An Efficient Software Clone Detection System based on the Textual Comparison of Dynamic Methods and Metrics Computation

Gurunadha Rao Goda
Tata consultancy Services, Chennai, India

Avula Damodaram, Ph.D
Professor of CSE & Director Academic Audit Cell, JNTUH, Hyderabad, India

ABSTRACT
Code clone is a kind of software recycling that has a greater influence on the maintenance of huge software systems. The business services are provided by web applications that employ a combination of page design and language code scripting. Usually, code redundancy in applications result from copy and paste practices called code clones. Searching for software clones is the key objective of this research. The clone identification procedure which is introduced in this paper is a hybrid approach that depends on template conversion and metrics comparison. There are four phases involved in the proposed scheme, namely, input and pre-processing, template conversion, metrics computation and clone type detection. File integration, elimination of white noise and statement normalization are the steps involved in the pre-processing stage. The metrics that are calculated includes numerous lines of code, arguments, looping statements, return statements and conditional statements. The pairs that have identical characteristics during textual evaluation are termed as the clones. This method of clone detection seems to be less complex with better accuracy and efficiency in contrast to other existing methods. The performance analysis is made against the prevailing systems to show efficiency improvement obtained through this method. The implementations are carried out with the help of JAVA.

Keywords
Clone detection, Template conversion, Metrics computation, File integration, Normalization.

1.INTRODUCTION
The software development process includes frequent actions like code fragment copying and reuse by means of pasting with or without slight variations or adjustments [5]. This kind of reuse technique for existing code is called as Code Cloning and the pasted code fragment is termed as Clone of the original. Improved results and lower complexity are offered by Software Clone Detection [8]. Clone Detection approach is to discover the reused code fragment in any application to retain. Clones are mostly the outcome of copy-paste events that are very simple and try to decrease the programming difficulties and time [2] by making use of the fragment of code that is previously available and not rewriting similar code from scratch. This functioning is usual, primarily in device drivers of operating systems with identical algorithms. Another cloning called the 'Accidental Cloning' [4] arises occasionally due to the utilization of the similar set of APIs to realize similar protocols rather than using direct copy and paste activities.

The quality, maintainability and comprehensibility of the software systems are greatly influenced by code clones [5-6]. The abnormality in probability update increases with cloning. Identification of all the cloned fragments that are related to a code fragment with a bug is necessary to fix the bug in question. Excessive cloning results in the system size increment and commonly specify design problems like missing inheritance and missing procedural abstraction [1-3] Moreover, the expense dealt with the maintenance of clones over a system's lifespan is high. Improved results with huge complexity have lead to the significant advancements in Clone Detection [10]. Greater part of the methods were restricted to discovering program fragments that are identical to their syntax or semantics, despite the fact that the smaller part of candidates which are really clones and fraction of actual clones known as candidates normally stay similar. The functional clones in C source code [8] can be detected by means of the estimation of Metric based [11] approach merged with the textual comparison of the source code. Detection procedure uses the values of a range of metrics that has been devised. The metric based method seems to be the least complex, largely accurate and well-organized way of detecting clones.

2. RELATED WORKS
Metrics-based approaches [6] involve the comparison of metric vectors of various metrics of code fragments that are collected, despite making direct comparison of codes. Numerous clone detection methods have evolved so far, that uses a wide range software metrics for identifying identical codes. A collection of software metrics known as fingerprinting functions are computed for one or more syntactic units like a class, a function, or a method or even statement and then the clone detection takes place by the comparison of the metric values over these syntactic units.

Mayrand et al. [15] have computed quite a few metrics for each function unit of a program. The code clones that are detected are nothing but the units having identical metric values. The units that partially similar units are not identified. Each and every function in the source code is embodied by the source code representation named Intermediate Representation Language (IRL). Names, design, expression and uncomplicated control flow of functions are used for metrics computation. A clone [2] can be described merely by pair of whole function bodies that possess similar metric values. This approach does not hold good for the identification of copy-paste at other granularity like segment-based copy-paste, which arise more repeatedly than function-based copy-paste.

Kontogiannis et al. [10] has made use of an abstract pattern matching tool that is based on Markov models to recognize possible matches. This method has failed to discover copy-pasted code but helps in determining the relationship among two programs. The approach makes a direct comparison of the metrics values to categorize a code fragment in the granularity [7] of begin − end blocks with the belief that two code fragments are related if their respective metric values are adjacent. In addition, the metric-based methods find application in detecting duplicated web pages or clones in web documents.

Di Ducca et al. [14] has put forward a method that uses the computation of distance between objects in web pages and for

finding the extent of similarity to detect similar static HTML pages. A string representation for each and every HTML/ASP pages of a Web Application (WA) is achieved through the replacement of each HTML/ASP control elements with a separate symbol from one of the two distinct set of alphabets, one for HTML tags and the other for ASP objects.

Lanubile, Calefato and colleagues [13-15] has introduced a semi automated method to detect cloned script functions. Initially, the method uses an automated approach to identify the potential function clones followed by a visual inspection in the chosen script functions. eMetrics is the tool used for recognizing the potential function clones [8] and the reports from the tool allow the visual inspection of the code of the selected script functions, sorting of suspect clones, and assembling of revealed function clones based on refactoring opportunities.

Davey et al. [12] uses the estimation of specific code block attributes to identify accurate, parameterized and near-miss clones followed by the utilization of neural networks to detect similar blocks based on their characteristics.

3.PROBLEM DEFINITION

The branch of Clone Detection has undergone a great progress in the past ten years. This advancement is due to the development of various methods, which makes use of complex algorithms and tool chains to offer clone detection with improved results and increased complexity. Various clone detection methods that are already available include textual comparison, token comparison, comparison of Abstract Syntax trees, Suffix trees and Program Dependency Graphs. The scalable and semantics-based approaches, which are prevailing today, are restricted to the discovery of program fragments that are identical in their syntax or semantically equivalent control structures alone. The above-mentioned techniques are in need of more complex parsing techniques that provide comparatively same precision and recall. In addition, these Clone Detection techniques are limited to a particular clone type only.

4.PROPOSED METHODOLOGY

In the preceding paper, the repeatedly used important methods and functions were recognized through dynamic coupling measurement. When the search for important methods has been completed, a new search for the clones has to be made in that function list. This research deals with the development of a novel technique for code clone detection that assist in detecting two or three clone types as specified in literature. It is a lightweight process for the identification of clones. Besides, it is capable of offering refactoring support for getting additional solutions with the detected clones. A new technique is introduced, which is the hybrid combination of metric-based approach and textual comparison of the source code for the detection of Clones. Several metrics have been developed to make use of their values during the detection process. The method introduced consists of 4 segments, namely, input and pre-processing, template conversion, metrics computation and detection of the clone types. The pairs that are identical in textual comparison are known to be the clones. In contrast to the other approaches, this method is of less complexity with higher accuracy, providing a better means for Clone Detection and it is implemented using the Java tool.

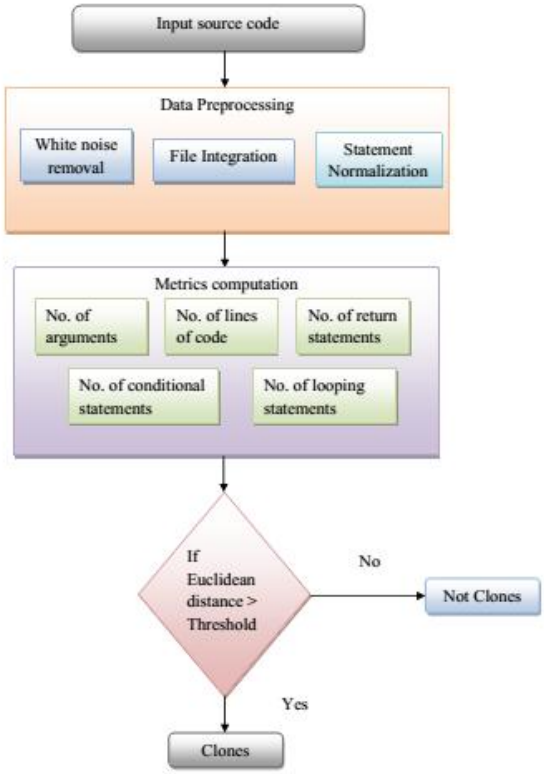

Fig 1: Architecture of the proposed methodology

4.1. Data Preprocessing

The undesired source codes for the comparison phase are filtered out in this stage of clone detection. This phase also offers file integration, white noise elimination and statement normalization.

(a). File Integration

File integration does the job of grouping or concatenation of all the files of the same project in to a single huge file for external parsing. The comments and the pre-processor statements can be eliminated during file integration.

(b). White noise removal

Following the removal of unwanted codes, the source code that continue to stay is partitioned into a set of disjoint fragments known as source units. These source units form the major source fragments that deal with direct clone relationship with one another and can be at any stage of granularity like files, classes, functions/methods, begin-end blocks, statements, or sequences of source lines. The comparison technique used by the tool decides whether, further partition of source units is necessary or not. For instance, source units may be further divided into lines or even tokens for comparison. Comparison units can be even obtained from the syntactic structure of the source units.

(c). Statement normalization

The source units may also be used as comparison units in situations where the subdivision of source units is not required.. For instance, the metric values can be evaluated from source units of any granularity in a metrics dependent tool. The similarity between the cloned fragments can be set up through the reorganization of source code into a standard format. Thus in the stage of normalization, all the identifiers in the source code are replaced by the same single identifier.

4.2. Template Conversion

Template conversion can be defined as the process of converting the source code input into a pre-defined collection of statements or into a standard form. It involves the renaming of data types, variables and function names. In textual evaluation, this kind of format is termed as the template. The textual comparison of the chosen candidates are modified while detecting the clones, function identifiers, variable names, types etc., and the textual comparison that is made, not during the cloning process, will not meet the requirements. The application of metrics necessitates the immediate storage of the source file and template file in the database after the completion of template conversion. This transformation can be as simple as elimination of white space and comment or very complex.

SOURCE CODE	TEMPLATES
i++	ASSIGNMENT
While	LOOP
Return TRUE	RETURN
int leng	DAT S

Fig 2: Example for template conversion

4.3. Metrics computation

This phase of clone detection is used to evaluate the metrics required for detecting clones and it involves a set of five metrics that are as follows:

1. Number of lines of code

The number of lines of code can be obtained by,

- Subtracting white space lines.
- Subtracting comment lines.
- Subtracting the lines that have block constructs alone.

2. Number of arguments

By denoting the function name followed by the function call operator and any data values that the function expect to receive, a function call can be made These values are the arguments for the parameters described for the function, and the process is termed as passing arguments to the function.

3. Number of conditional statements

In computer science, conditional statements, conditional expressions and conditional constructs are characteristics of a programming language that can achieve various computations or activities based on whether a programmer-specified Boolean condition proves to be true or false.

4. Number of looping statements

A looping statement allows a statement to be evaluated several times as required. It is of much use when some constraints are to be verified with a certain value.

5. Number of return statements

A return statement is used to stop the processing of the recent function and returns control to the caller of the function. A value-returning function must incorporate a return statement, holding an expression.

For each and every method that is detected, the metrics are evaluated to store its respective values in a database. Following the metric values evaluation, the records in the database are compared to detect the method pairs with equal or similar set of values. The resulting set of candidates is then processed to obtain the clone pairs.

The metrics acquired through the metrics computation are then compared and the Euclidean distances were calculated. Euclidean distance is calculated by means of the formula:

$$\text{Euclidean } distance(ED) = \sqrt{(x_1 - x_2)^2} + \sqrt{(y_1 - y_2)^2} + \sqrt{(z_1 - z_2)^2} \quad \textbf{(1)}$$

A threshold is set after the estimation of the Euclidean distance. The metrics with values larger than or equal to Euclidean distance is known as clone.

$$result = \begin{cases} Clone, ED \geq \tau, \\ Not\,clone, ED < \tau \end{cases} \quad \textbf{(2)}$$

Thus the detection of software clones is made.

5. RESULTS AND DISCUSSION

The proposed software clone detection system has been implemented in the working platform of JAVA (version JDK 1.5). The key objective of the proposed method is to detect the clones in the source code. This can be realized through the combination of both textual conversion and metrics computation. The step by step results obtained from the proposed method is described as follows.

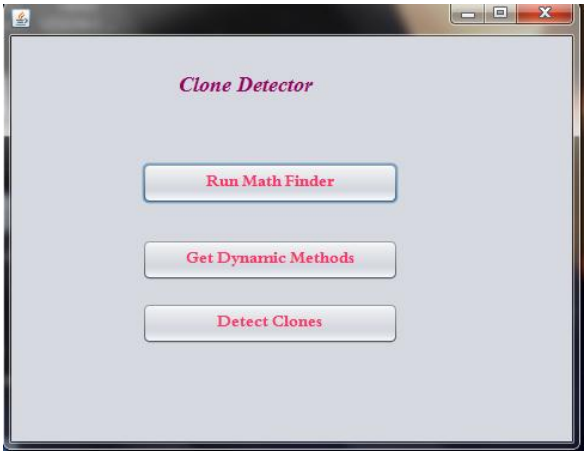

Fig 3: Initial process in the proposed methodology

The initial process of the proposed clone detection system is given in fig 3. The application that was used for the implementation of the methods is represented in fig 4.

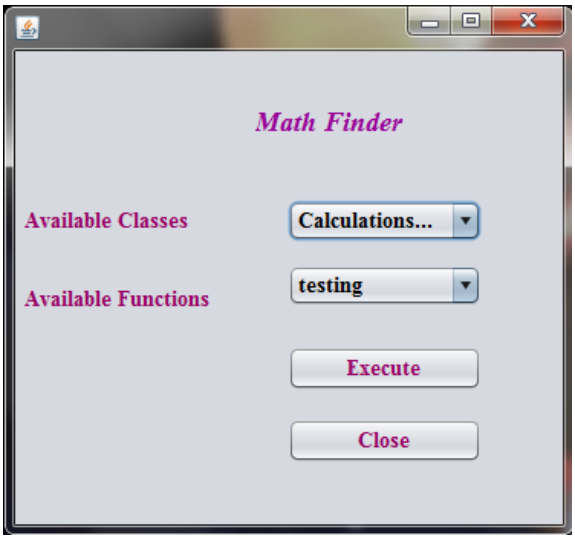

Fig 4: Application for the execution of methods

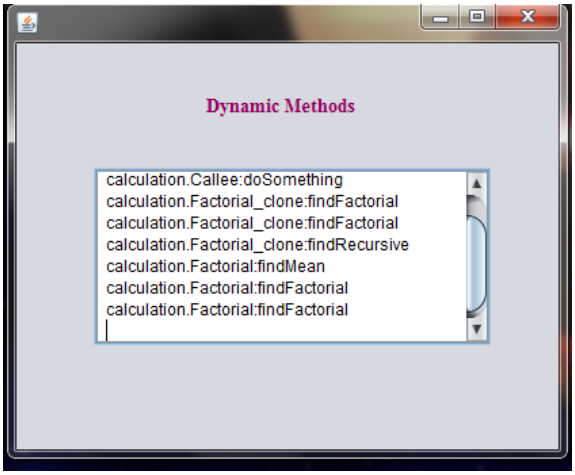

Fig 5: Executed output dynamic methods

Table 1: Sample metric values

SI. No.	Metrics	Value
1.	Number of lines of code	5
2.	Number of conditional statements	1
3.	Number of arguments	0
4.	Number of return statements	0
5.	Number of looping statements	4

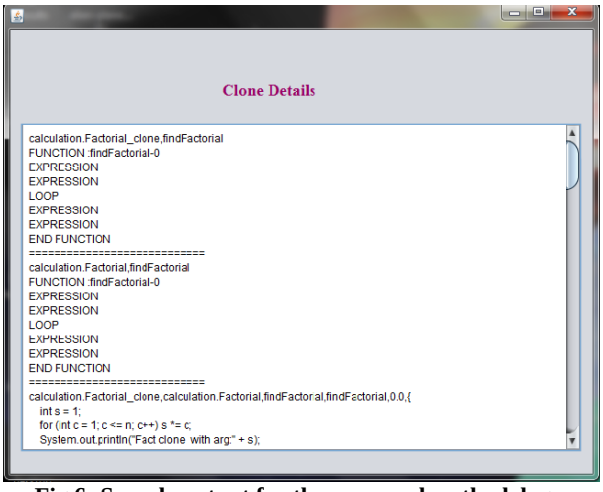

Fig 6: Sample output for the proposed methodology

The executed dynamic output methods are shown in fig 5. The sample of clones detected is given in fig 6.

5.1. Performance Measures

The quality of the system can be estimated through the quality metrics. The quality metrics considered in the proposed methodology are:

- Precision
- Recall
- Accuracy

1. Precision

Precision measures the proportion of actual clones which are correctly identified.

$$\text{Precision} = \frac{Number of clones correctly found}{Total number of clones} \quad (3)$$

2. Recall

Recall measures the proportion of non-clones which are correctly identified.

$$\text{Recall} = \frac{Number of clones found correct}{Total number of clones in the source code} \quad (4)$$

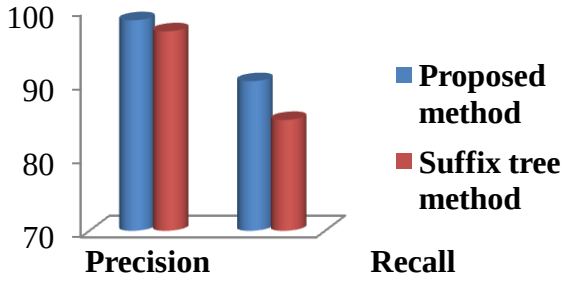

Fig 7: Graph for comparison of precision and recall

Table 2: Performance measure

Methods	Performance Measure	
	Precision	Recall
Proposed method	98.5	90.25
Suffix tree method	97	85

Fig 7: shows the comparison of the proposed methodology with the suffix tree method that currently exists. It shows that the proposed methodology has precision and recall rates higher than that of the existing techniques. So that it is evident that the method of clone detection proves to be better than the other prevailing methods.

6. CONCLUSION

In this paper, a novel clone detection technique that uses template conversion and metrics computation in a combined manner is being proposed. The results of the proposed system are analyzed based on a source code to offer a significant tempo of accuracy, precision and recall and reveal that more accurate clone detection is possible from the source code provided. The comparison results also show that our proposed clone detection system based on template conversion and metrics computation has given high exactness than the past methods. Hence, the proposed clone detection system, by utilizing the template conversion and metrics computation, is capable of proficiently identifying the clones in the input source code.

REFERRENCES

[1] L. Almagor, K. D. Cooper, A. Grosul, T. J. Harvey, S. W. Reeves, D. Subramanian, L. Torczon, and T. Waterman. Finding effective compilation sequences. In Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems, 2004.

[2] Mayrand, J., Leblanc, C., Merlo, E.: Experiment on the automatic detection of function clones in a software system using metrics. In: Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press, Monterey, 1996, pp. 244-253.

[3] Johnson, J.H.: Identifying redundancy in source code using fingerprints. In: CASCON, 1993, pp. 171-183.

[4] Kontogiannis, K., De Mori, R., Bernstein, R., Galler, M., Merlo, E.: Pattern matching for clone and concept detection. Journal of Automated Software Engineering 3, 1996, pp. 77-108.

[5] Buss, E., De Mori, R., Gentleman, W., Henshaw, J., Johnson, H., Kontogiannis, K., Merlo, E., Muller, H., Mylopoulos, J., Paul, S., Prakash, A., Stanley, M., Tilley, S., Troster, J., Wong, K.: Investigating reverse engineering technologies for the case program understanding project. IBM Systems Journal 33, 1994, pp. 477-500.

[6] Merlo, E., Antoniol, G., DiPenta, M., Rollo, F.: Linear complexity object-oriented similarity for clone detection and software evolution analysis. In: Proceedings of the International Conference on Software Maintenance - IEEE Computer Society Press, IEEE Computer Society Press, 2004, pp. 412-416.

[7] Kamiya, T., Kusumoto, S., Inoue, K.: Ccfinder: A multi-linguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering, 2002, pp. 654-670.

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

[9] Chanchal K. Roy, James R. Cordy, NICAD: Accurate Detection of Near-Miss Intentional Clones using Flexible Pretty-Printing and Code Normalization in ICPC, 2008, pp. 172-181.

[10] M. Kim and G. Murphy. An Empirical Study of Code Clone Genealogies. In FSE, 2005, pp. 187-196.

[11] Merlo, Detection of Plagiarism in University Projects using Metrics based Spectral Similarity. In the Dagstuhl Seminar: Duplication, Redundancy, and Similarity in Software, 2007.

[12] Giuseppe Antonio Di Lucca, Damiano Distant, and Mario Luca Bernardi. Recovering Conceptual Models from Web Applications. In Proceedings of the 24th Annual Conference on Design of communication (SIGDOC'06), October 2006, pp. 113-120.

[13] Saumya K. Debray, William Evans, Robert Muth, and Bjorn De Sutter. Compiler techniques for code compaction. ACM Transactions on Programming Languages and Systems (TOPLAS'00), March 2000, pp. 378-415.

[14] Andrea De Lucia, Rita Francese, Giuseppe Scanniello and Genoveffa Tortora. Reengineering Web Applications Based on Cloned Pattern Analysis. In Proceedings of 12th International Workshop on Program Comprehension (IWPC'04), June 2004, pp. 132-141.

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