Preserving Core Components of Object-Oriented Packages while Maintaining Structural Quality

Amarjeet*, Jitender Kumar Chhabra

*Department of Computer Engineering, NIT Kurukshetra, Haryana, 136119, INDIA

Abstract

Different software maintenance activities, carried out from time to time, lead to structural quality degradation. To improve the degraded structural quality of the software system, re-structuring of software entities is desirable and the same can be achieved by using a suitable software clustering technique. Current techniques require too many components (e.g., classes) to be moved between modules (e.g., packages) to achieve high quality software. In such a scenario, the core components of the packages may also move, resulting into loss of identity of the packages. This paper presents a multi-objective evolutionary optimization technique to improve the quality of the existing software while preserving the core components of the package. We evaluate structural quality of six real-world and one random problem instances, using MCA and ECA multi-objective approach.

Keywords: Multiobjective optimization; Software modularization; modularization quality; Search based software engineering;

1. Introduction

Need of faster development of software system, without strictly following design principles usually results into generation of the poor quality system1. Over time to time, regular maintenance of such a system further deteriorates the system quality2. Hence after some maintenance, the system quality deteriorates so much that the future maintenance becomes extremely difficult3. As it has been found that a well-structured software system is easier to understand and maintain4, hence it becomes necessary to restructure the deteriorated system quality, to attain the
original quality. In restructuring process, the components move from one package to another so as correlated component can be placed in the same package and dissimilar components can be placed in different packages. Such restructuring of correlated components into a package is termed as software module clustering. Throughout this paper, we use term component for class, and cluster or module for package. In software system, for a package there are some important classes, which reflect the identity of the package, referred as core components. Hence preservation of such components in package becomes necessary while restructuring the system. In literature, the software module clustering techniques have been proposed, but the drawback is that, they do not preserve the core components during clustering.

There exist many traditional analytical methods to solve the software module clustering problems in literature. These methods perform well for small size software problems but for the large software problem, these methods become infeasible as they cannot solve the problem within reasonable amount of time. The emergence of Search Based Software Engineering (SBSE) concepts has made the complex software engineering problems efficiently and effectively solvable. In software module clustering the search based techniques help to solve problem within reasonable time. In these techniques, software module clustering problem is formulated as search based optimization problem. The search based software module clustering has been used as single objective optimization problem as well as multi objective optimization problem where quality criteria are formulated as objective function. In single objective module clustering optimization, the cluster quality criteria which are to be optimized is formulated into a single objective function. In literature modularization qualities (MQ) function is used as a single objective function which is the tradeoff between the coupling and cohesion of the individual clusters. However, in multi-objective software module clustering many conflicting quality criteria is optimized. Researchers have formulated module clustering problem as multi-objective approach i.e., Equal Cluster-size Approach (ECA) and Maximizing Cluster Approach (MCA). Both objectives contain five objectives in which four objectives i.e., cohesion, coupling, MQ, and number of clusters are common and fifth objective was different. ECA includes, the difference between the maximum and minimum number of modules in a cluster and was used as fifth objective while MCA includes the number of clusters with zero modules as fifth objective. They tried to improve the MQ objective function value of module with optimization of MQ function as primary objective and some other additional supporting conflicting objectives such as cluster size, difference between maximum and minimum number of entities in module etc. The additional conflicting objectives help to explore the search space for primary objective.

Most of the work on software module clustering based on multi-objective evolutionary algorithms, allows a possible movement of each components from one module to any other module to improve the quality of system. From the theoretical point of view, allowing all components to move can distort the original package design identity and refactoring such system can take more efforts. To overcome these issues, we need to apply the some constraints in the original package structure to ensure of core components within their package. So during software module clustering the maintainer needs to identify and keep the core components in their respective packages. In such situation all components in the package are allowed to move except the core components. This observation provides the motivation to propose techniques for the software module clustering to preserve the core components of packages and also improve structural quality of system. In this paper we present multi-objective software module clustering to preserve the core components of package while improving the structural quality of software system. The technique includes core components identification phase and module clustering phase. In core components identification phase we identify the core components using various metrics and in module clustering phase several alternatives of original structure are created and compared to each other on the basis of MQ value for selection of high MQ value.

The remaining part of the paper is organized as follows: Section 2 provides background and relevant material on software module clustering. Section 3 presents the multi-objective concepts of module clustering. Section 4 presents core components of package concepts. Section 5 describes the experimental methods, problem instances used. Section 6 presents the findings of the experimental study. Section 7 concludes the paper.

2. Software Module Clustering: Background

Software system is essentially the collection of software entities and the process of organizing them into module is known as software module clustering. A proper module clustering improves various quality parameters such as understandability and maintainability. To measure the goodness of module quality, various software metrics are
developed in which coupling and cohesion are two most popular basic metrics used to measure the module quality. The software module clustering technique uses such kind of quality metrics to evaluate the module quality. There are many ways to approach the module clustering problem. Early work on software module clustering was based on the traditional analytical methods and these methods are more suitable for small size specific problem where exact solution is highly required. Later consensus based techniques (CBTs) suggest to combine the strength of various analytical methods to solve the module clustering problem. These techniques overcome the deficiencies in individual algorithms and become effective in improving the cluster quality.

As the size of software system increases, the analytical and consensus based techniques become inefficient to solve the software clustering problem. Emergence of search based software engineering makes the application of evolutionary algorithm in software problem easier and solves it efficiently. The software module clustering problem exhibit many characteristics that help to formulate it as a search based software engineering problem. Mancordis et al. earlier formulated the software module clustering problem as a search based optimization problem. The authors have modeled the software structure into module dependency graph (MDG) and partitioned it using Hill-Climbing algorithms as single objective optimization techniques. Following a similar approach, Doval et al. also formulated the module clustering problem as single objective optimization problem. They proposed a genetic algorithm to address the problem. The representation of problem plays an important role in producing effective and efficient solution. Harman et al. have proposed a normalized representation of software structure for a software module clustering, that minimized the size of the search space and improved the outcome of Genetic Algorithms. Kiarash Mahdavi et al. introduced a Multiple Hill Climbing algorithm that improves the efficiency of the module clustering. Harman et al. concentrate on evaluation of fitness functions and demonstrated the robustness of two well known fitness function MQ and EVM module clustering fitness function through an empirical study. The study showed that as the noise in software structure increased the fitness value of both functions degraded smoothly, but EVM fitness function degrades slowly in comparison of MQ fitness function. Bilal Khan et al. proposed the application of self adaptive Evolution Strategy Based Automated Software Clustering Approach (ESBASCA) for a large and complex solution space to improve the structural quality. They compared the approach with a widely used genetic algorithm approach on many real world systems. Their results showed the considerable improvement in terms of effectiveness and efficiency of the solutions for all used real world problem. Praditwong et al. proposed a multi-objective evolutionary algorithm based on a two-archive genetic algorithm which was shown to be more effective than the Hill-Climbing approach. They used two composite objective functions Equal Cluster-size Approach (ECA) and Maximizing Cluster Approach (MCA) for software module clustering optimization. The experimentation results showed that the ECA produced better solutions for both weighted and un-weighted Module Dependency Graph (MDG) compared to MCA composite objective and Hill-Climbing single objective approach. However, the ECA requires processing time twice of Hill-Climbing approach. Barros performed experimentation for evaluation of fitness functions and demonstrated the multi-objective approach by replacing MQ objective with EVM. The experimentation results showed that replacing MQ objective with EVM objective in ECA approach helped finding solution in less time with improved error ratio and generational distance, particularly for large instances.

3. Software Module Clustering Problem as Multi-objective Optimization Problem

In order to formulate software engineering problem as multi-objective optimization problem, the problem representation and objective function need to be defined in proper way. In software module clustering the literatures have represented the software system as a module dependency graph (MDG). An MDG is an graph $G=<V, E>$, where $V$ is the set of nodes and $E$ is the set of edges. Software system representations as MDG, the components are represented as nodes and connectivity between nodes are represented as edges. The MDG is represented in the form of array, where components (i.e., classes) are associated with array index and clusters (i.e., package) are associated with array value. For example a MDG with eight classes and five packages can be represented as the array $\{1, 1, 2, 3, 4, 4, 5, 5\}$; the class 0 and 1 are located in same cluster 1. After the suitable representation of problem, the objective functions need to be defined that can evaluate the cluster quality. The approach of software module clustering problem as a multi-objective problem requires two or more competing and incomparable cluster quality criteria (objective) that can suggest a good module clusters for software system. In this paper we use, two most popular multi-objective approaches Equal Cluster-size Approach (ECA) and Maximizing Cluster Approach (MCA) for software module clustering. The multi-objective optimization for software module
clustering is defined as follows:

\[
\text{Minimize / Maximize } f_m(C), \quad m = 1,2,...,M; \quad C = (c_1, c_2,...,c_n)^T
\]

Subject to

\[
g_j(C) \geq 0, \quad j = 1,...,J; \\
h_k(C) = 0, \quad k = 1,...,K; \\
c_i^{(L)} \leq c_i \leq c_i^{(U)}, \quad i = 1,2,...,n.
\]

where \( f_m(C) \) are \( M \) objective functions that are to be minimized or maximized and the variable \( C \) denotes the vector of \( n \) decision variable. The term \( g_j(C) \) and \( h_k(C) \) represents inequality constraint function and equality constraint function respectively. The last set of constraints \( c_i^{(L)} \) and \( c_i^{(U)} \) shows the lower and upper bound of decision variable. As this optimization task consists of more than one objective function so it may not be possible to determine single best solution. For two clustering solutions \( C_1, C_2 \), clustering \( C_1 \) is said to dominate clustering \( C_2 \) (denoted as \( C_1 \succ C_2 \)) if and only if

\[
\forall m \in (1,...,M) f_m(C_1) \leq f_m(C_2) \land \exists \in (1,...,m) f_m(C_1) < f_m(C_2)
\]

A clustering \( C_1 \in C \) is said to be non-dominated if and only if there is no other clustering dominating \( C_1 \). The set of all non dominated clustering is called the Pareto optimal set and the corresponding set in the objective space is called the Pareto front. The problem representation and objective function plays important role search based optimization hence it needs to be defined very carefully.

4. Core Component of Package

Most successful software systems are often maintained to satisfy the new requirement or better quality. When maintenance is applied to software system, the software developer who is not familiar with the system first acquires the knowledge about the system before the maintenance process. The software developer tries to find out some components in each package which are important for that package and these help to understand the packages more easily. These important components of packages are often referred as core components. The definition of core components for a package can vary according to application. The core components are usually those classes which implements main functionality of that package. Thus the core component indicates the primary purpose of the corresponding package. It is but obvious that such core components must remain confined in the same package, otherwise the identity of the package will get lost. In this paper we consider those components as core components that have the certain level of connectivity within package, or those components which interact more frequently by other classes in the same package. To preserve the core components of the package while maintaining the software quality, we present a technique which can identify the core components in the package and don’t allow them to move out of their packages while module clustering process. To show the concept we define software system by \( S=\langle P, D \rangle \). \( P \) is the set of all packages in system and \( D \) is pair wise connectivity among packages: \( D \subseteq P \times P \). Packages are container of the classes: each package \( p \in P \) contains a set of classes \( C(p) \subseteq C \); \( C \) is the set of all classes and every class \( c \) belongs to only one package \( p(c) \). Core \( (C(p)) \subseteq C \) is set of core classes in package \( p \), connectivity\( (c(p)) \) is the total weight of connectivity from class \( c \) to other classes in package \( p \). The connectivity \( (c_i, c_j) \) represents connectivity weight between class \( c_i \) and \( c_j \).

4.1. Core Components Identification Phase

In this paper we use undirected graph with unit weight edge. To identify core components in the packages we calculate the connectivity weight of each class in package using the following formula.

\[
\text{Connectivity}(c_1(p_i)) = \sum_{j \in p_i} \text{Connectivity}(c_1, c_j)
\]
First we calculate the connectivity (c (p)) of each class in the package. After calculation of connectivity (c (p)) for each class we develop histogram. With the help of the histogram we select those classes that have the connectivity above certain level. We consider these classes as the core classes for that package. In this paper we have considered unit weight for connectivity. For example, we have manual MDG of software system given in Fig. 1. where 25 classes from c1 to c25 are organized into three package p1, p2 and p3. Now let package p1(c) =  {c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12} and their class connectivity as follows.

Connectivity (c1 (p1)) = {c2, c3, c5, c6, c8, c9, c10, c12} =8, Connectivity (c2 (p1)) = {c1, c4, c5, c7, c8, c11} =6
Connectivity (c3 (p1)) = {c1, c4, c6} =3, Connectivity (c4 (p1)) = {c2, c3, c5, c7, c10} =5
Connectivity (c5 (p1)) = {c1, c2, c4, c11} =4, Connectivity (c6 (p1)) = {c1, c2, c6, c8, c12} =4
Connectivity (c7 (p1)) = {c2, c4, c10, c11} =4, Connectivity (c8 (p1)) = {c1, c2, c11} =3
Connectivity (c9 (p1)) = {c1, c6} =2, Connectivity (c10 (p1)) = {c1, c4, c7} =3
Connectivity (c11 (p1)) = {c2, c5, c7, c12} =4, Connectivity (c12 (p1)) = {c1, c6} =2

Next we generate the histogram for each package. We select top x % classes that have the higher connectivity weight in the same package as core classes of package. The criteria of selection of percentage of classes can vary according to application and developers. We have decided to have select 10% of classes that have the highest connectivity. It is believed that the components having higher connectivity are related with many other components of the same package and thus represent the core component of that package. For example the histogram of package p1 is shown in Fig.1. The class 1 is the core class of the package p1.
4.2. Module Clustering Phase

In this phase we use the multi-objective evolutionary algorithm NSGA II to perform the module clustering where the classes except the core classes can move from their package. Hence here we preserve the core components in the module clustering process by not allowing them to move from their original package. For preserving the core components we use the constraints on decision variable along with the objective functions.

5. Experiment Setup

This section provides detail about the experimental methods used to perform the empirical study. The section 5.1 describes the multi-objective algorithms and their parameter used. Section 5.2 describes the problem instances used. Section 5.3 describes techniques for data collection.

5.1. Multi-objective Algorithm and Parameter

This paper uses the Non-Dominated Sorting Algorithms (NSGA-II) as multi-objective evolutionary algorithm for software module clustering. NSGA-II is genetic based multi-objective optimization evolutionary algorithm. Parameter settings for the NSGA II algorithms are the same for both multi-objective approaches (MCA and ECA). In this algorithm, we use the uniform mutation operator and the probability of mutation is determined as $0.04 \times \log_2(N)$, where $N$ is the number of classes in problem instance. For the crossover operator, we use the single point crossover operator; the probability of crossover is 80% for the problem instances of less than 100 classes and 100% otherwise. For the selection operator, we use the binary tournament for all size of problem instances. The NSGA-II algorithm is implemented and executed with the JMetal framework.

5.2. Problem Instance Selection

To demonstrate the concepts we use six real-worlds and one random problem instance. All the real world problem instances are open-source or free-software projects based on the Java programming language with distinct features and sizes. The PF-CDA, an open-source static analysis tool is used to collect the class dependency information. The following Table 1. describes the characteristics of all seven selected problem instances.

| S.N. | Problem Instance | Modules | Dependencies |
|------|------------------|---------|--------------|
| 1    | Java Servlet API | 63      | 131          |
| 2    | JUnit            | 100     | 276          |
| 3    | JavaCC           | 154     | 722          |
| 4    | XML API DOM      | 119     | 209          |
| 5    | Java XML API     | 184     | 413          |
| 6    | DOM4J            | 195     | 930          |
| 7    | Random 100       | 100     | 342          |

5.3. Collecting Results from experiment

For each pair of multi-objective approach (i.e., MCA and ECA) and problem instances, NSGA-II algorithm is executed 20 times. Each execution of NSGA-II algorithm yielded a set of solutions in objective space i.e., Pareto front ($PF_i$). After executing all pairs of approaches and problem instances for 20 times, the Pareto front with best MQ value in each execution cycle is collected. Now mean and standard deviation of best MQ values from each execution cycle are calculated for the analysis purpose.

6. Results and Analysis

Table 2. present the results of the empirical study for real and random problem instances. In this paper first we conducted four experiments. In the first and second experiment, we executed the NSGA II for both approaches MCA and ECA, without core components preservation on all seven problem instance. In third and four experiments,
the same thing is done with core components preservation constraints. In this experiment 10% components that have the higher degree of connectivity in package are considered as core components of the package. The mean and standard deviation of all 20 execution’s best MQ values, before and after core components preservation for each pair of approaches and problem instances are given in Table 2.

| S.N. | Problem Instance | Before Preservation | After Preservation |
|------|------------------|---------------------|--------------------|
|      |                  | MCA(MQ)             | ECA(MQ)            | MCA(MQ) | MCA(MQ) |
|      |                  | Mean Std             | Mean Std           | Mean Std | Mean Std |
| 1    | Java Servlet API | 3.2855 0.1253 | 3.3676 0.0341 | 3.6310 0.1172 | 3.4245 0.9801 |
| 2    | JUnit            | 4.1123 0.1673 | 4.2967 0.0189 | 4.1089 0.2543 | 3.9162 0.0436 |
| 3    | JavaCC           | 3.4517 0.3429 | 3.5123 0.3312 | 3.6184 0.3244 | 3.5367 0.3023 |
| 4    | XML API DOM      | 6.2764 0.2834 | 6.6435 0.1254 | 6.3034 0.2654 | 6.7198 0.2317 |
| 5    | Java XML API     | 10.1743 0.3511 | 10.3945 0.2311 | 10.2464 0.2354 | 10.4312 0.1251 |
| 6    | DOM4J            | 9.1255 0.1524 | 9.2545 0.1023 | 9.1043 0.2495 | 8.2866 0.2011 |
| 7    | Random 100       | 4.3312 0.3211 | 4.5189 0.2760 | 4.5463 0.1353 | 4.6323 0.1930 |

The experiment results show that the mean and standard deviation of MQ values before core components preservation for ECA approach is improved in comparison of MCA approach, in most of the problem instances. Now if we compare the results of MCA and ECA after the components preservation, the ECA approach performs better than the MCA approach in most of the problem instances. If we compare the performance of MCA approach before and after preservation, we find that MCA in after preservation performs better than the MCA before preservation in most of the problem instances. If we compare the performance of ECA approach before and after preservation, we find that ECA approach, after preservation also performs better than the ECA approach, before preservation in most of the problem, except some problem instance given in bold letter. In two problem instance out of seven problem instances the quality deteriorates with minor differences. Hence these observations indicate clearly that we can preserve some core components in their packages and at the same time can also preserve/improve the quality of system.

7. Conclusion and Future Works

This paper proposed a multi-objective software module clustering methodology, to preserve the core components of packages, while maintaining the quality of software. We performed four empirical studies, each with seven problem instances and two multi-objective approaches. For the first two experimentations, mean and standard deviation of MQ objective was calculated for both MCA and ECA approaches, without preservation of core components. These results indicate that the ECA approach performs better than the MCA approach in most of the cases. For next two experimentations also evaluate the mean and standard deviation of MQ objective was computed for both MCA and ECA approaches, with preservation of core components. In these results the ECA approach again performs better than the MCA approach in most of the cases. We compared the system quality, before and after preservation of core components with each MCA and ECA approaches. The results show that in most of the cases after preserving core components, the quality of system does not deteriorate. Hence it can be concluded from the above proposed technique and experimentations that preservation of core-components in object-oriented packages is desirable and the same can be achieved during re-structuring process without compromising on software quality.

References

1. Praditwong K, Harman M, Yao X. Software Module Clustering as a Multi-Objective Search Problem. IEEE Transactions on Software Engineering; 2011. p. 264-282.
2. Constantine LL, Yourdon E. Structured Design. Prentice Hall; 1979.
3. Pressman R. Software Engineering: a Practitioner’s Approach. 3rd ed. McGraw-Hill; 1992.
4. Sommerville I. Software Engineering, 6th ed. Addison-Wesley; 2001.
5. Hutchens D, Basili R. System Structure Analysis: Clustering with Data Bindings. *IEEE Transactions. Software Engineering*; 1985. p. 749-757.
6. Schwanke R. An Intelligent Tool for Re-Engineering Software Modularity. Proc. 13th Int’l Conf. Software Engineering; 1991.
7. Lindig C, Snelting G. Assessing Modular Structure of Legacy Code Based on Mathematical Concept Analysis. Proc. Int’l Conf. on Software Engineering; 1997.
8. Eisenbarth T, Koschke R, Simon D. Aiding Program Comprehension by Static and Dynamic Feature Analysis. Proc. IEEE Int’l Conf. Software Maintenance; 2001.
9. Mancoridis S., Mitchell BS, Ronres C, Chen YF, Ganser ER. Using Automatic Clustering to Produce High-Level System Organizations of Source Code. Proc. Int’l Workshop Program Comprehension; 1998. p. 45-53.
10. Mancoridis S., Mitchell BS, Ronres C, Chen YF, Ganser ER. Bunch: Recovery and Maintenance of Software System Structures. Proceedings of the IEEE International Conference on Software Maintenance; 1999. p. 50-59.
11. Mitchell BS, Mancoridis S. On the Automatic modularization of Software Systems Using the Bunch Tool. *IEEE Transactions. Software Engineering*; 2006. p. 193-208.
12. Doval D, Mancoridis S, Mitchell BS. Automatic Clustering of Software Systems using a Genetic Algorithm. Proc. Of the International Conference on Software Tools and Engineering Practice; 1999.
13. Mahdavi K, Harman M, Hierons RM. A Multiple Hill Climbing Approach to Software Module Clustering. Proc. IEEE Int’l Conf. Software Maintenance; 2003. p. 315-324.
14. Harman M, Swift S, Mahdavi K. An Empirical Study of the Robustness of two Module Clustering Fitness Functions. Proceedings of Genetic and Evolutionary Computing Conference; Washington DC; 2005.
15. Barros M. An analysis of the effects of composite objectives in multiobjective software module clustering. Proceedings of the fourteenth international conference on Genetic and evolution; 2012. p. 1205-1212.
16. Yourdon S, Constantine LL. Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, Yourdon Press; 1979.
17. Forestier G, Arski PG, Wemmert C. Collaborative clustering with background knowledge. *Data and Knowledge Engineering*; 2010. p. 211–228.
18. Kashef R, Kamel MS. Cooperative clustering. *Journal of Pattern Recognition*; 2010. p. 2315–2329.
19. Maqbool O, Babri MH. Hierarchical clustering for software architecture recovery. *IEEE Transactions on Software Engineering*; 2012. p. 759–780.
20. Naseem R, Maqbool O, Muhammad S. Cooperative clustering for software modularization. *Journal of Systems and Software*; 2013. p. 2045–2062.
21. Harman M, Jones BF. Search based software engineering. *Information and Software Technology*; 2011. p. 833–839.
22. Harman M, Hierons R, Proctor M. A new representation and crossover operator for search-based optimization of software modularization. In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference; New York; Morgan Kaufmann Publishers; 2002. p. 1351–1358.
23. Tucker A, Swift S, Liu X. Grouping Multivariate Time Series via Correlation. *IEEE Transactions on Systems, Man, & Cybernetics*; Part B: Cybernetics; 2001. p.235-245.
24. Khan B, Sohail S, Javed MY. Evolution Strategy Based Automated Software Clustering Approach. *Advanced Software Engineering and Its Applications*; 2008. p. 27-34.
25. Harman M, McMinn P, Souza J, Yoo S. Search based software engineering: techniques, taxonomy, tutorial. Empirical software engineering and verification, Springer-Verlag Berlin, Heidelberg; 2012. p. 1–59.
26. Harman M, Masouri S, Zhang Y. Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications. *ACM computing survey*; 2012.
27. Deb K, Pratap A, Agarwal S, Meyarivan T. A Fast and Elitist Multi-objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computing*; 2002. p. 182-197.
28. Durillo JJ, Nebro AJ, Luna F. JMetal: A Java Framework for Multi-objective Optimization. *Advances in Engineering Software*; 2011. p. 760-771.
29. http://jmetal.sourceforge.net/
30. http://www.dependency-analyzer.org/