A Hybrid Framework for Software Clone Detection

Neha Saini[1], Sukhdip Singh[2]

[1]Deenbandhu Chhotu Ram University of Science and Technology, Murthal, 131001, India

[2]Deenbandhu Chhotu Ram University of Science and Technology, Murthal, 131001, India

Abstract

Software systems are becoming increasingly complex, and managing them is a critical topic in the software business today. Cloning is one of the fundamental factors that makes software maintenance more difficult. Code cloning is a copy-paste approach for reproducing code portions. Developers find it simple to copy and paste code throughout the early stages of a project's development. This paper presents a hybrid framework for detecting code clones. To detect code clones, it combines a token-based technique with metrics-based technique. Initially, metric based technique is applied to locate prospective clones. After identifying prospective clones, token-based comparison is done to confirm that they are indeed clones.

Keywords: Code Clone, Code Clone Detection, Metric, Token

1 Introduction

Code clone detection locates clones, or identical or similar portions of code, within or between software systems. Clones are made for a variety of reasons, including copy-paste-modify programming, unintentional code functionality resemblance, plagiarism, and code generation[1]. Software practitioners rely on code clone detection techniques and tools to detect and manage code clones;therefore, they've long been a research topic[2].

Clone management is critical for maintaining software quality, detecting and preventing new issues, as well as lowering development risks and expenses [3]. The availability of high-quality tools is also critical for clone research. At least 70 different tools have been reported in the literature, according to Rattan et al[4] Despite the fact that various strategies for clone identification have been developed over the year the accuracy and scalability of clone detection tools and techniques is still a hot topic of research.

1.1 Code Clone Terms

- **Code Fragment:** It's defined as a set of code lines with varying degrees of resemblance between different code fragments in the source code. There may or may not be comments in these comparable code fragments. For instance, a series of sentences, a begin-end block, and so on[5].
- **Clone Set**: It is defined as a collection of all code fragments that are identical or comparable[6].
- **Clone Pair:** A Clone Pair is formed when two code fragments are inspected and there is a clone relationship between the two code pieces[7].

- **Clone Class**: The clone class is defined as a set of all clone pairs in which the existing clone pairs have some clone relationship between them.

1.2 Clone Types

- **Type 1:** Identical copy with the exception of white space and comments.
- **Type 2:** As type 1, but with the addition of variable renaming.
- **Type 3:** Similar to type 2, but with a few changes or additions.
- **Type 4:** Syntax isn't always the same, but it's semantically same.

2 Related works

A software, according to Tajima et al.[8], is made up of a group of programmers. Each team member is responsible for writing code for a certain part. Because eachprogrammer works individually, it's possible that a handful of them will develop the identical code. The generated code will have a clone when the project leader integrates the entire code. Extra processing and storage will be required as a result of this. Rainer [9] has published a report on software redundancy, duplication, and cloning, as well as their various kinds. He has examined the origins of clones as well as their harmful consequences. The study's contribution is knowledge on how to avoid clones, as well as an evaluation of existing methodologies and clone detection benchmarks.

Rattan et al. [4]conducted a thorough examination of the various code clone detection tools and approaches available. They have identified some open research questions in this subject. The study can assist users in determining the utility of a tool based on their needs.

Mondal et al. [10] used a common framework to perform an experiential study. Their plan was to use four tools to deploy nine code clone methods on 15 systems in order to investigate the impact of code cloning on software preservation. The study's flaw is that it hasn't been properly applied yet. Nonetheless, the findings reveal that copied code is more likely to be updated and is less stable.

3 Proposed Framework

The proposed hybrid framework combines metrics-based clone detection with token-based clone detection. There are three stages to the process. To find probable clones in the first stage, a metric-based technique is applied. Potential clones are chosen based on how closely the two source files' metrics matches. The metrics are calculated at the class, function, and threshold levels, with the threshold level being defined for metric matching. In the second phase, only actual clones are recognized using a token-based technique if the metrics match count above the threshold value; otherwise, there is no need to calculate actual clones because there is no potential clone between the source files. The current work describes a method for choosing a collection of relevant metrics for code clone detection approaches that use metrics. A set of metrics is evaluated from a large number of metrics presented in the literature of code clone detection such that the metrics in the set are independent of one another, i.e there is no correlation between the metrics, and these metrics also produced good

results in code clone detection. To reduce the comparison cost and make the approach computationally efficient, independent metrics are used. Two tools [11], [12]for identifying software metrics are utilized to determine the values of required metrics for implementing the strategy. The metrics values can be exported as a Comma Separated Values (CSV) file using these tools. Starting with all combinations of one metric, metrics are evaluated on precision and recall before gradually increasing the number of metrics in the metrics combinations until the entire set of metrics involved in the approach is utilized. The suggested method overcomes a fundamental drawback of metrics-based code clone detection strategies, namely, low precision.

3.1 Architecture of Proposed Metric and Token Based Software Clone Detection and Management

The proposed framework is divided in to three steps (1) Selection of prospective clones on the basis of metrics match, (2)Processing of prospective clone candidates by token-based technique to determine whether two prospective clones really are clone of each other and (3) Clone Management by Ranking of Clones on the basis of Management Overhead. All three steps are discussed in detail in the subsequent sections. The architecture of the proposed framework is depicted in Figure 1.

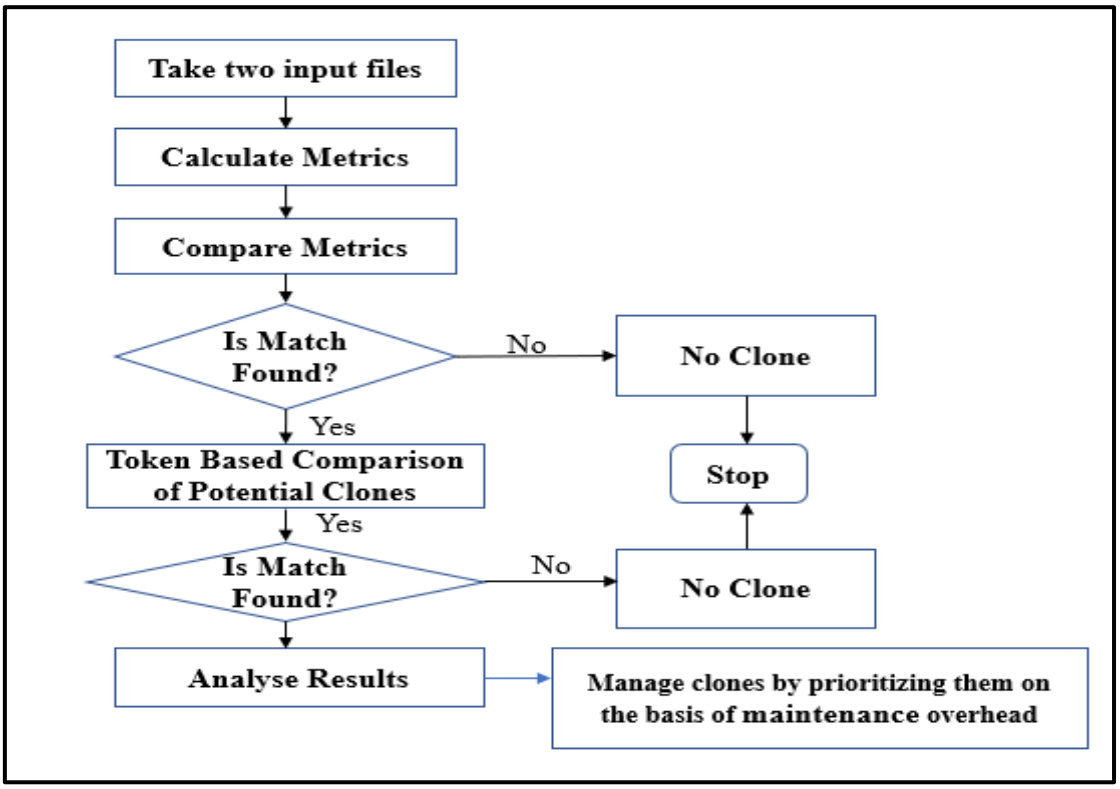

Figure1: Architecture of Proposed MTB-SCDM Framework

3.2 Selection of prospective clones on the basis of the metric match.

In the first step, selection of prospective clone candidates is done on the basis of selected software metrics. The output of this step is passed on to the next step for further refinement of detected clones.

1. Software Metrics: Software metrics are numerical numbers for some software properties or software units. Metrics that quantify functions as software units are necessary in the suggested approach.

2. Metrics Classification

Metrics can be broadly classified in to three:

- Product
- Process
- Project

The literature on clone detection has a vast number of measures. Among such metrics, however, only a set of nine is picked. These metrics were chosen because they are independent of one another, i.e. there is no correlation between them, and they also produced good results in terms of code clone identification. NOR is a novel statistic that results in a significant increase in precision for the experiment done. As a result, the suggested method evaluates a total of nine metrics which are given in Table 1.

Table 1 Metrics Name and Meaning

Metric Number	Metric Name	Metric Meaning
1	Cyclomatic Complexity	Number of decision points+1 or Edges – Nodes +Connected components
2	Depth	The maximum level of nesting of control constructs
3	LOC	The number of lines in a function

4	CountInput	The number of functions that a function uses plus the number of unique subprograms calling the function i.e. Functions called by+ Parameters read + Variables read
5	CountOutput	The number of outputs that are set i.e. Functions Call+ Parameter set/exchange+ variables set/exchange
6	CountPath	Number of Unique paths through a function
7	CountStmtDecl	Number of Declarative Statements
8	CountStmtExe	Number of Executable Statements
9	NOR	Number of return statements in a function

We began by applying a single statistic from the metrics listed above, which resulted in nine distinct combinations: (1), (2), (3), (4), (5), (6), (7), (8) and (9). The stages for each of these combinations are as follows:

1. **Calculation of metrics and creation of CSV files**

Two tools are used to calculate the metrics indicated inside each combination. LOC, Complexity, CountInput, CountOutput, CountPath, CountStmtDecl, and CountStmtExe are seven metrics calculated using the tool, Understand [11], which is a tool for calculating metrics. The metrics, Depth and NOR are calculated with the help of a tool called Source Monitor [12]. The metrics values can be exported as a CSV file using these tools. The data from the CSV files is then entered into the database.

2. **Formation of Clone Pairs and Clone Classes**

By using a comparison algorithm, clone classes and clone pairs are formed for each combination. The pseudocode for proposed comparison algorithm is given in Table 2. It is possible to distinguish between type-1 and type-2 clones using this method.

Table 2 Pseudocode for Comparison Algorithm

Input: $\forall fun_i \in$ FS input values of m1k, m2k,……. mrk.
Output: Clone classes cc1, cc2,……ccn for MCk.

The different notations used in the algorithm are:

- ***N: total number of functions in the system where we need to find out clones***
- ***FS: set of all functions of software system s.t. FS={$fun_1, fun_2, \ldots\ldots\ldots\ldots fun_N$}***
- ***fun_i:i^{th} function of software system where i=1 to N***
- ***MC_kMetric combination for which clone classes need to be detected.***
- ***$m_{rk}$$r^{th}$ metric in metric combination MC_ks.t.MC_k= (m_{1k}, $m_{2k,}$, m_{rk}).***
- ***f_i. m_{rk}Value of metric m_{rk} for function f_i.***
- ***CRep: Clone Repository where all detected clones are stored***

```
CRep<-NULL
for i=1 until N
{
if ( funi∉cci) ∀cci ∈CRep then
  cci ←{funi}
end if
for j = i+1 until N
     {
     if (funim1k == funjm1k && funim2k == funjm2k &&……..funimrk ==
funjmrk )
     then
      cci ← {funj}
     end if
    }
CRep ← cci
}
```

3.3 Processing of prospective clone candidates by token-based technique to determine whether two prospective clones really are clone of each other.

The clone classes generated in the previous step are input to the token-based algorithm.After that precision and recall values are calculated for the generated clone classes and selection of subset of metrics is done on the basis of precision and recall values.

4 Results and Discussion

The proposed approach is implemented on wget[13] which is a large sized software system and is described in below Table 3:

Table 3 Description of case used

Case	LOC	Functions
wget	17K	247

Table 4 shows the maximum precision and recall scores for each metric combination as we progressed from single-metric combinations to combinations comprising all nine metrics.

Table 4 Precision and Recall Value for different Metrics Combination

Number of Metrics used	Metrics Combinations	Precision	Recall	No. of Detected Clone Classes
One	(1)	0.15	1	26
Two	(1,3)	0.25	1	24
Three	(1,3,6)	0.5	0.8	22
Four	(1,3,6,7)	0.61	0.76	21
Five	(1,3,4,6,7)	0.8	0.8	19
Six	(1,2,4,6,7,9)	0.95	0.87	17
Seven	(1,2,3,4,5,6,7)	0.88	0.38	14
Eight	(1,2,3,4,5,6,7,8)	0.89	0.55	15
Nine	(1,2,3,6,7,4,5,8,9)	0.90	0.86	18

When only one metric is used to detect clones, the maximum precision is 15 percent and the highest recall is 100 percent. When metric number one, complexity, is utilised, these values are obtained. In the event of a combination of two measurements, the highest precision and recall are 25 percent and 100 percent respectively. Table 4 shows that the value of greatest precision grows as the number of metrics increases, although is only applicable up to six metrics. The precision value declines when seven metrics were utilized. The best combination

is (1,2,4,6,7,9), which yielded a precision of 95% and a recall of 87%. As a result, the combination (1, 2, 4, 6, 7, 9) is regarded as a set of significant code clone detection metrics. The highest precision and recall values for combinations of multiple metrics are shown in Figure 2.

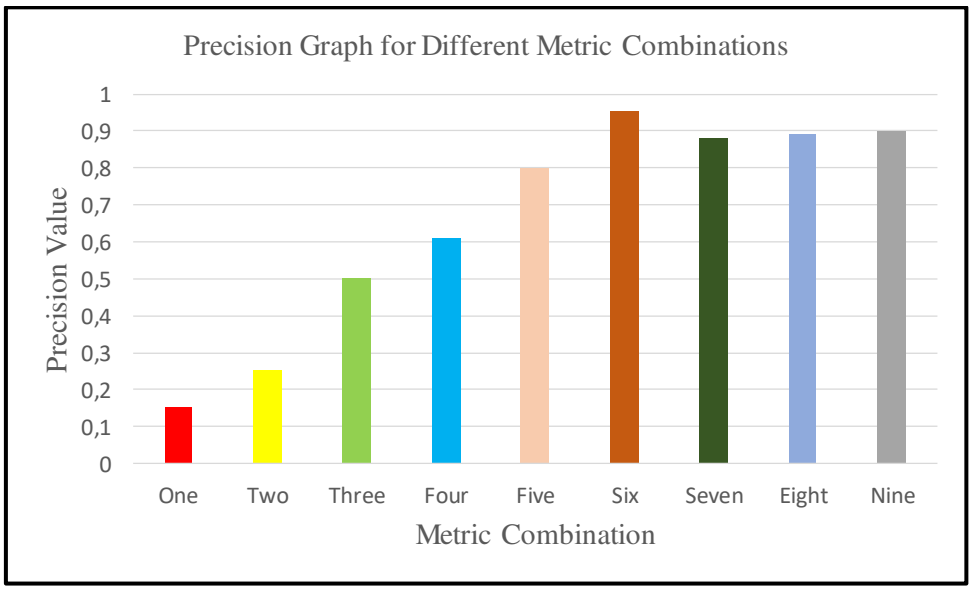

Figure 2: Precision Graph for different metrics combination

Table 5 Precision and Recall Value for different Metrics Combination

Number of Metrics used	Metrics Combinations	Precision	Recall	No. of Detected Clone Classes
One	(1)	0.15	1	26
Two	(1,3)	0.25	1	24
Three	(1,3,6)	0.5	0.8	22
Four	(1,3,6,7)	0.61	0.76	21
Five	(1,3,4,6,7)	0.8	0.8	19
Six	(1,2,4,6,7,9)	0.95	0.87	17
Seven	(1,2,3,4,5,6,7)	0.88	0.38	14
Eight	(1,2,3,4,5,6,7,8)	0.89	0.55	15

Nine	(1,2,3,6,7,4,5,8,9)	0.90	0.86	18

5 Conclusion

In this paper, a hybrid framework for detecting and managing clones is proposed which combines metric-based techniques with token-based techniques. Selection of prospective clones, comparison of prospective clones and their management are three main steps of the proposed framework. Type-1 and Type-2 clones are detected using the proposed framework, which starts with a metric based match and then moves on to a token based match. The proposed technique uses metric and token based matching to detect clones. The proposed method searches for clones at function levels. The proposed method uses metric matching to identify probable clones. To establish whether two prospective clones are truly clones of one other, potential clones are compared token by token.

The proposed technique significantly improves the process of managing code clones. Although code clone detection is the essence of clone management, the detected clones need to be treated very carefully. They can either be eliminated or refactored inside the code base, as deleting the code clones is not always viable. It is also not always required to refactor the complete code clone set; some of the less damaging clones can be omitted.

References

[1] H. A. Basit and S. Jarzabek, "A Data Mining Approach for Detecting Higher-Level Clones in Software," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 497–514, Jul. 2009, doi: 10.1109/TSE.2009.16.

[2] S. Ducasse, M. Rieger, and S. Demeyer, "A language independent approach for detecting duplicated code," in *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). "Software Maintenance for Business Change" (Cat. No.99CB36360)*, Aug. 1999, pp. 109–118. doi: 10.1109/ICSM.1999.792593.

[3] P. Kumar, Nitin, V. Sehgal, K. Shah, S. S. P. Shukla, and D. S. Chauhan, "A novel approach for security in Cloud Computing using Hidden Markov Model and clustering," in *2011 World Congress on Information and Communication Technologies*, Dec. 2011, pp. 810–815. doi: 10.1109/WICT.2011.6141351.

[4] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," *Information and Software Technology*, vol. 55, no. 7, pp. 1165–1199, Jul. 2013, doi: 10.1016/j.infsof.2013.01.008.

[5] T. Kamiya, "Agec: An execution-semantic clone detection tool," in *2013 21st International Conference on Program Comprehension (ICPC)*, May 2013, pp. 227–229. doi: 10.1109/ICPC.2013.6613854.

[6] S. Thummalapenta, L. Cerulo, L. Aversano, and M. Di Penta, "An Empirical Study on the Maintenance of Source Code Clones," *Empirical Software Engineering*, vol. 15, pp. 1–34, Feb. 2010, doi: 10.1007/s10664-009-9108-x.

[7] Y. Higo, T. Kamiya, S. Kusumoto, and K. Inoue, "ARIES: refactoring support tool for code clone," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 4, pp. 1–4, May 2005, doi: 10.1145/1082983.1083306.

[8] R. Tajima, M. Nagura, and S. Takada, "Detecting functionally similar code within the same project," in *2018 IEEE 12th International Workshop on Software Clones (IWSC)*, Mar. 2018, pp. 51–57. doi: 10.1109/IWSC.2018.8327319.

[9] R. Koschke, R. Falke, and P. Frenzel, "Clone Detection Using Abstract Syntax Suffix Trees," in *2006 13th Working Conference on Reverse Engineering*, Benevento, Italy, 2006, pp. 253–262. doi: 10.1109/WCRE.2006.18.

[10] M. Mondal, C. K. Roy, and K. A. Schneider, "A comparative study on the bug-proneness of different types of code clones," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2015, pp. 91–100. doi: 10.1109/ICSM.2015.7332455.

[11] "Scitools Licensing | Download Understand." https://licensing.scitools.com/download (accessed Apr. 10, 2021).

[12] "SourceMonitor." http://www.campwoodsw.com/sourcemonitor.html (accessed Apr. 10, 2021).

[13] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and Evaluation of Clone Detection Tools," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 577–591, Sep. 2007, doi: 10.1109/TSE.2007.70725.