# A Novel Approach to Detect Duplicate Code Blocks to Reduce Maintenance Effort

Sonam Gupta
Research Scholar, Suresh Gyan Vihar University,
JAIPUR (RAJASTHAN), INDIA

Dr. P. C Gupta
Associate Professor, Department of Computer Science & Informatics, University of Kota
KOTA (RAJASTHAN), INDIA

***Abstract*—It was found in many cases that a code might be a clone for one programmer but not the same for another one. This problem occurs because of inaccurate documentation. According to research, the maintainers are not aware of the original design and thus, face the difficulty of agreeing on the system's components and their relations or understanding the work of the application. The problem also occurs because of the different team of development and maintenance resulting in more effort and time during maintenance. This paper proposes a novel approach to detect the clones at the programmer side such that if a particular code is a clone then it can be well documented. This approach will provide both the individual duplicate statements as well as the block in which they appear. The approach has been examined on seven open source systems.**

***Keywords*—Clones; Program Dependence Graph (PDG); Control Flow Graph (CFG); Abstract Syntax Tree (AST)***

## I. Introduction

Detecting duplicate code occurrence requires a powerful understanding of the clones. Studies[1,2] connected with clone investigation conveys with its assessments of partner degree question inside one of the supply code, as in an exceedingly net application [3], inside the identification of clone-related bugs [4], partner degreed in a product bundle piece [5]. Distinctive studies measure different previews upheld the modification history of the code to see the genealogic nature of clones [6], to watch the consistency of clones being looked after [7], and to spot real refactoring of clones [8].

Alongside this, there are difficulties connected with the understanding of clones at the clone group level and potential monstrous measures of data that may be recovered from clone identification instruments. Also, difficulties were found inside the upkeep of clones in light of the fact that it identifies with the evacuation of their related duplication through the strategy for refactoring.Redundant code is additionally typically deceptively referred to as cloned code within the literature—although that means that one piece of code springs from the other one within the original sense of this word. In step with the Merriam-Webster lexicon, a clone is one that seems to be a replica of a resourceful kind. It's an equivalent word to duplicate. Despite the fact that exploration winds up in repetitive code, not every excess code could be a clone. There are additional cases inside which two code fragments that aren't any duplicate of each option just happen to be comparative or perhaps indistinguishable all of a sudden. Likewise, there is additionally repetitive code that is semantically proportional. However, it consolidates an absolutely different usage. There is no understanding of the investigation group on the exact thought of repetition and cloned code. The meaning of clones communicates this dubiousness as clones square measure portions of code that square measure comparative in venture with some meaning of likeness. There are various exact studies on the advancement of clones that depict some consideration getting perceptions. Antoniol, *et al*. propose measurement got from clones over numerous arrivals of a framework to watch and foresee the development of clones [9]. Their study for the data base framework mSQL demonstrated that the forecast of the normal assortment of clones every work is genuinely solid. In an alternate detailed analysis for the UNIX working framework piece, they found that the extent of cloned codes is confined. Exclusively few clones are regularly found crosswise over frameworks; most clones are completely contained inside a subsystem. Inside the framework building design, constituting the equipment plan reflection layer, more up-to-date equipment architectures have a tendency to display marginally higher clone rates. The explanation behind this improvement is that more current modules are normally gotten from existing comparative ones.

Reusing code pieces by reiteration and sticking with or while not minor adjustment may be a typical movement in programming bundle advancement. Therefore, programming bundle frameworks ordinarily contain areas of code that are awfully comparable, alluded to as programming bundle clones. Prior exploration demonstrates that a real part of the code in an extremely commonplace programming framework has been cloned , and in one great case it had been even five hundredth [10]. While such duplicate is generally purposeful and may be useful from multiple points of view [11], it might be unsafe in programming bundle support and development. Case in point, if a bug is located in a code piece, all similar parts should be checked for a comparative bug [12]. Copied pieces might impressively expand the work to be carried out once upgrading or adjusting code [13]. A late study that worked inside the setting of business frameworks demonstrates that conflicting changes to codes are successive and bring about extreme astounding conduct [14]. Numerous other options moreover demonstrate that product bundle frameworks with code clones will be harder to keep up [15,16] and may present refined blunders [12,17]. In this way code clones are thought about one amongst the undesirable "odours" of a PC code [18] and it is wide accepted that cloned code will make programming bundle upkeep and advancement impressively a great deal of troublesome. Accordingly, the location, viewing and evacuation of code clones is a crucial subject in programming bundle support and development [19]. Thus, the algorithm proposed in this paper will help the programmer to locate the exact position of the cloned code. Along with the position it will also inform about the percentage of clone within the system. If the percentage is more than that of threshold value then the programmer either documents it correctly or else removes the clone. This reduces the maintenance effort.

## II. Algorithm to Detect the Duplicate Codes

The algorithm proposed is based on the hybrid technique which combines program dependence graphs (PDG), control flow graphs (CFG), and abstract syntax tree (AST) based approach. This approach is better than other approaches [21] since by combining three techniques it will work well at the complier level and AST will help to construct the nodes of duplicate codes. The algorithm will work as follows.

First, the statements will be inserted in a function named InsertStatements()

Step 1: Generate a key for entering statement into navigable collection (which is the value of the statement)

Step 2: increment the no of statements present in navigable collection

Step 3: if identity is already present, add this value into that key value pair

Step 4: else add a new item into the navigable collection and terminate

After making the pool of all statements, the matrix will be used for setting the values as true in a square matrix of size equal to no. of statements in an item of the navigable collection and the matrix is updated only in the upper half portion, that is, above the upper right principal diagonal only with no operation on lower half of matrix. The working of matrix will be done in putAMarkOnMatrix() function :

Step 1: initialize index1 with starting index of statement1

Step 2: initialize index2 with starting index of statement2

Step 3: if index1 < index2

Set all the cells true or 1 from index1 to index2

Step 4: else

Set all the cells true or 1 from index2 to index1

Now the duplicity of the statements will be checked in StatementGroup.java function. For checking the duplicity, it uses a matrix to store all the different statements in the class

Step1: if (index1 < index2) then collector.setTrueToCell(index1, index2);

Step2: else collector.setTrueToCell(index2, index1);

Whenever a duplicate statement is found, the cell in the matrix for showing the result is highlighted, and a mark is put on the code which is duplicated again using the above given code as given in following code:

```
Step 1: for (inti = 0; i< N - 1; i++) {  //N Array size
           Statement s1 = statArray.get(i);
 Step2: for (int j = i + 1; j < N; j++) {
          Statement s2 = statArray.get(j);
           //Match or not
             if (s1.equals(s2)) {      //insert the result to matrix
                   putAMarkOnMatrix(collector, s1, s2);
Step 3: Repeat above steps till N
```

Above code is checking the code for the duplicity. If it is duplicate code, then it puts the mark on the duplicate code using above pseudocode. Now the AST structure will be constructed so that the clones can be identified.

The working of construction of AST will be as follows:

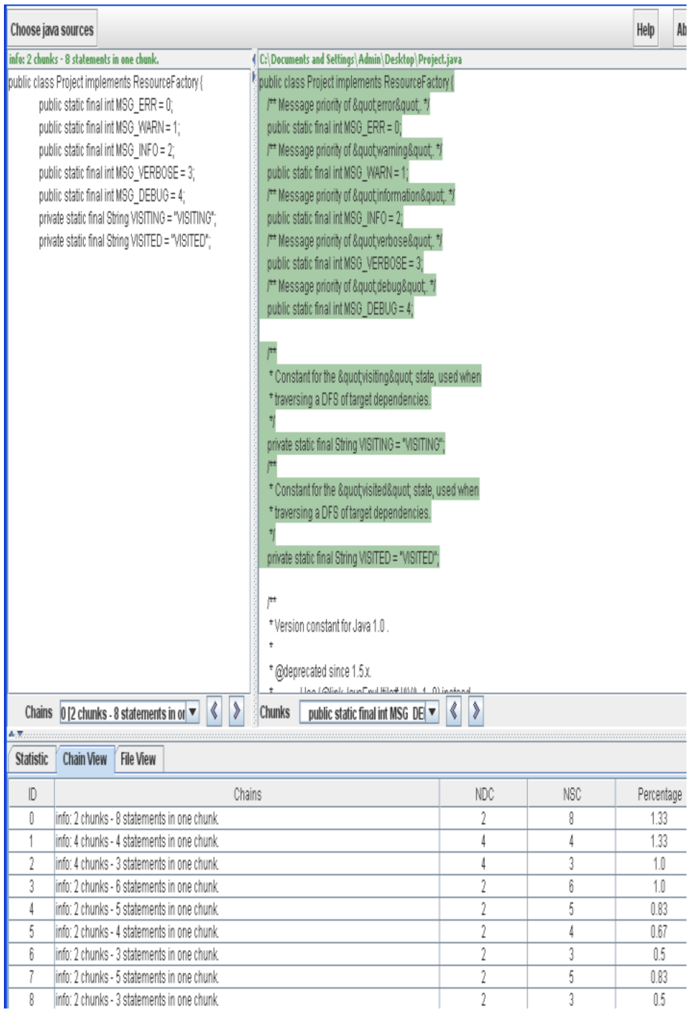
```
Step 1: Initialize token = new CodeReviewToken(i, s);
Step 2: Make type= astType; //token.getType();
Step 3: if (Configuration.anonymizeType(type)) then
 return  JavaRecognizer._tokenNames[type];
Step 4: return token.getText();
Step 5; if (token != null) then
        index = (token).getStartIndex();
Step 6:Initialize tid = this.token.getID();
 Step 7: if (tid > -1) then tl.addToken(tid)
Step 8: if (getFirstChild() != null) then
        if (!Configuration.anonymizeType(astType)) then
           ts += ((CodeReviewAST)
getFirstChild()).toStringList(tl);
Step 9: Display ts as the duplicated code block.
```

The AST constructed will also be able to identify non-contiguous clones as now if any extra line is inserted or deleted, it will form a block of code.

By using the above-mentioned pseudocode every token will be assigned a unique id so that when converting each text into tokens all of them can be grouped on the basis of their numbers and then each duplicate code can be categorized under single head.

## III. Results and Experimental Study

The above proposed algorithm has been developed in Java. The tool has been analysed on seven open source systems namely Apache Ant 1.7.0, Columbia 1.4, EMF 2.4.1, JMeter 2.3.2, JEdit 4.2, JFreeChart 1.0.10, and JRuby 1.4.0. The result will be viewed as given in Figure 1.



info: 2 chunks - 8 statements in one chunk.

```
public class Project implements ResourceFactory {
        public static final int MSG_ERR = 0;
        public static final int MSG_WARN = 1;
        public static final int MSG_INFO = 2;
        public static final int MSG_VERBOSE = 3;
        public static final int MSG_DEBUG = 4;
        private static final String VISITING = "VISITING";
        private static final String VISITED = "VISITED";
```

C:\Documents and Settings\Admin\Desktop\Project.java

```
public class Project implements ResourceFactory {
  /** Message priority of &quot;error&quot;. */
  public static final int MSG_ERR = 0;
  /** Message priority of &quot;warning&quot;. */
  public static final int MSG_WARN = 1;
  /** Message priority of &quot;information&quot;. */
  public static final int MSG_INFO = 2;
  /** Message priority of &quot;verbose&quot;. */
  public static final int MSG_VERBOSE = 3;
  /** Message priority of &quot;debug&quot;. */
  public static final int MSG_DEBUG = 4;

  /**
   * Constant for the &quot;visiting&quot; state, used when
   * traversing a DFS of target dependencies.
   */
  private static final String VISITING = "VISITING";
  /**
   * Constant for the &quot;visited&quot; state, used when
   * traversing a DFS of target dependencies.
   */
  private static final String VISITED = "VISITED";

  /**
   * Version constant for Java 1.0 .
   *
   * @deprecated since 1.5.x.
```

| ID | Chains | NDC | NSC | Percentage |
|---|---|---|---|---|
| 0 | info: 2 chunks - 8 statements in one chunk. | 2 | 8 | 1.33 |
| 1 | info: 4 chunks - 4 statements in one chunk. | 4 | 4 | 1.33 |
| 2 | info: 4 chunks - 3 statements in one chunk. | 4 | 3 | 1.0 |
| 3 | info: 2 chunks - 6 statements in one chunk. | 2 | 6 | 1.0 |
| 4 | info: 2 chunks - 5 statements in one chunk. | 2 | 5 | 0.83 |
| 5 | info: 2 chunks - 4 statements in one chunk. | 2 | 4 | 0.67 |
| 6 | info: 2 chunks - 3 statements in one chunk. | 2 | 3 | 0.5 |
| 7 | info: 2 chunks - 5 statements in one chunk. | 2 | 5 | 0.83 |
| 8 | info: 2 chunks - 3 statements in one chunk. | 2 | 3 | 0.5 |

Fig. 1. Screenshot of proposed tool

As shown in Figure 1, the tool will tell the programmer about the number of duplicate blocks and number of statements in each block along with its location. Now the programmer can make the changes in the documentation or in the code as required. This approach will reduce the maintenance effort to a much lower level. The number of clones found in each system by the help of this tool were compared with the JDeodorant tool[20] as well as with manual detection[20]. The comparison between the proposed approach and JDeodorant is shown in Figure 2. The results clearly show that the proposed tool extracted more number of bad smells. Similarly. the cloned blocks detected by the proposed tool are compared with the manual approach as shown in Figure 3. It clearly shows that the proposed tool is able to find almost same number of cloned blocks as that were actually present in the systems.

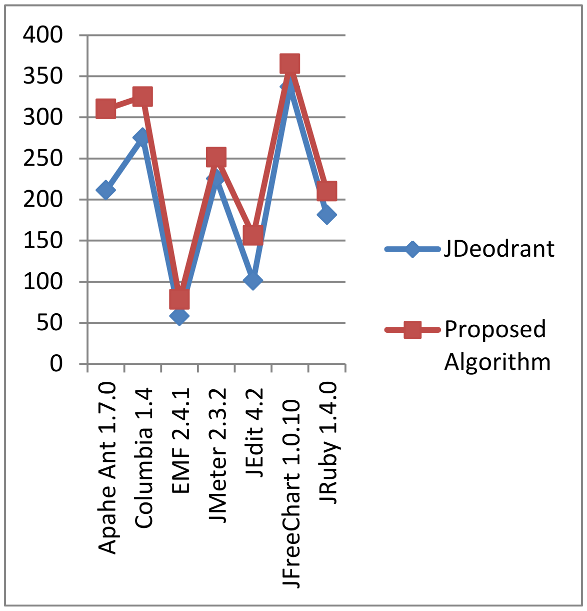


Fig. 2. Comparison of detected clones

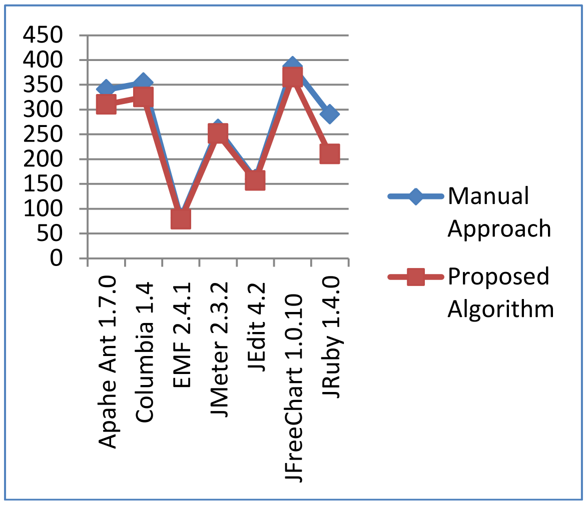


Fig. 3. Comparison of detected clones with manual approach

## IV. Conclusion

The proposed tool has been developed to reduce the maintenance effort, as it has been proved [5] that a lot of cost and effort is wasted in maintenance due to clones. This tool consists of hybrid techniques of PDG, CGF and AST thereby overcoming the disadvantages of each. The tool has been experimented of seven open source systems. Along with it the results have also been compared with JDeodorant tool as well as with manual extraction. The results clearly show that the tool is able to find more number of clones which actually have bad smell. Moreover. the tool also provides the location of each duplicate block along with its percentage. Now the programmer can set the threshold value based on the percentage. If he finds that the percentage is higher, then he will either remove the duplicity or will document it correctly so as to reduce maintenance time and cost.

## V. Future Work

The tool has been experimented on just open source systems. In future. the work will be extended to study on licensed systems. Along with this, the work will be extended to convert the duplicated block into functions so that a single change can be reflected in all versions.

## References

[1] Basili, V. R. and B. T. Perricone (1984). "Software errors and complexity: an empirical investigation." Commun. ACM 27(1): 42-52.

[2] Parnas, D. L. (1994). Software aging. Proc. Int'l Conf. on Software Engineering (ICSE). Sorrento, Italy, IEEE Computer Society Press**:** 279-287.

[3] Damith Rajapakse and Stan Jarzabek, “Using Server Pages to Unify Clones in Web Applications: A Trade-Off Analysis,” *International Conference on Software Engineering*, Minneapolis, Minnesota, May 2007, pages 116 - 126.

[4] Lingxiao Jiang, Zhendong Su, and Edwin Chiu, “Context-Based Detection of Clone-Related Bugs,” *Joint Meeting of the European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Dubrovnik, Croatia, September 2007, pages 55 - 64.

[5] Robert Tairas and Jeff Gray, “An Information Retrieval Process to Aid in the Analysis of Code Clones,” *Empirical Software Engineering*, Volume 14, Number 1, February 2009, pages 33 - 56.

[6] Miryung Kim, Vibha Sazawal, David Not kin, and Gail Murphy, “An Empirical Study of Code Clone Genealogies,” *Joint Meeting of the European Software Engineering Conference and Foundations of Software Engineering*, Lisbon, Portugal, September 2005, pages 187 - 196.

[7] Lerina Aversano, Luigi Cerulo, and Massimiliano Di Penta, “How Clones are Maintained: An Empirical Study,” *European Conference on Software Maintenance and Reengineering*, Amsterdam, The Netherlands, March 2007, pages 81 - 90.

[8] Robert Tairas and Jeff Gray, “Sub-clones: Considering the Part Rather than the Whole,” *International Conference on Software Engineering, Research, and Practice*, Las Vegas, Nevada, July 2010.

[9] Antoniol, G., Casazza, G., Penta, M.D., Merlo, E.: Modeling clones evolution through time series. In: International Conference on Software Maintenance, IEEE Computer Society Press (2001) 273–280.

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

[11] Cory Kaiser and Michael W. Godfrey. “Cloning Considered Harmful” Considered Harmful: Patterns of Cloning in Software. Empirical Software Engineering, Vol. 13(6):645–692 (2008).

[12] 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.

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

[14] E. Juergens, F. Deissenboeck, B. Hummel and S. Wagner. Do Code Clones Matter? In Proceedings of the 31st International Conference on Software Engineering (ICSE’09), pp. 485–495, Vancouver, Canada, May 2009.

[15] Xian, Y., Angler, D.: Using redundancies to find errors. In: Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering, ACM Press (2002) 51–60.

[16] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: local algorithms for document fingerprinting. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD’03), pp. 7685, San Diego, California, June 2003.

[17] A. Chou, J. Yang, B. Chelf, S. Hallem and D. R. Angler. An Empirical Study of Operating System Errors. In *Proceedings of the 18th ACM symposium on Operating systems principles (SOSP’01)*, pp. 73–88, Banff, Alberta, Canada, October 2001.

[18] Martin Fowler. *Refactoring: Improving the Design of Existing Code*, Addison-Wesley, 1999.

[19] B. Lagu¨e, D. Proulx, J. Mayrand, E. Merlo and J. Hudepohl. Assessing the Benefits of Incorporating Function Clone Detection in a Development Process. In Proceedings of the 13th International Conference on Software Maintenance (ICSM’97), pp. 314– 321, Bari, Italy, October 1997.

[20] Krishnan, Giri Panamoottil, and Nikolaos Tsantalis. "Unification and refactoring of clones." Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. IEEE, 2014.

[21] C.K. Roy and J.R Cordy, A Survey on Software Clone Detection Research, Queen’s School of Computing Technical Report: 541 pp., September 2007.