International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 11
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 December 2023*

Mutation-Based Detection of Code Clones in Open-Source Software

Harpreet Kaur

Department of Computer Science and Engineering

Punjabi University, Patiala, INDIA

harpreet.ce@pbi.ac.in

Abstract— Numerous tools, templates, and frameworks exist in the literature to detect code clones from source code. However, every tool and method exhibits performance measures as per the application. In this paper, we presented a mutation-based analysis of our already proposed framework to detect function clones. Mutants were injected into JhotDraw open-source code to verify the accuracy of the proposed template. Clone detection has been performed by computing accuracy. Our experiments demonstrated confidence in the accuracy of our Mutation Injected framework. The schema provided will help in building repositories of function clones. In this research, function granularity has been chosen because functions are the most reusable part of any software.

Keywords- Granularity; Template; Function Clones; Signature; Complexity;

I. INTRODUCTION AND RELATED WORK

The process of identifying similar code portions in one or more source files is known as clone detection. In a software project, having clones makes maintenance difficult. Therefore, one of the most significant tasks in software quality enhancement is clone detection. After being incorporated in a software project, clones evolve. Finding altered clone components could improve the chances of refactoring and latent mistake identification as part of the clone detection process. Reusing code and copying code fragments—a process called "Code Cloning"—has both beneficial and detrimental effects on software development. Cloning is the easiest approach to accelerate software development, which is its main benefit when it comes to software reuse. It can provide a development team with a quick start, and often a rapid solution to the problem. Without negotiating software ownership, customization can be achieved by copying processes with a function that is similar to the one that is needed and then changing that code. However, from the perspective of program analysis, any bug that spreads throughout reusable code will result in clones of code that are prone to errors and will require additional maintenance work. Developers customize code fragments: they copy for development purposes based on their specifications. As a result, code fragments have typically evolved in distinct ways from their original parts. It causes gaps to appear between the copied and original code segments. It is critical to find these gaps in order to properly identify code clones. Understanding clones and the software systems better requires the detection of gapped clones. The only way to address this problem is to offer a completely new rebuild architecture. [1]. Kodhai E et al. [2] proposed a framework using a light-weight approach to detect method-level clones. However, this framework has not been tested using mutants. Ó Cinnéide M [3] tested various software metrics that serve as a basis for refactoring. Our framework also proposed function-level detection to serve as a path for method-level refactoring. Ishihara [16] detected inter-project clones, and our proposed method is also applicable to detect the same. Roy CK. [8] implemented an approach to detect near-miss clones but had limitations not checking this with injected mutants. Our proposed template followed the technique of encoding all data types available in Java. Instead, Ami R., and Haga H. [9] replaced all variables/identifiers with '$' sign. The researcher has not encoded data types 'int', reserve words: return, if, else, public etc. Therefore, the template proposed will definitely consume more space. Rather, the researcher could use some short encoding for the same, to reduce execution time and space. Marcus & Maletic [10] used the removal of comments and token regularization for the Intermediate representation/ transformation technique for code fragments and files. Modifications to short source-code sentences have been performed to detect any changes to any source-code line [13]. Detection of lexical similarity between sentences has been performed in literature [14], because researchers follow minor modifications to short sentences as well. Refactoring techniques have been applied to refactor function clones from open-source code [1], this is helpful in the removal of cloned

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 11
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 December 2023*

function bodies. Clones carry *important* domain knowledge, and that knowledge can only be accessed by identifying clones at a particular level of granularity. Granularity varies from lines, methods, and functions to file level. The larger the granularity, the fewer will be the clones, and will lead faster search process, of course, it will provide some meaningful information because long text possesses some meaning. Keeping the concept in mind, the research carried out in this paper focuses on function/method level clone detection from source code, because function/method possesses the maximum functionality of the software. A generalized template has already been proposed by our research [11] and a basic methodology has also been proposed by us [12], which serves as a basis for clone detection. Nomenclature for Keywords and Reserve Words has been adopted. Afterward, in further research, mutants have been embedded in the input dataset to verify the accuracy of the proposed template. Clone detection has been performed by computing accuracy. Open-source software JhotDraw has been chosen as the input dataset in Java. Around 50 mutants have been injected into the JhotDraw dataset, but only 25 random mutants are listed in Table 1. Additionally, our method reports mutation within clones, which might help refactor and identify error origins. Similar to other methods, we find similar code fragments. Nevertheless, modified parts components are found as part of our clone detection process. This improves the chances of refactoring and latent error identification. Source code translation can also be verified using mutants, source-to-source code translation automates the process of converting a program from one programming language to another. Existing research on code translation primarily evaluates effectiveness using either syntactic similarity metrics (e.g., BLEU score) or test execution results. However, syntactic metrics fail to capture semantic correctness, while test execution results, though more reliable, often suffer from insufficient test coverage and data. MBTA (Mutation-Based Code Translation Analysis), a novel approach in literature applies mutation analysis to assess code translation quality. MTS (Mutation-Based Translation Score), a metric that quantifies the reliability of a translator. This approach identifies translation bugs by comparing the execution results of mutated versions of a program and its translated counterpart. [17]

II. RESEARCH METHODOLOGY

In this section, a conceptual diagram of the framework is shown as in Figure 1 below. In the first phase, some *randomly mutated functions* have been injected (M1, M2, M3….) to already extracted function bodies from the system.

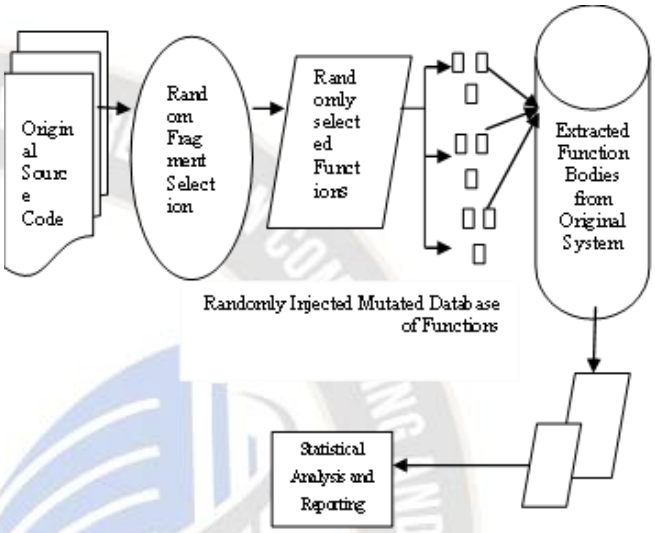

Figure1. A Mutation Based Framework for detecting Clones

During second step, randomly injected mutated database of functions is produced and in the last step validation testing is performed to analyze the performance. Table 1 shows selected mutants with respective assigned ids, Number of parameters of each mutant and the intermediate template of each mutant. Input source Code base has been processed using Rabin-Karp Rolling hash to compute hash code for every mutant function. For type-1 and type-2 clones, same hash signatures will be produced, because same intermediate template has been generated for these two types of clones. The template for type-3 clones is slightly different, because the number of parameters, insertion/deletion in any line, insertion/deletion of any line may get vary in these clones, therefore it will generate different hash codes.

Table 1**:** Injected Mutants in source code

Random Mutants	**Mutant Id**	**Intermediate Template**	**No of parameters**	**Hash Code**
public Figure owner() { return fOwner;}	M1	pu##(){re#;}	**0**	**27**
public Figure ownerExtended() { return fOwner;}	M2	pu##(){re#;}	**0**	**27**

public Figure ownerExtended(int x, int y) { return fOwner;}	M3	pu##(#,#){re#;}	**2**	**33**
public Rectangle displayBox() { return owner().displayBox();}	M4	pu##(){re#().#();}	**0**	**36**
public Rectangle display() { return owner().displayBox();}	M5	pu##(){re#().#();}	**0**	**36**
public Rectangle display(int x, int y) { return owner().displayBox();}	M6	pu##(#,#){re#().#();}	**2**	**40**
protected Connector findConnector(int x, int y, Figure f) { return f.connectorAt(x, y);}	M7	pr##(#,#){re#.#(#,#); }	**2**	**44**
protected Connector findConnectorExtended(int x, int y, Figure f) { return f.connectorAt(x, y);}	M8	pr##(#,#){re#.#(#,#); }	**2**	**44**
protected Connector findConnector(int x) { return f.connectorAt(x, y);}// (int y, Figure f) deleted from parameters	M9	pr##(#){re#.#(#,#);}	**2**	**40**
public Tool tool() { return fTool;}	M10	pu##(){re#;}	**2**	**28**
public Tool toolExtended() { return fTool;}	M11	pu##(){re#;}	**2**	**28**
public Tool toolExtended(int x, int y) { return fTool;}	M12	pu##(#,#){re#;}	**2**	**33**
static public Locator east() { return new RelativeLocator(1.0, 0.5);}	M13	stpu##(){rene#(#.#,#.#);}	**2**	**29**
static public Locator eastExtended() { return new RelativeLocator(1.0, 0.5);}	M14	stpu##(){rene#(#.#,#.#);}	**2**	**29**
static public Locator eastExtended(char x, char y) { return new RelativeLocator(1.0, 0.5);}	M15	stpu##(#,#){rene#(#.#,#.#);}	**2**	**38**
public Enumeration points() { return fPoints.elements();}	M16	pu##(){re#.#();}	**2**	**31**
public Enumeration pointsExtended() { return fPoints.elements();}	M17	pu##(){re#.#();}	**2**	**31**
public Enumeration points(float x, float y) {	M18	pu##(#,#){re#.#();}	**2**	**39**

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 11
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 December 2023*

return fPoints.elements();}				
protected ToolButton createToolButton(String iconName, String toolName, Tool tool) { return new ToolButton(this, iconName, toolName, tool);}	M19	pr##(#){rene#(#,#,#,#);}	**3**	**-38**
protected ToolButton createToolButtonExtended(String iconName, String toolName, Tool tool) { return new ToolButton(this, iconName, toolName, tool);}	M20	pr##(#){rene#(#,#,#,#);}	**3**	**-38**
public Figure owner() { return fOwner;}	M21	pu##(){re#;}	**0**	**27**
public Tool tool() { return fTool;}	M22	pu##(){re#;}	**0**	**28**
public Rectangle displayBox() { return owner().displayBox();}	M23	pu##(#,#){re#.#();}	**0**	**36**
static public Locator east() { return new RelativeLocator(1.0, 0.5);}	M24	stpu##(){rene#(#.#,#.#);}	**0**	**29**

Testing has been implemented using Java and evaluated it on three different versions of JhotDraw (5.2, 6.0b1 and 7.0.6), which is an open source software and has been analyzed vastly in the literature [3, 4, 5, 6, and 7]. Details of these variants are shown Table 1a. Experimentation has been performed on varying size of functions. As it can be observed from table 2 that minimum function size is LOC=2 and maximum function size is LOC=235. Details of clone sets, clone pairs and total number of clones reported in each version have been shown in Table 3. Maximum number of clone sets, clone pairs and maximum number of clones has been reported in JhotDraw 6.0b1; therefore, it has been concluded that JhotDraw 6.0b1 is the maximum cloned software among these three versions.

Table1a: Software System under Investigation

System	**Size**	**Files**	**LOC**	**Classes**	**Return Func**	**Super Functions**	**Blank Functions**	**Total Functions**
JHotdraw 5.2	5.72 MB	160	14,577	208	266	56	78	1037
JHotdraw 6.0b1	14 MB	484	21,091	405	759	110	106	4256
JHotdraw 7.0.6	7.0 MB	310	32447	350	566	35	155	2811

Table 2**:** Functions with varying LOC (Lines of Code)

System	**Function (Lines of Code)**		
	Minimum	**Average**	**Maximum**
JhotDraw 5.2	2	6	35

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 11
Article Received: *25 July 2023* ***Revised:*** *12 September 2023* ***Accepted:*** *30 December 2023*

JhotDraw 6.0b1	2	6	195
JhotDraw 7.0.6	2	8	235

Table 3**:** Details of clone sets, clone pairs and number of clones

System	Clone Sets	Total number of clones pairs	Total number of clones detected
JhotDraw 5.2	488	1167	2340
JhotDraw 6.0b1	499	1497	2898
JhotDraw 7.0.6	389	1464	2524

III. RESULTS AND DISCUSSION

A. Mutation Based Validation

Injected mutants (M1,M2,M3…)have been grouped into different clusters C0, C1….C8. Figure 2 represents the number of clusters of mutants and Figure 3 explains the type of clone reported in each cluster. Clusters (C0, C2 and C4) reported Type-2 clones with C2 containing one false positive (FP), (C1, C3 and C7) reported Type-3 clones and (C5,C6, C8) reported Type-1/Type-2 clones. It has been validated manually that all clusters reported 100% precision except cluster-C2. Because there is one false positive i.e. M6 in cluster C2, so the precision value of C2 is 66.66 %. Recall of all clusters is 100%, because all mutants in code base have been detected successfully.

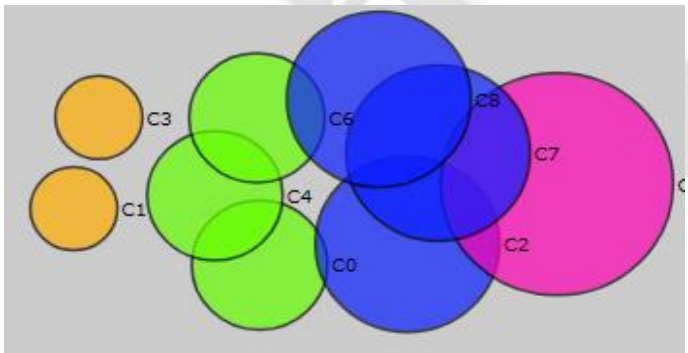

Figure 2: Clusters of Mutants

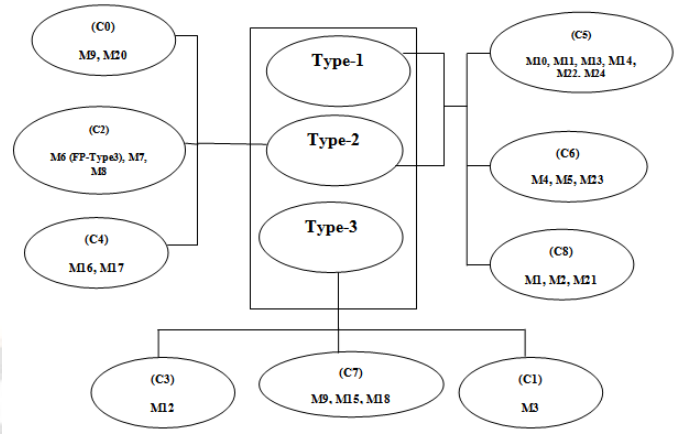

Figure 3**:** Cluster wise Mutation Based Function Clones (C0-C8 clusters) (FP-False Positive)

Figure 4 depicts that number of files and number of classes in version 6.0b1 is more than other two versions. But Lines of Code (LOC) has been increased gradually from 5.2 to 7.0.6 as depicted by Figure 5. Figure 6 represents the elaborated details of various functions in all three versions.

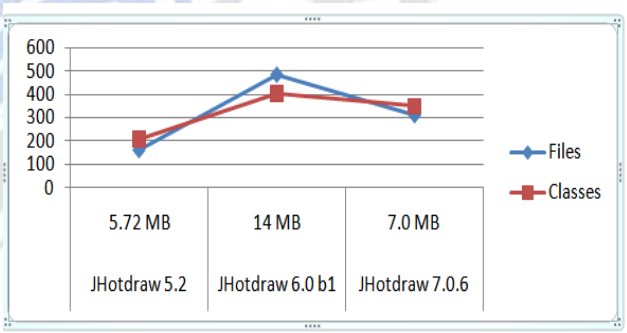

Figure 4**:** Files and Number of Classes in JhotDraw variants

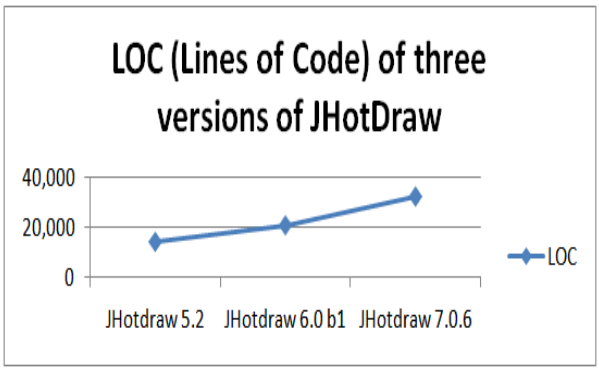

Figure 5: Lines of Code

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 11
Article Received: 25 July 2023 *Revised:* 12 September 2023 *Accepted:* 30 December 2023

Figure 6**:** Different functions detected

Figure 7 demonstrates clustering results on the JHotDraw code dataset (including original code and mutants) using k-Means clustering, reduced to two dimensions with PCA (Principal Component Analysis).

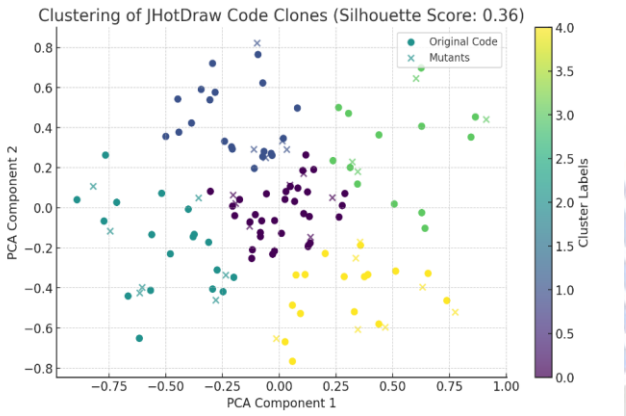

Figure 7**:** Clustering of JhotDraw Clones

1) Data Representation

Original Code: Represented by circular markers (o), these points indicate samples from the original JHotDraw source code. Mutants: Represented by cross markers (x), these are generated by making slight alterations to the original code (such as variable renaming or minor structure changes) to simulate clones.

2)Clustering

k-Means Clustering was used with five clusters. Each cluster is color-coded, indicating the groupings of code samples that k-Means found to be most similar. Each color represents a distinct cluster, with closely located points in the same color indicating code samples that are more similar to each other based on their features. The clusters show groups of code samples that are likely clones or variants, as similar codes should ideally fall within the same cluster.

3)Interpretation of Clusters

Cluster Quality: Some clusters are tightly packed, suggesting strong similarity within the cluster (e.g., the purple and yellow clusters). These may represent groups of highly similar code, possibly exact or near-exact clones. Distribution of Mutants: Mutant samples (crosses) generally appear close to the original code samples (circles) in the same cluster. This closeness indicates that the clustering model successfully identifies mutants as similar to their originals, grouping them in the same cluster. Cluster Overlap: There are some cases where clusters overlap, which could indicate code samples that have mixed characteristics or ambiguity in their classification as clones.

B. Clustering Metrics and Observations

1)Cluster Count and Density:

There are 5 clusters (as indicated by the color gradient scale on the right). Some clusters are dense and concentrated, indicating strong intra-cluster similarity. For example, the purple and yellow clusters are tightly packed, meaning that the code samples within these clusters are more similar to each other. The presence of spread-out clusters, like the ones with green and blue colors, suggests that those clusters contain code samples with more variation within the group, potentially capturing broader functional similarities rather than exact or near-exact clones.

2)Distribution of Mutants and Originals:

Mutant samples (represented by crosses) are generally located near their corresponding original samples (represented by circles) within the same cluster. This distribution confirms that the clustering algorithm effectively groups mutants with their original versions, identifying them as similar based on code features. Some mutants appear slightly separated from their originals, which may indicate minor structural or semantic differences introduced during mutant generation, leading to slight variations in their feature representation.

3)Intra-cluster vs. Inter-cluster Distance:

Clusters that are visually close to each other, such as the blue and teal clusters, may contain code samples with less distinct boundaries between them. This is consistent with a moderate silhouette score, as the proximity of these clusters can cause some overlap. The separation between clusters, especially between the yellow and teal clusters, shows that the algorithm could distinguish some sets of code clones with higher confidence.

4)Color Gradient (Cluster Labels):

The color gradient represents different clusters, with each unique color (purple, yellow, green, teal, blue) representing a distinct group of code samples. This color distinction allows easy visual identification of clusters, making it easier to observe which code samples are grouped as clones or near-clones.

IV. THREAT TO VALIDITY

Despite the encouraging results, this work has one potential threat to validity. The similar functions reported in the results might not be considered cloned (duplicate) for any other developer, because two different people would have a different understanding of the same line. It is hard to avoid subjectiveness while evaluating open-source software because it lacks a template benchmark.

V. CONCLUSIONS

Mutation-based framework has been analyzed by injecting mutants into the input dataset (JHotdraw). Files and Number of Classes in JhotDraw variants, Details of clone sets, clone pairs and number of clones have been computed in JHotDraw. Generalized templates have been generated for all mutants. Afterwards, the Clustering approach is followed to group the same type of clones. Tyep-1, Type-2, and Type-3 clones are detected with injected mutants and a high recall value has been reported. Mutant samples (crosses) generally appear close to the original code samples (circles) in the same cluster. This closeness indicates that the clustering model successfully identifies mutants as similar to their originals, grouping them in the same cluster.

Competing Interests- Not Applicable
Funding Information- Not Applicable

Author contribution

1. The research introduces a new method that adds intentional changes (mutants) to open-source code (JHotDraw) to test how well clone detection works, offering a more reliable and realistic way to measure its accuracy compared to older approaches
2. This study concentrates on functions and methods, considering them the most reusable parts of software. It applies generalized templates to detect Type-1, Type-2, and Type-3 code clones.3.

3. Experiments were conducted on three versions of JHotDraw (5.2, 6.0b1, 7.0.6), and statistical analysis like number of clone sets, clone pairs, and function sizes were provided, showcasing the framework's effectiveness in real-world open-source systems.

Data Availability Statement- The Dataset is available on https://sourceforge.net/projects/jhotdraw/files/JHotDraw/JHotDraw%207.0.x/

SourceForge: The primary repository for JHotDraw, offering various versions including 5.2, 6.0b1, and 7.0.6

Research Involving Human and /or Animals: No human or animals were harmed during this research.
Informed Consent. NA

REFERENCES

1. Davey N, Barson P, Field S, Frank R, Tansley D. , "The development of a software clone detector", International Journal of Applied Software Technology, pp. 219-236, 1995.
2. Kodhai E, Kanmani S., "Method-level code clone detection through LWH (Light Weight Hybrid) approach", Journal of Software Engineering Research and Development, Vol. 2, no.1, 2014.
3. Ó Cinnéide M, Tratt L, Harman M, Counsell S, Hemati Moghadam I., "Experimental assessment of software metrics using automated refactoring", Proceedings of the ACM-IEEE international symposium on Empirical Software Engineering and Measurement, pp. 49-58, 2012.
4. Chen X, Wang AY, Tempero E., "A replication and reproduction of code clone detection studies", Proceedings of the Thirty-Seventh Australasian Computer Science Conference, pp. 105-114, 2014
5. Mubarak-Ali AF, Syed-Mohamad SM, Sulaiman S., "Enhancing Generic Pipeline Model for Code Clone Detection using Divide and Conquer Approach", Int. Arab J. Inf. Technol., Vol. 12, no. 5, pp 510-517, 2015.
6. White, Martin, et al. "Deep learning code fragments for code clone detection." Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. ACM, pp. 87-98, Aug 25, 2018.
7. Li, Z., Sun, J., "An iterative, metric space-based software clone detection approach", 2nd International Conference on Software Engineering and Data Mining (SEDM), pp. 111–116, 2010.
8. Roy CK., "Detection and Analysis of near-miss software clones", IEEE International Conference on Software Maintenance, ICSM, pp. 447-450, 2009.
9. Ami R, Haga H., "Code Clone Detection Method Based on the Combination of Tree-Based and Token-

Based Methods", Journal of Software Engineering and Applications, 10:13, p.891, 2017.

10. Marcus and J. I. Maletic, "Identification of high-level concept clones in source code", Proceedings of 16th IEEE International Conference on Automated Software Engineering (ASE'01), pp. 107-114, 2001.
11. Harpreet Kaur, "Detection Of Code Clones In Open Source Software Using Proposed Generalized Template", *Webology,* Volume 19, Number 2, 2022, ISSN: 1735-188X), p. 4284.
12. Harpreet Kaur, " Basic Methodology to Detect Code Clones ", NeuroQuantolog, 20(6), June 2022, pp 101405-101414.
13. Kaur H, Maini R. Assessing lexical similarity between short sentences of source code based on granularity. International Journal of Information Technology. 2019 Sep 1;11:599-614.
14. Kaur H, Maini R. Granularity-based assessment of similarity between short text strings. In Proceedings of the Third International Conference on Microelectronics, Computing and Communication Systems: MCCS 2018 2019 (pp. 91-107). Springer Singapore.
15. Kaurr H, Maini R. Function clone removal using refactoring techniques. Advances in Mathematics: Scientific Journal. 2020;9(6):4001-13.
16. Ishihara T, Hotta K, Higo Y, Igaki H, Kusumoto S. Inter-project functional clone detection toward building libraries-an empirical study on 13,000 projects. In 2012 19th Working Conference on Reverse Engineering 2012 Oct 15 (pp. 387-391). IEEE.
17. Guizzo G, Zhang JM, Sarro F, Treude C, Harman M. Mutation analysis for evaluating code translation. Empirical Software Engineering. 2024 Jan;29(1):19