A Measurement of Similarity to Identify Identical Code Clones

Mythili ShanmughaSundaram and Sarala Subramani
Department of Information Technology, Bharathiar University, India

Abstract: *Code clones are described as a part of the program which is completely or partially similar to the other portions. In the earlier research the code clones have been detected using fingerprinting technique. The major challenge in our work was to group the code clones based on similarity measure. The proposed system measures the similarity based on similarity distance. The defined expression considers two parameters for calculating the similarity measure namely the similarity distance and the population of the clone. Thereby the code clones are clustered and ranked on the basis of their similarity measures. Indexing is used to interactively identify the clones which are caused due to inconsistent changes. As a result of this work all the identical clusters for most similar and more similar categories are identified.*

Keywords: *Clone detection, software clones, fingerprinting, clustering, reuse.*

Received May 11, 2013; accepted March 19, 2014; published online August 25, 2015

1. Introduction

Code clones are the duplicated segments of the software which are produced by simple copy and paste mechanism [11]. This unwarranted duplicated code gives rise to many issues. For instance, if a user wants to correct an error in a system with duplicated code, all possible duplications of that specific error must be corrected. Code duplication normally increases the size of the code, thereby extending compile time and expanding the size of the program. Code duplication often indicates design problems in the software. During duplication, errors in systematic renaming can lead to unintended aliasing, resulting in latent bugs that crop up much later and the effect of all of these will lead to software aging. As a result even smaller design changes [2] become cumbersome and increase the complexity of the software.

Some clones are easy to detect, like clones with similar variable and clones with similar comments. Apart from that, there are many more delicate clones and to find such types of clones, a perfect clone detection technique is essential, which will prove more useful in finding the obvious clones. In general, clones may be described using the topology as described in Table 1 [13, 14].

Table 1. Clone types.

Type 1	An exactly identical source code, with no changes at all.
Type 2	An exactly identical source code clone, but with indentation, comments or identifier changes.
Type 3	A functionally identical clone, but with small changes made to the code to tailor it to some new function.
Type 4	A functionally identical clone, developed possibly by the originator who is unaware that already there exists a function that accomplishes essentially the same function.

According to the survey various clone detection techniques have been proposed to detect these types of clones [4, 5, 6]. In this research the proposed technique extracts the identical clones based on the similarity factors. The higher level clones which were detected from our previous work are considered as input. Based on varying degrees of similarity, the clones are clustered using the hierarchical clustering algorithm. Then, they are rank-ordered by using the method of indexing.

2. Motivation of Clone Detection

Many clone detection techniques have been previously designed in order to identify similar clones. The earlier work [9] has also identified the similar clones using the fingerprinting technique. The idea of the technique was to find the similarity using fingerprints. It maps a large dataset of arbitrary length into a same bit sequence. The message digest algorithm computes unique fingerprints for every token at the method level, file level and the directory level. The similarity between the fingerprints is calculated using Locality Sensitive Hashing. When the similarity value lies between the threshold ranges $0.8 \leq \delta \leq 1.0$ the clones are most similar and when it lies between the ranges $0.6 \leq \delta < 0.8$ they are more similar and the others are least similar. Under each threshold value method level, file level and directory level clones are identified. Finally, the clone pairs and clone set are formed. As an extension to the previous work, the motivation of this research is to find the identical clones in each category based on two parameters, statement similarity and the occurrence similarity Further the identical clones are clustered and ranked.

Hierarchical clustering [1, 8] is a set of clusters organised as hierarchical trees. The trees are visualized as dendrogram. Hierarchical clustering algorithm assigns each code clone to a cluster, so that N code

clones will have *N* clusters. The similarities between the clusters are the same as the similarities between the code clones. This algorithm finds the closest pair of clusters and merges them into a single cluster. It again computes the distances between the new cluster and the old clusters and repeats the steps until a single cluster remains. It finally produces a set of nested clusters organized as a hierarchical tree which can be visualized as a dendrogram. The major advantage of this clustering technique is that they do not assume any particular number of clusters. The clusters which are finally formed show a group of the code clones that are similar and identical.

Ranking is a process by which the items are ranked based on similarity factor. In our work since the similarity measure is calculated, the clones are ranked based on the similarity measure. After ranking, indexing is done to facilitate the automatic detection of clones. Given a clone, the rest of the clones can be automatically identified.

The contribution of this research is the description of generation of the similarity matrix used for clustering. This research also describes the ranking of the clones. Then, the interactive identification of clones is also discussed. The research work is organized in sections. In section 3 the related works for similarity generation and clustering is presented. In section 4 clone clustering system, details of the clone matrix representation, and ranking of clones are described. The quantitative evaluation of the approach in reported in section 5 and conclusions is given in section 6.

3. Related Works

The technique given in [15] is a method for detecting similar code blocks and for quantifying their similarity. It also detects the clone clusters for a set of code blocks within a user-supplied similarity threshold. The clones are further ranked and ordered based on the similarity and this technique is implemented for clone-detection in C programs. It is also suggested to incorporate this technique in many existing clone-detection tools to provide more flexibility in the definitions of similar clones.

The method of approximate clone detection as in [17] puts forward two techniques for detecting clusters of approximate clones. The experiments show that the proposed techniques accurately retrieve clusters of approximate clones that originate from copy-paste mechanism followed by independent modifications to the copied fragments.

Incremental clone detection tool [10] called ClemanX, represents code fragments as subtrees of Abstract Syntax Trees (ASTs) measures their similarity levels based on their characteristic vectors of structural features. ClemanX solves the task of incrementally detecting similar code. The empirical evaluation of the tool on large-scale software projects shows the usefulness and good performance of ClemanX.

Detecting near-miss clones [3, 12] employ a token-based system and use lightweight mechanisms for ensuring syntactic validity of potential clones. The method of similarity between the codes is the same as our method of calculating the similarity measure.

The tool CP-Miner as in [7] is proposed for identifying copy-paste bugs in large systems. The code sequences are transformed so that common subsequence can be identified using data mining techniques. CP-Miner fingerprints statements, although the fingerprints do not preserve similarity. Compared with other techniques, CP-Miner does not quantify the degree of similarity that exists between potential clones.

Real-time clone detection tool called SHINOBI [16] is implemented, in order to detect code clones from source code immediately by a real-time method. The clone detection and ranking module searches for clones with the search key, sent from the SHINOBI client using the Suffix Array Index. The order of returned clones is determined by ranking value. The ranking value is the sum of two values: The ratio of files committed at the same time and the ratio of files opened or edited at the same period of time.

4. Research Mechanism

The architecture takes source code as input and consists of four modules namely, extraction of higher level clones, similarity measure generation for clone pairs, clustering of identical clones and ranking as shown in Figure 1.

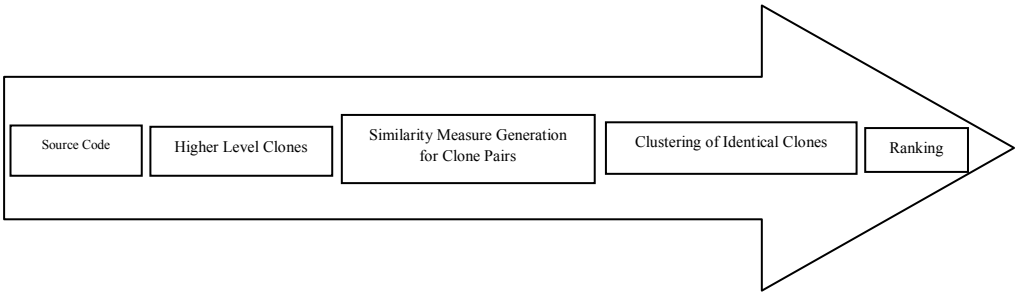

Figure 1. Architecture for clone detection.

4.1. Extraction of Higher Level Clones

In our previous work [9], any source code considered to have clones was taken as input. This was then pre-processed and tokenized to form tokens. Those tokens were converted in to fingerprints by the message digest algorithm. The above process is language independent. The fingerprints obtained by MD5 were further compared for similarity. Locality sensitive hashing was used for calculating similarity. It hashes for vectors such that the probability that two vectors having the same hash value is strictly decreasing function of their corresponding distance. In other words two vectors having the smaller distance will have the higher probability of having the same hash code. The final output of above process was a set of clone pairs and a clone set. The clone set thus formed consists of the list of the code clones which belong to one of the three categories. A clone in the clone set is any block of code such as a method, file or directory whose fingerprints are most similar, more similar or least similar to the other block of code belonging to the same category.

4.2. Similarity Measure Generation

Similar code clones are not always identical. In order to find the identical clones the module in the proposed architecture calculates a similarity measure using two factors. The first factor of similarity measure in Equation 3 is the similarity distance between two clones and the second factor is the distance between the numbers of occurrences of the clones.

$$Similarity\ \ Distance(S1,S2) = ((S1 \cap S2)/S1, (S1 \cap S2)/S2) \quad (1)$$

$$P(S) = No\ \ of\ \ Occurance\ of\ \ Clone\ \ in\ a\ Container \quad (2)$$

The similarity distance is calculated by using the Equation 1.The distance factor is an ordered pair. The first part of an ordered pair $(S1 \cap S2)/S1$ is a fraction of fingerprints in the first clone that is common to both the clones and the second part of the ordered pair $(S1 \cap S2)/S2$ is a fraction of fingerprints in the second clone common to both the clones. $P(S)$ in Equation 2 is the population of a clone which gives the maximum occurrence of a clone in a container. These two factors when considered for calculating the distance, the factors identify the matching clones that are not only having the common statements but also the clones which have occurred for the same number of times in a method, file or directory. When the statement similarity and the occurrence similarity are used to find the distance, it can be assured that the identical clones will be detected from a large set of similar clones. The similarity measure is calculated using the formula.

$$Similarity\ \ Measure = \sqrt{(S1 - S2)^2 + (P1 - P2)^2} \quad (3)$$

4.3. Clustering of Identical Clones

Normally the clones are clustered on the basis of the similarity measure. The advantage of this technique is to find the similarity measure only when the considered clone pair contains same fingerprints that are common to both. By doing this the number of candidate clones for clustering is reduced. Hierarchical clustering technique is used for clustering the clones, considers each of the candidate clones as a cluster and computes the similarity matrix. Consider a clone set with clones {*m*1, *m*2, *m*6, *m*7, *m*9, *m*10, *m*12}, then the similarity matrix has to be calculated for the above clones.

As a first step, Figure 2 shows each clone as a individual cluster. Next the similarity matrix is generated for every pair. Each value in the similarity matrix is a similarity measure between the pair of clones. The similarity measure between the same clones will be '0' and the similarity measure between the points sm_{ij} will be same as sm_{ji}.

Figure 2. Clones as individual clusters.

Table 2 indicates the similarity measure between the clone pairs. This is repeated for every clone pair in the clone set. The next step is to merge the clusters with closest similarity distance. If the similarity measure between the clones of the pair (*m*7, *m*10) is very close then merge the clusters of *m*7 and *m*10 and update the similarity matrix as in Figure 3 and Table 3. Finally a dendrogram is generated to display the clusters of identical clones.

Table 2. Similarity matrix before merging.

	M1	**M2**	**M6**	**M7**	**M9**	**M10**	**M12**
M1	0	sm_{12}	sm_{16}	sm_{17}	sm_{19}	sm_{110}	sm_{112}
M2		0	sm_{26}	sm_{27}	sm_{29}	sm_{210}	sm_{212}
M6			0	sm_{67}	sm_{69}	sm_{610}	sm_{612}
M7				0	sm_{79}	sm_{710}	sm_{712}
M9					0	sm_{910}	sm_{912}
M10						0	sm_{1012}
M12							0

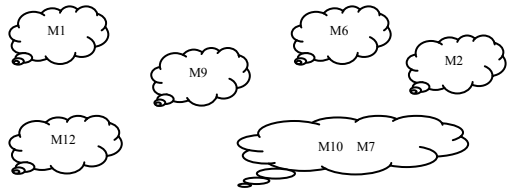

Figure 3. Clones as merged clusters.

Table 3. Similarity matrix after merging.

	M1	**M2**	**M6**	**M7**	**M9**	**M10**	**M12**
M1	0	sm_{12}	sm_{16}	sm_{17}	sm_{19}	sm_{110}	sm_{112}
M2		0	sm_{26}	sm_{27}	sm_{29}	sm_{210}	sm_{212}
M6			0	sm_{67}	sm_{69}	sm_{610}	sm_{612}
M7				0	sm_{79}	sm_{710}	sm_{712}
M9					0	sm_{910}	sm_{912}
M10						0	sm_{1012}
M12							0

4.4. Ranking

The main aim of the clone detection technique is to provide the clone list in an ordered format. In this work

the similar clones and the identical clones have been identified. Hierarchical clustering technique has also clustered the clones to show their identity. Since, the identity is based on the similarity measure the clones can also rank-ordered on the basis of this measure. The clones which are present in the same cluster are given the same rank. The method of indexing enables us to group all the clones based on the rank. The grouped clones along with their indexes are then stored in a database. Finally for a given clone, the method finds out and interactively lists all the other clones which are identical to it.

5. Experiments and Results

The source code given to our pervious system [9] has identified and extracted higher level clones at the method level, file level and the directory level. The output of the previous research [9] has identified 53 out of 181 methods, 90 out of 129 files, and 2 out of 3 directories as clones. The above clones have been identified as most similar clones for the threshold range 0.9-1.0. The next step is to detect the identical clones from same set of 53, 90 and 2 clones and then group them into clusters. Seven out of 53 clones for a threshold of 0.9-1 are considered for the study. The similarity distance and the similarity measures are calculated and Table 4 gives the similarity matrix for the considered clones. After finding the similarity measures hierarchical clustering is applied. Here the main aim of clustering is to group the clones with smaller distance, so the clustering process is limited to a distance<=0.2.

Table 4. Similarity matrix with similarity measures.

Simila Methods	M1	M2	M6	M7	M9	M10	M12
M1	0	0.2	0.72	0.4	0.28	0.36	0.36
M2	0.2	0	0.28	0.28	0.22	0.66	0.27
M6	0.72	0.28	0	0.22	0.56	0.96	0.4
M7	0.4	0.28	0.22	0	0.21	0.70	0.23
M9	0.28	0.22	0.56	0.21	0	0.47	0.07
M10	0.36	0.66	0.96	0.70	0.47	0	0
M12	0.36	0.27	0.4	0.23	0.07	0	0

The process of clustering is given in Tables 5, 6, 7 and 8. The result of the clustering algorithm is a group of identical clusters represented as a cluster diagram and dendrogram as in Figures 4 and 5 respectively.

Table 5. Hierarchical clustering step 1.

Similar Methods	M1	M2	M6	M7	M9	M10	M12
M1	0						
M2	0.2	0					
M6	0.72	0.28	0				
M7	0.4	0.28	0.22	0			
M9	0.28	0.22	0.56	0.21	0		
M10	0.36	0.66	0.96	0.70	0.47	0	
M12	0.36	0.27	0.4	0.23	0.07	**0**	0

Table 6. Hierarchical clustering step 2.

Similar Methods	M1	M2	M6	M7	M9	M10/M12
M1	0					
M2	0.2	0				
M6	0.72	0.28	0			
M7	0.4	0.28	0.22	0		
M9	0.28	0.22	0.56	0.21	0	
M10/M12	0.36	0.27	0.4	0.23	**0.07**	0

Table 7. Hierarchical clustering step 3.

Similar Methods	M1	M2	M6	M7	M9/M10/M12
M1	0				
M2	**0.2**	0			
M6	0.72	0.28	0		
M7	0.4	0.28	0.22	0	
M9/M10/M12	0.28	0.22	0.4	0.21	0

Table 8. Hierarchical clustering step 4.

Similar Methods	M1/M2	M6	M7	M9/M10/M12
M1/M2	0			
M6	0.28	0		
M7	0.28	0.22	0	
M9/M10/M12	0.22	0.4	0.21	0

The results show that *m*10, *m*12 and *m*9 are identical and *m*1, *m*2 are identical to each other, whereas the other clones m6 and *m*7 with a greater distance are not identical and they are in separate clusters. Given a similar clone our system efficiently lists out all its identical clones. Identical clones are given the rank, based on their existing cluster and it is shown in Table 9. The ranked clones are indexed and stored in a database as shown in Table 10. The method of indexing helps in interactive identification of clones. If an inconsistent change is made to the software which results in a clone, then all the other clones that belong to the same index are listed out interactively.

Table 9. Clones with ranks.

Clones	M1	M2	M6	M7	M9	M10	M12
Rank	2	2	3	4	1	1	1

Table 10. Clones with index.

Index	Clones
1	M12, M10, M9
2	M1,M2
3	M6
4	M7

Table 11 shows the result of the identical clusters extracted from the sample java system considered for the study. It shows the list of identical clusters for the most Similar (ES) and the More Similar (MS) categories. For the threshold value 0.9-1.00 in most similar category out of 53 clones, 14 identical clusters are generated each having the identical clones. Similarly for a threshold of 0.75-0.8 out of 43 clones, 16 identical clusters are generated for the more similar category. The results also show the identical clones for methods, files and directories under different threshold values.

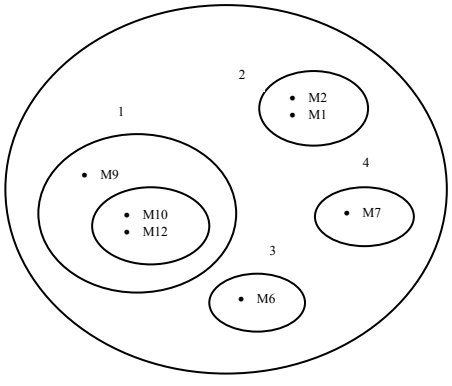

Figure 4. Cluster of identical clones.

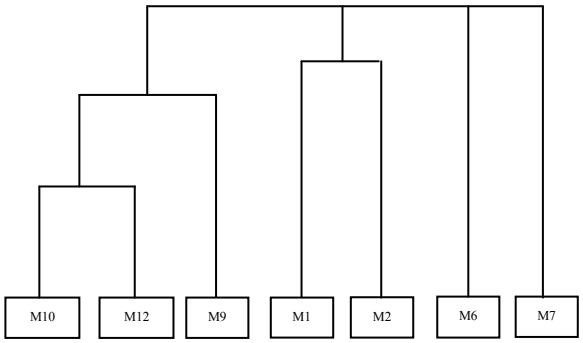

Figure 5. Dendrogram for identical clones.

Table 11. Identical clusters for methods, files and directories.

Threshold Value		Methods				Files				Directory			
		Clone Sets		Identical Clusters		Clone Sets		Identical Clusters		Clone Sets		Identical Clusters	
ES	MS	ES	MS	ES	MS	ES	MS	ES	MS	ES	MS	ES	MS
0.8-0.85	0.6–0.7	57	57	25	23	110	0	26	0	2	0	1	0
0.85-0.9	0.7-0.75	54	42	22	19	99	3	18	2	2	0	1	0
0.9-1.00	0.75-0.8	53	43	14	16	90	3	12	1	2	0	1	0

6. Conclusions

The proposed architecture takes the simple clones and clusters the clones based on the similarity measure. The similarity measure takes two factors for consideration, the similarity distance and the population of clone. These two factors are calculated for all the clones in the clone set. The resultant is a cluster which groups the identical clones from a set of similar clones. The identical clones are ranked based on the cluster. The ranked clones are finally indexed to automatically list the identical clones. The research can be further extended to find the structural similarity. It can also be used to find the clones in other data structures.

References

[1] Abbas O., "Comparisons Between Data Clustering Algorithms," *the International Arab Journal of Information Technology*, vol. 5, no. 3, pp. 320-325, 2008.

[2] Barbour L., Khomh F., and Zou Y., "Late Propagation in Software Clones," *in Proceedings of the 27th IEEE International Conference on Software Maintenance*, Williamsburg, USA, pp. 273-282, 2011.

[3] Cordy R., Dean R., and Synytskyy N., "Practical Language-Independent Detection Of Near-Miss Clones," *in Proceedings of the 14th IBM Centre for Advanced Studies Conference*, pp 1-12, 2004.

[4] Gode N. and Koschke R., "Studying Clone Evolution using Incremental Clone Detection," *Journal of Software: Evolution and Process*, vol. 25, no. 2, pp. 165-192, 2013.

[5] Hemel A., Kalleberg K., Vermaas R., and Dolstra., "Finding Software License Violations Through Binary Code Clone Detection," *in Proceedings of the 8th Working Conference on Mining Software Repositories*, New York, pp. 63-72, 2011.

[6] Koschke R., "Large-Scale Inter-System Clone Detection using Suffix Trees and Hashing," *Journal of Software: Evolution and Process*, vol. 26, no. 8, pp. 747-769, 2013.

[7] Li Z., Shan L., Myagmar S., and Zhou Y., "CP-Miner: Finding Copy-Paste and Related Bugs in Large-Scale Software Code," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp 176-192, 2006.

[8] Miyamoto S. and Terami A., "Constrained Agglomerative Hierarchical Clustering Algorithms with Penalties," *in Proceedings of IEEE International Conference on Fuzzy Systems*, Taipei, China, pp. 422-427, 2011.

[9] Mythili S., Sarala S., "Enhanced Technique to Identify Higher Level Clones in Software," *in Proceedings of the 2nd International Conference on Soft Computing and Problem Solving*," pp. 1175-1182, 2012.

[10] Nguyen T., Nguyen H., Al-Kofahi J., Pham N., and Nguyen T., "Scalable And Incremental Clone Detection for Evolving Software," *in Proceedings of International Conference on Software Maintenance*, Edmonton, pp 491-494, 2009.

[11] Roy C. and Cordy J., "A Survey on Software Clone Detection Research," available at: http://maveric0.uwaterloo.ca/~migod/846/papers/roy-CloningSurveyTechReport.pdf, last visited 2007.

[12] Roy C. and Cordy J., "NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty Printing and Code Normalization," *in Proceedings of the 16th International Conference on Program Comprehension*, Amsterdam, pp. 172-181, 2008.

[13] Roy C., Cordy J., and Koschke R., **"**Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach," *Journal Science of Computer Programming,* vol. 74, no. 7, pp. 470-495, 2009**.**

[14] Schwarz N., Lungu M., and Robbes R., "On How Often Code is Cloned Across Repositories," *in Proceedings of the in Proceedings of the 34th International Conference on Software Engineering*, pp. 1289-1292, 2012.

[15] Smith R. and Horwitz S., "Detecting and Measuring Similarity in Code Clones," available at: http:// research.cs.wisc.edu/ wpis/ papers/codeClonesWorkshop09.pdf, last visited 2009.

[16] Yamashina T., Uwano H., Fushida K., Kamei Y., Nagura M., Kawaguchi S., Iida H., "SHINOBI: A Real-Time Code Clone Detection Tool For Software Maintenance," *in Proceedings of the 16th Working Conference on Reverse Engineering*, Lille, French, pp 313-314, 2009.

[17] Yoshioka S., Yoshida N., Fushida K., and Iida H., "Scalable Detection of Semantic Clones Based on Two-Stage Clustering," available at:

http://sdlab.naist.jp/pman3/pman3.cgi?DOWNLOAD=50, last visited 2011.

Mythili ShanmughaSundaram is a PhD Research Scholar in Bharathiar University, India. She is graduated with MCA, MPhil degree in computer science. She has published and presented papers in various Journals and Conferences. Her areas of interest include software engineering and software testing.

Sarala Subramani is a Assistant Professor, Department of Information Technology at Bharathiar University. She completed her PhD in object oriented software testing, Anna University, Chennai. She joined as a Junior Research Fellow in the Department of Computer Science and Engineering, Anna University in December 2001. She completed her B.Sc Physics in Quiad-E-Millath Women's College, affiliated to Madras University, Chennai and M.C.A in Computer Applications from Madras University, Chennai. She has a teaching and research experience of 9 years and has presented papers in various National and International Conferences. Her areas of interest include software testing, software engineering, object oriented programming concepts, data structures and compiler design.