International Journal of Computer Applications (0975 – 8887)
Volume 93 – No 6, May 2014

Implication of Clone Detection and Refactoring Techniques using Delayed Duplicate Detection Refactoring

M. Deepika
Research Scholar
Department of Information Technology
Bharathiar University
Coimbatore, India

S. Sarala, Ph. D
Assistant Professor
Research Scholar
Department of Information Technology
Bharathiar University

ABSTRACT

Code maintenance has been increased when the similar code fragments is reduced in the software systems. Refactoring is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior based on code, the refactoring mechanism is used to discover the clone detection. The proposed algorithm insists semantic relevance between files, classes and methods towards c# applications. The delayed duplicate detection refactoring technique uses the code analyzer and semantic graph for quickly detect the duplicate files in the application. The implemented clone refactoring technique enhances the Semantic Relevance Entity Detection algorithm which provides better performance and accurate result for unifying the process of clone detection and refactoring.

Keywords

Refactoring; Code clones; Clone detection; Parsing; Source code fragments; Delayed duplicate detection; Abstract Syntax Tree (AST).

1. INTRODUCTION

Besides the existing work refactoring and clone detection enhanced with graph based technology that liberates a new probability of production. Duplicate code is a sequence of source code that occurs more than once within a program or across different programs owned by the identical entity called code clones. Clones are creating the various critical problems in industrial systems. The software system may not be affected on the reason of clones; it seems to be very difficult in future developments. Refactoring has several methods which plays vital role in code clone areas. The refactoring process improves the existing code by changing its internal structure without affects its external behavior [27] [28]. Thus the refactored code is easier to recognize in view of less modification in real time systems. Copying code fragments is reuse by pasting with or without minor modifications are called code cloning and hybrid combination of metric-based approach used for textual comparison of source code [22]. According to the survey four types of clones have been identified [17] and structural, functional, model based clones in codes [7].

Detect the clones is not only fulfill of software development and maintenance. Then fix the clones used the refactoring technique is one of the efficient mechanism for code clones. In this research initially detect the clones in software system and then used refactoring technique to fix the clones. Refactoring result is produced in code with enhanced maintainability and it is regard as a preventive maintenance activity. The refactoring technique have five steps of processes, these are required to perform before applying refactoring in software system [14].

The strategy design pattern technique uses for method is automatically identified the refactoring opportunities towards the Strategy design pattern and gives the quality assessment in experimental evaluation [10]. The technique based on metrics can be used to large process model repositories of automatically identifying refactoring opportunities and it can be conclude automatically detect a number of anti-patterns that can be corrected by refactoring [11].

This work explores the concept of code cloning, types of clones and detection of clones and clones are copied by another one software with or without minor modifications [3]. The Work introduced a language independent method level clone detection using the Rabin-Karp fingerprint representation that is string matching algorithm for identifying duplicated codes. The exception handling refactoring differs from a normal refactoring and it is used for object oriented applications and eliminates the exception handling smells [18]. The relevance measuring algorithm works based on a graph structure between resources within the information retrieval system [20].

C# programming has uncountable defects in compiler process. The interface anomaly has vital significance to destroy the similarity of code. Therefore, the experiments in refactoring uses C# source code fragments to detect the clones which are experimented in the research [4]. This proposed approach also belongs to object oriented refactoring in c# applications. Some techniques have been used for detect and fix the clones in single class. But this proposed method used to detect and fixes the clones in multiple class of software files towards c# applications via semantic graph, Semantic Relevance Entity Detection algorithm and refactoring methods using delayed duplicate detection refactoring technique.

2. RELATED WORK

Several literatures exist in the area of clone detection and refactorings which includes AspectJ Development Tool, CMC, CloneTracker tools that have been used for identifying clones in the software systems [23] [30] and [21]. The research compares the various clone detection techniques and tools based on hypothetical evaluation of clone types scenarios [17]. SHINOBI is a code clone detection modification tool for automatically detects code clones in implicitly and integrated with Microsoft visual studio [18].

CCFinder has found clones by comparing token-by-token and also used metrics on the code clones [29]. CReN tracks the code clones when copy and paste operations occurs in the

Eclipse IDE [24]. The work describes a token and AST based hybrid approach to automatically detecting code clones in Erlang/OTP programs. Clone detector and the refactoring integrate within Wrangler thereby refactoring tool has been developed at Kent for Erlang/OTP [31]. The method CeDAR has proposed with integrated development environment in single class of file towards an Eclipse in the unifying process of clone detection and refactoring [5]. The CP-Miner uses the data mining techniques to identify copy-pasted code and copy-paste related bugs in operating systems [25].

The approach categorizes the Extract Method refactoring opportunities which are interrelated with variable and object of an object-oriented system [12]. The refactoring tool is used to discover the good quality trade-offs in the usage of storage, computation resources of signal processing applications [13]. The graph based relevance measure is used to find the various relations and rules between information resources [32]. Description Graphs explore a general approach to assess the existence of semantic entities in multimedia content [33]. Thus the research examines, clone graphs insist the relevance entities in the source code of C# application.

The Asta method explores code clones by using abstraction syntax trees [26], then compares the greedy and manual approaches and finally refactoring effort model is very useful to software systems [15]. The Refactoring Recommender agent is developed for object oriented system into aspect oriented system in refactorization. The agent uses a Markovian algorithm to detect the types of restructurings is needed to the source code [9].

The empirical study of decision making describes refactoring decisions in the system architecture of embedded software. Technical management and requirements have the less importance on humans [8]. The Adaptive K – Nearest neighbor clustering algorithm is used to perform the clustering in ill- structured software entities refactoring at function level [16]. The pull up method refactoring is used to eliminate the duplicate codes in subclasses. Pre and post conditions produce the guarantee to detect the source code behaviors by refactoring techniques [6].

3. METHODOLOGY

The proposed system uses the semantic graph technique that has been coupled with delayed duplicate detection at memory level.

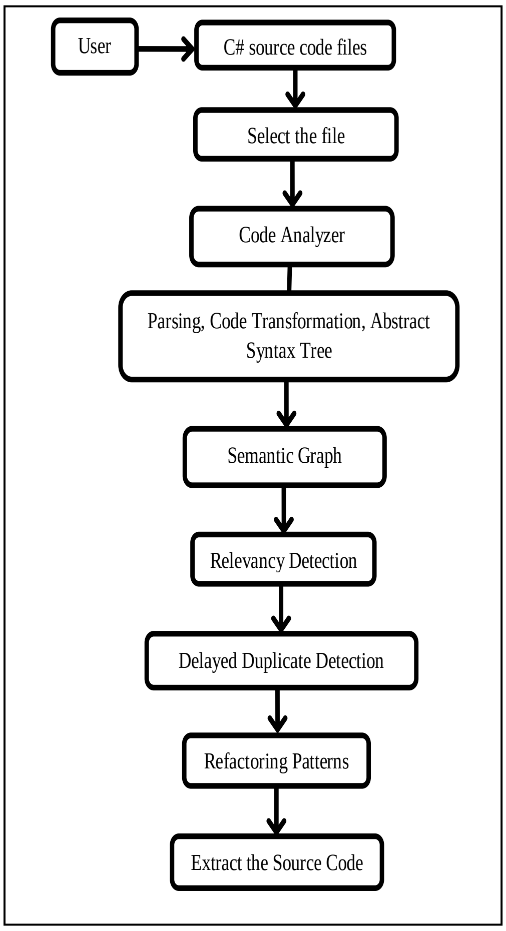

Figure 1: Delayed Duplicate Detection Refactoring

Code refactoring is performed to improve the efficiency of the refactoring technique and to cover more refactoring factors as much as possible.

The figure 1 indicates the refactoring method using semantic graph and delayed duplicate detection technique at memory level. Initially c# code has been loaded and then selects the file for parse to identify the object oriented entities in source code fragments which are namespaces, classes, methods and parameters using the code analyzer. The code clones are identified using the parsed information and syntax tree and semantic graph generation. Besides the semantic graph has been plotted the all methods and parameters for identify the semantic relevance entities.

The delayed duplicate detection refactoring used for reduces the time and memory level in cache performance after the detection of code clones process using the breadth first search technique. It is used for discovers the graph coordinates to store in a hash table. The items of hash table list have been sorted in the ordered list.

It is useful even when all nodes fit in memory, resulting in reduced running time due to improved cache performance. In the standard implementation of breadth-first search in memory, the Open list is stored in a hash table. The DDDR implementation uses a hash table with FIFO queue in memory, reading nodes off the head of the queue, and appending them to the tail.

The new node is generated; it is looked up in the hash table, which often results in a cache miss, since the hash function is designed to randomly scatter the nodes. Once a level of the search is completed, the queue is sorted in memory using an algorithm such as quicksort, and the sorted queue is scanned, merging duplicate nodes.

The advantage of this approach is that the queue is only accessed at the head and tail or at two points in between during quicksort, and hence most memory references will reside in cache, reducing the running time. The quick sort technique applied for time consuming refactoring by considering refactoring techniques.

3.1 Semantic Graph

Initially formed the syntax tree, next objective is measuring the semantic relevance between nodes of the tree structure to detect clones. To discover the node relevance the proposed system intercepts the class wise nodes into a graph structure where the relevance can be calculated as a numerical value and the clone graph is formed.

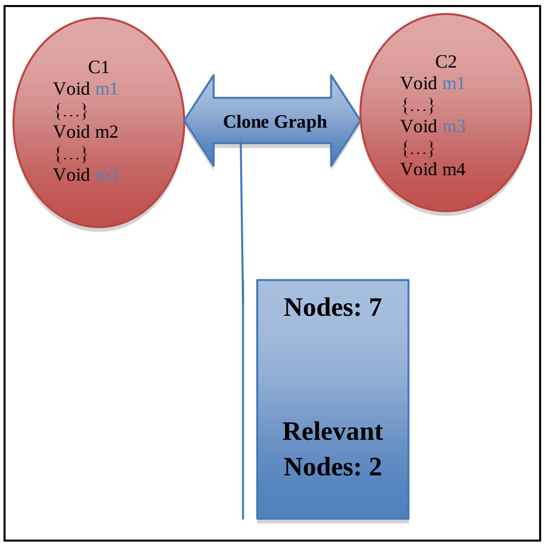

Figure 2: Semantic Relevance of Clones

To estimate the semantic relevance between classes use a semantic graph based relevance measure to exploit various relations and node roles between resources. The figure 2 shows the classes C1 and C2 contains methods which are called as nodes. C1, C2 are denoted as classes and m1, m2, m3, m4,… are denoted as methods.

TABLE 1: Attestation for Semantic Relevance

Method Name	Number of Occurrence(s)
M1	**2**
M2	1
M3	**2**
M4	1
M5	1

Clone graph is drawn with nodes of both the classes. Nodes are 7 and the relevancy node is m1 and m3. Table 1 explains the how to conclude that methods are cloned in software systems.

3.2 Relevance Entities of Code Clones

The research work involves the following relationships are used between the software entities constrained by the object oriented context to detect the different code clones.

```
Input: Clone Graph CGr
Output: OL of refactorable methods
  1. Get the Clone Graph CGr
  2. Construct the Hash Table HT with the CGr Vertex.
  3. Arrange the resultant vertices and optional vertices in CGr.
  4. To get the relevance of REs.
  5. Obtain the Ordered List (OL) of clones
  6. Scan the OL until reaching a vertex with a c smaller than the current C(RE).
  7. Return the OL.
  8. Detect refactoring patterns to clones.
  9. Apply the appropriate pattern in automatically.
  10. Extract the results of input fragments.
```

Figure 3: Semantic Relevance Entity Detection Algorithm

Each relationships are determines a set of source code and clone graph constructs. The entities are following below:

A **Relevance Entity (RE)** represents the programming constructs such as class, methods, expressions, statements and so on. In the given C# class code, RE values are represented in the semantic graph as vertex.

A **Relation (R)** is a feature that requires two or more REs to achieve its same representation or meaning in semantic terms. A Relation lacks any information when it is not associated to any RE.

A **Clone Graph (CG)** assigns REs and Rs to its vertices in order to describe a higher-level RE. REs are linked by Rs using directed edges, in such a way that the neighborhood of the REs is always Rs, and vice versa. The vertices are divided into necessary or meanwhile optional. Necessary vertices must be present in any instance of a RE model. The presence of optional vertices is not mandatory to have an evaluable instance of a RE model.

The figure 3 delayed duplicate detection refactoring algorithm have been indicated the clone graph from AST and then build hast table for all methods of source code fragments. Sequentially the cloned methods of list are refactored using refactoring patterns/methods appropriately. The details of specific refactoring pattern chooses to clones will also be explained in further.

Hash tables are used to store the CG's that can be used as a reference for the detection of REs in C# codes. A confidence value C (RE) expresses the performance of RE which is represented into the code. The confidence values of CG's instances are calculated using the vertex belongs to CG. Relevance of the vertex (r) denotes the relation of the CG instances. Semantic relevance entity detection (SRED) algorithm uses the model instance architecture. Hash tables are used to store the CG's that can be used as a reference for the detection of REs in C# Codes.

The relevancy values are expressed in 0 and 1. The confidence value of CG produces the relevancy values of 0 and 1. The list is first sorted in decreasing order according to their associated c in to an Ordered List (OL). The ordered list is scanned adding optional vertices to the expression until reaching a vertex with a c smaller than the current C(RE). The output list is stored as hash table.

3.3 Refactoring Methods

The entities are loaded and sorted in the hash table. Each relevant methods class names are identified to get the class name. The refactoring technique is decided based on the class. The research work focuses the various methods of refactoring which are pull up method, push down method, extract class method and rename method.

- **Pull Up:** This refactoring is especially useful when subclasses of a certain class share some functionality but each implements it separately. It pulls methods from sub classes to super class.
- **Push Down:** Push down refactoring involves moving a method from a superclass into a subclass.
- **Extract Class:** The Extract Class refactoring is useful once classes happen to overweight with too many methods and its purpose becomes unclear and efficiency high-level. Extract Class refactoring employs creating a new class and moving methods and/or data to the new class.
- **Rename Method** Rename Method is a refactoring that changes a name of a method into a new one that better reveals its purpose.

4. EXPERIMENTAL RESULTS

As a result, the clone analysis and detection performed in c# applications, construct the graph plots using the semantic relevance entity detection algorithm and delayed duplicate detection refactoring technique for reduces the running time. The following table 2 lists the execution id, object oriented programming code of c# applications and class counts.

5. METRICS IN CLONE DETECTION STATES

After the elimination of the all noise, all applications are analyzed for compute the each object oriented metrics in quantitatively. The metrics are computed for following scenarios from that quantitatively analysis: single class, sibling, super class, same method and unrelated class. The figure 4 explains the percentage of clones occurred in object oriented entities.

TABLE 2: Results of Clone Detection

Id	Source Code	No of Classes	No of Methods	No of Same Name Method Clones	No of Clones	Refactorin g Method
E1	AllLink.cs	3	3	4	Two	Rename
E2	Kmeans.cs	5	14	5	Five	Pull Up
E3	Class3.cs	1	3	2	Nil	Rename
E4	ROI.cs	1	19	3	One	Extract
E5	C45.cs	2	4	0	Nil	Pull Down

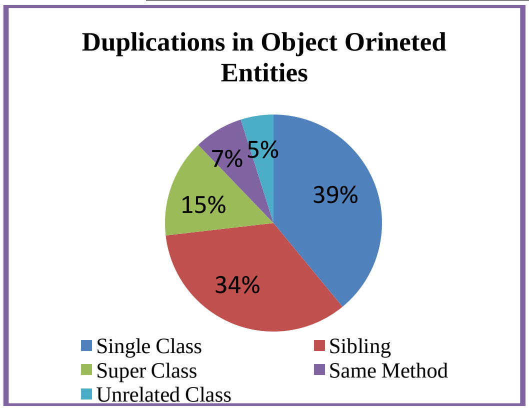

Figure 4: Percentage of Duplications

6. CONCLUSION

Refactoring is a planned and disciplined process of code transformation. In this research work, refactoring mechanism with the technique of delayed duplicate detection refactoring for detect and fixes the clones in multiple classes of c# files towards C# applications and among the help of clone graph, Semantic Relevance Entity Detection algorithm and Delayed Detection Duplication Refactoring technique. This technique is used for reduces the running time due to improved cache performance in the memory level of application. The work has been limited to object oriented source code, because each programming languages have different semantics for applying refactoring methods and fixing the clones. The method introduces automation to identification of namespaces; classes and methods of object oriented source code find out clones using graph plots and eliminate the clones using refactoring opportunities.

The results are identified as analyses and detect the clones in source code. In addition, the nature of the evaluation tasks, metrics made a number of predefined changes to the parameters, perhaps this work been limited to object oriented source code.

7. REFERENCES

[1] S. Sarala, M. Deepika, “Unifying Clone Analysis and Refactoring Activity towards C# Applications”, Proceedings of the IEEE 4th International Conference on Computing, Communication and Methodologies, ISBN: 978-1-4799-3925-1, pp. 1-5, 2013.

[2] S. Sarala, S.Mythili, “A Language Independent Approach for Method Level Clone Detection Using Fingerprinting”, International Journal of Advanced Research in Computer Science, Vol. 3, No. 2, pp. 368-371, 2012.

[3] S. Sarala, S. Mythili, “Experimental Approach of Clone Detection Techniques”, Proceedings of the International Conference on Systems, Methodologies, Automation and Research Trends, pp. 1-6, 2012.

[4] S. Sarala, “Defects Detection in Imperative Language and C# Applications – Towards Evaluation Approach”, Proceedings of the International MultiConference of Engineers and Computer Scientists, Vol. I, pp. 940-944, 2008.

[5] Robert Tairas, Jeff Gray, “Increasing clone maintenance support by unifying clone detection and refactoring activities”, Journal of Information and Software Technology, Elsevier Publications, Vol. 54, pp. 1297-1307, 2012.

[6] Wafa Basit, Fakhar Lodhi, “Preservation of Externally Observable Behavior after Pull Up Method Refactoring”, 15th International Conference on Computer and Information Technology, pp. 309-313, 2012.

[7] Dhavleesh Rattan, Rajesh Bhatia, Maninder Singh, “Software clone detection: A systematic review”, Journal of Information and Software Technology, Elsevier Publications, Vol. 55, pp. 1165-1199, 2013.

[8] Sara Dersten, Jakob Axelsson, Joakim Froberg, “An empirical study of refactoring decisions in embedded software and systems”, Journal of Procedia Computer Science, Elsevier Publications, Vol. 8, pp. 279-284, 2012.

[9] Santiago A.Vidal, Claudia A.Marcos, “Building an expert system to assist system refactorization”, Journal of Expert Systems with Applications, Elsevier Publications, Vol. 39, pp. 3810-3816, 2012.

[10] Aikaterini Christopoulou, E.A. Giakoumakis, Vassilis E. Zafeiris, Vasiliki Soukara, “Automated refactoring to the Strategy design pattern”, Journal of Information and Software Technology, Elsevier Publications, Vol. 54, pp. 1202-1214,2012.

[11] Remco Dijkman, Beat Gfeller, Jochen Küster, Hagen Völzer, “Identifying refactoring opportunities in process model repositories”, Journal of Information and Software Technology, Elsevier Publications, Vol. 53, pp. 937-948, 2011.

[12] Nikolaos Tsantalis, Alexander Chatzigeorgiou, “Identification of extract method refactoring opportunities for the decomposition of methods”, The Journal of Systems and Software, Elsevier Publications, Vol. 84, pp. 1757-1782, 2011.

[13] Calin Glitia, Pierre Boulet, Eric Lenormand, Michel Barreteau, “Repetitive model refactoring strategy for the design space exploration of intensive signal processing applications”, Journal of Systems Architecture, Elsevier Publications, Vol. 57, pp. 815-829, 2011.

[14] Liming Zhao, Jane Huffman Hayes, “Rank-based refactoring decision support: two studies”, Journal of Innovations Syst Softw Eng, Springer Publications, Vol. 7, pp. 171-189, 2011.

[15] Minhaz F. Zibran, Chanchal K. Roy, “A Constraint Programming Approach to Conflict-aware Optimal Scheduling of Prioritized Code Clone Refactoring”, 11th IEEE International Working Conference on Source Code Analysis and Manipulation, pp. 105 – 114, 2011.

[16] Abdulaziz Alkhalid, Mohammad Alshayeb, Sabri Mahmoud, “Software refactoring at the function level using new adaptive K-Nearest Neighbor algorithm”, Journal of Advances in Engineering Software, Elsevier Publications, Vol. 41, pp. 1160-1178, 2010.

[17] Chanchal K. Roy, James R. Cordy, Rainer Koschke, “Comparison and evaluation of code clone detection techniques and tools: A qualitative approach”, Science of Computer Programming, Elsevier Publications, Vol. 74, pp. 470-495, 2009.

[18] Shinji Kawaguchi, Takanobu Yamashinay, Hidetake Uwano, “SHINOBI: A Tool for Automatic Code Clone Detection in the IDE”, Proceedings of the 16th Working Conference on Reverse Engineering, pp. 313-314, 2009.

[19] Chien-Tsun Chen, Yu Chin Cheng, Chin-Yun Hsieh, I-Lang Wu, “Exception handling refactorings: Directed by goals and driven by bug fixing”, Journal of Systems and Software, Elsevier Publications, Vol. 82, pp. 333-345 , 2009.

[20] Sang Keun Rhee, Jihye Lee, Myon-Woong Park, “Semantic relevance measure between resources based on a graph structure”, Proceedings of the International Multiconference on Computer Science and Information Technology, pp. 229–236, 2008.

[21] Ekwa Duala-Ekoko, Martin P. Robillard, “CloneTracker: Tool Support for Code Clone Management”, 30th International Conference on Software Engineering, pp. 1-4, 2008.

[22] G. Anil kumar, Dr. C.R.K.Reddy, Dr. A. Govardhan, Gousiya Begum, “Code Clone Detection with Refactoring support Through Textual analysis”, International Journal of Computer Trends and Technology, Vol. 2, No. 2, pp. 147-150, 2011.

[23] R. Tairas, J. Gray, I. D. Baxter, “Visualizing clone detection results”, Proceedings of the 22nd IEEE/ACM international conference on Automated software engineering, pp. 549–550, 2007.

[24] P. Jablonski and D. Hou, “CReN: a tool for tracking copy-and paste code clones and renaming identifiers consistently in the ide”, Proceedings of the OOPSLA workshop on eclipse technology eXchange, pp. 16–20, 2007.

[25] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, “CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code”, Proceedings of the Sixth Symposium on Operating Systems Design and Implementation, Vol. 6, pp. 289-302, 2004.

[26] William S. Evans, Christopher W. Fraser, Fei Ma, “Clone detection via structural abstraction”, Journal of Software Qual J, Springer Science+ Business Media Publications, pp. 309-330, 2009.

[27] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, “Refactoring: Improving the Design of Existing Code”, Addison Wesley, 1999.

[28] W.C. Wake, “Refactoring Workbook”, Addison Wesley, 2003.

[29] T. Kamiya, S. Kusumoto, and K. Inoue, “CCFinder: A Multi- Linguistic Token-based Code Clone Detection System for Large Scale Source Code”, IEEE Transactions on Software Engineering, Vol. 28, No. 7, pp. 654–670, 2002.

[30] M. Musuvathi, D. Park, A. Chou, D. R. Engler, and D. L. Dill, “CMC: A pragmatic approach to model checking real code”, Proceedings of the Fifth Symposium on Operating Systems Design and Implementation, Vol. 36, pp. 75-88, 2002.

[31] Huiqing Li, Simon Thompson, “Clone Detection and Removal for Erlang/OTP within a Refactoring Environment”, Proceedings of the ACM SIGPLAN workshop on Partial Evaluation and Program Manipulation, pp. 169-178, 2009.

[32] Sang Keun Rhee , Jihye Lee , Myon-woong Park, “Ontology Based Semantic Relevance Measure”, Proceedings of the First International Workshop on Semantic Web and Web 2.0 in Architectural, Product and Engineering Design, pp. 1-6, 2007.

[33] X Giro, F Marques, “Semantic Entity Detection Using Description Graphs”, Proceedings of the Workshop on Image Analysis for Multimedia Application Services, pp. 1-4, 2003.