregation**: This is the final stage of the clone detection procedure. It refers to proper data analysis and data reduction. Clone classes and clone families are formed by combining the detected clone pairs [4]. Clone pairs are aggregated to clusters, classes, cliques of clones, clone groups, and so on in order to reduce the amount of data or perform certain analyses.

V. TECHNIQUES FOR DETECTING CLONES

There are various techniques for code detection:

- Tree based
- Token based
- (Abstract syntax tree) AST based
- Program Dependency Graph based
- Metric based
- Hybrid based

Table 1.Provides a detailed description of various techniques based on various parameters, as well as a comparison of them

Parameters	**Techniques**					
	Text Based	**Token Based**	**Abstract Syntax Tree (AST) Based**	**Program Dependency Graph (PDG) Based**	**Metric Based**	**Hybrid**
Portability (Ability to run on multiple platforms)	High	Medium	Low	Low	Metrics dependent	Depends on type of technique used
Efficiency (Quality of results)	High	Low	High	High	High	High
Integrality(Level of difficulty involved in integrating the technique in current environment)	Low	High	Low	Medium	Medium	Medium
Transformation (Method of preprocessing	Removes white space and comments	Tokens are generated from source code	AST is generated from source code	PDG is generated from source code	Metrics value are calculated after AST is generated from source code	Depends on the hybrid technique
Comparison Based (Type of input taken by clone detection technique)	Lines of code	Token	Nodes of tree	Nodes of program dependency graph	Metrics value	Depends on the hybrid technique
Computational Complexity(upper bound on time taken for clone detection)	Depends on algorithm	Linear	Quadratic	Quadratic	Linear	Depends on the hybrid technique
Refactoring Opportunities (Refactoring can be done easily or not)	Good for exact matches	Good	Good for refactoring as it finds syntactic clones	Good for refactoring	Manual inspection required	Depends on the hybrid technique
Representation (How source code will be represented)	Normalized source code	In the form of tokens	In the form of abstract syntax tree	In the form of program dependency graph	Set of metrics values	Depends on the hybrid technique
Language Dependency (Language targeted by clone detection technique)	Adaptable	It needs a laxer but no syntactic knowledge required	It needs a laxer but no syntactic knowledge required	Parser required	Parser required	Depends on the hybrid technique
Type of Clone Detected	Type 1,2 and 3	Type 1,2 and 3	Type 1,2,3 and 4	Type 1,2,3 and 4	Type 1,2,3 and 4	Depends on the hybrid technique

Output Type	Clone pairs and Clone Classes	Clone pairs and Clone Classes	Clone pairs, Clone Classes, AST nodes	Clone pairs and Clone Classes	Clone pairs, Clone Classes, metrics value	Depends on the hybrid technique

VI. Conclusion & Future Work

In this paper, we led a writing survey on code clone examination to improve programming/software maintenance measures. In figure 1, process of detecting clones along with their elaborated definition has been discussed later we have discussed several code clone detection techniques along with their detailed description based on various parameters and their comparison which is proposed by researchers to enhance code clone maintenance process in Table 1. We likewise talked about benefits and disadvantages of code clone. The aftereffects of this examination may fill in as a guide to possible clients of clone location methods, to help them in choosing the correct tool or procedure for their inclination.
Likewise, there isn't a lot research in the space of clone the executives and model clones. These territories can be investigated
research in the space of clone the executives and model clones. These territories can be investigated like detection of semantic clones, reliability of code clones and how efficiency can be more proficient if maintenance efforts are reduced.

References

[1] R. Koschke, Frontiers of software clone management, in: Proceedings of Frontiers of Software Maintenance (FoSM'08), Beijing, China, 2008, pp. 119–128

[2] B.S. Baker, On finding duplication and near-duplication in large software systems, in: WCRE'95: Proceedings of the 2nd Working Conference on Reverse Engineering, IEEE Computer Society, 1995, pp. 86.

[3] R. Koschke, survey of research on software clones, in: Duplication, redundancy, and similarty in software, Dagstuhl Seminar Proceddings, 2007, p. 24.

[4] S. Thummalapenta, L. Cerulo, L. Aversano, M.D. Penta, An empirical study on the maintenance of source code clone, Empirical Software Engineering 15 (1) (2010) 1–34.

[5] Y. Ueda, T. Kamiya, S. Kusumoto, K. Inoue, Gemini: Maintenance Support Environment Based on Code Clone Analysis, 8th International Symposium on Software Metrics, pages 67-76, June 4-7, 2002.

[6] T. Kamiya, S. Kusumoto, and K. Inoue, CCFinder: A multi-linguistic tokenbased code clone detection system for large scale source code IEEE Transactions on Software Engineering, 28(7):654-670, 2002.

[7] Ekwa Duala-Ekoko, Martin Robillard. Tracking Code Clones in Evolving Software. In Proceedings of the International Conference on Software Engineering (ICSE'07), pp.158-167, Minneapolis, Minnesota, USA, May 2007

[8] E. Merlo, M. Dagenais, P. Bachand, J.S. Sormani, S. Gradara, and G. Antoniol. Investigating large software system evolution: the linux kernel. In Proceedings of the 26th International Computer Software and Applications Conference (COMPSAC'02), pp. 421426, Oxford, England, August 2002

[9] 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

[10] John Johnson. Substring Matching for Clone Detection and Change Tracking. In Proceedings of the 10th International Conference on Software Maintenance, pp. 120-126, Victoria, British Columbia, Canada, September 1994

[11] Fowler, Martin, and Kent Beck (1999), "Refactoring: improving the design of existing code, Addison-Wesley Professional.

[12] Min H and Li Ping Z 2019 Proceedings of the 2019 3rd International Conference on Management Engineering, Software Engineering and Service Sciences - ICMSS 2019 (New York, New York, USA: ACM Press) pp 9–16 ISBN 9781450361897 URL http://dl.acm.org/citation.cfm?doid=3312662.3312707

[13] Rachel Edita Roxas, "Automation generation of Plagiarism Detection among students Plagiarism", In IEEE Transactions on Software Engineering, September 2006

[14] Stefan Bellon, Rainer Koschke, Giuliano Antoniol, Jens Krinke, and Ettore Merlo.Comparison and Evaluation of Clone Detection Tools. In IEEE Transactions on Software Engineering, Vol. 33(9): 577-591, September 2007.

[15] C.Kapser and M. Godfrey"Cloning Considered Harmful" Considered Harmfu". In WCRE, pp. 19 -28, 2006.

[16] Chanchal K. Roya, James R. Cordy, Rainer Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," Science of Computer programming, ELSEVIER, pp 470-495, 2009.

[17] Tung Thanh Nguyen, "Cleman X: Incremental Clone Detection Tool for evolving Software", ICSE'09, May 16-24, 2009, Vancouver, Canada 978-1-4244-3494-7/09@ 2009 IEEE.

[18] Kodhai.E, Perumal.A, and Kanmani.S, "Clone Detection using Textual and Metric Analysis to figure out all Types of Clones", Proceedings of the International Joint Journal Conference on Engineering and Technology, pp. 99-103, 2010.

[19] Saif Ur Rehman, Kamran Khan, "An Efficient New Multi-Language Clone Detection Approach from Large Source Code," International Conference on Systems, Man, and Cybernetics, IEEE, pp 937-940, 2012.

[20] Rochella Elva,Gary T. Leavens, "A Semantic Clone Detection Tool for Java Code," March 2012.

[21] Deepak Sethi, Manisha Sehrawat, "Detection of code clones using Datasets," IJARCSSE, pp 263-268, July 2012.

[22] Tahira Khatoon, Priyansha Singh, Shikha Shukla, "Abstract Syntax Tree Based Clone Detection for Java Projects," Journal of Engineering, IOSR, pp 45-47, Dec 2012.

[23] Priyanka Bhatta, "HYBRID TECHNIQUE FOR SOFTWARE CODE CLONE DETECTION," International Journal of Computers and Technology, pp 97-102, April 2012.

[24] Rupinder Kaur, H. K, "Evaluation of Token Based Tools On The Basis Of Clone Metrics" International Journal of Advanced Research in Computer Science and Electronics Engineering, 2012.

[25] Dhavleesh Rattan, Rajesh Bhatia, Maninder Singh, "Software Clone Detection: Systematic Review," Information and Software Technology, ELSERVIER, pp 1165-1199, 2013.

[26] Kanika Raheja, Rajkumar Tekchandani, "An Emerging Approach towards Code Clone Detection: Metric Based Approach on Byte Code," IJARCSSE, Vol.3, May 2013.

[27] Rupinder Kaur, Harpreet Kaur, "Clone Detection in Web Application using Clone Metrices" International Journal of Advanced Research in Computer Science and Software Engineering, July 2014.

[28] Manpreet Kaur, Madan Lal, "Review on various code clone detection Techniques", Computer Science and Software Department, Punjabi University, May 2015.

[29] Shashank Prabhakar, Sonam Gupta "A review on Code Clone Detection and implementation", Computer and Communication Engineering, February 2016.

[30] Li L, Feng H, Zhuang W, Meng N and Ryder B 2017 Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017 ISBN 9781538609927

[31] Ghosh A and Kuttal S K 2018 2018 IEEE Symposium on Visual Languages and HumanCentric Computing (VL/HCC) (IEEE) pp 315–317 ISBN 978-1-5386-4235-1 ISSN 19436106 URL https://ieeexplore.ieee.org/document/8506550/

[32] Alomari H W and Stephan M 2018 2018 IEEE 12th International Workshop on Software Clones (IWSC) (IEEE) pp 58–59 ISBN 978-1-5386-6430-8 URL http://ieeexplore.ieee.org/document/8327320/

[33] Gabel M, Jiang L and Su Z 2008 Proceedings of the 13th international conference on Software engineering - ICSE '08 (New York, New York, USA: ACM Press) p 321 ISBN 9781605580791 ISSN 02705257 URL http://portal.acm.org/citation.cfm?doid=1368088.1368132

[34] Thaller H, Linsbauer L and Egyed A 2020 2020 IEEE 14th International Workshop on Software Clones (IWSC) (IEEE) pp 64–69 ISBN 978-1-7281-6269-0 (Preprint 2001.07399) URL https://ieeexplore.ieee.org/document/9047635/

[35] Chanchal K. Roya, James R. Cordy, Rainer Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," Science of Computer programming, ELSEVIER, pp 470-495, 2009.