Survey on Software Modularization Techniques

Sunil Kumar N
Post-Graduate Student
Department of Computer Science and Engineering, Karunya University
India

Bright Gee Varghese
Assistant professor
Department of Computer Science and Engineering, Karunya University
India

Abstract

When software is evolved during the manufacture process, due to poor design decision, it is often hard to understand the packages and to maintain them. This is because they group together classes with unrelated responsibilities. One way to improve the quality of the software is to decompose the package and come up with higher cohesion. This paper is a survey on how package can be re-modularized by using structural and semantic measures. A software maintainer might modify the source code without an insight into the system design. As the software changes and evolves over time, it is inevitable that the undisciplined approach to software maintenance will have negative effect on the quality of the software. Eventually the system structure might change. Appropriate abstractions are needed to understand the structure and to cluster it. Architectural level views must be created directly from the source code.

1. Introduction

During the maintenance of a software system, most of the effort is usually devoted to understanding the structure of the software system. This task is facilitated if a system is well modularized with less coupling and maximum cohesion, making it easier to change it and also to evaluate the side effects of a change. Coupling is the degree of dependency between the modules and Cohesion is the inter-dependency within a single module. Low coupling is the sign of a well-structured software system and a good design. The concept of software cohesion has been defined by, who defined it as the degree to which the internal contents of a module are related. In Object Oriented software, cohesion is usually applied at class level and it can be extended to package level. When combined with high cohesion, it provides high reliability and maintainability.

In the software domain, an important application of cluster analysis is to modularize a software system by grouping together software entities that are similar or related to each other to achieve minimum coupling and maximum cohesion. Entities within a cluster share similar characteristics or features and they are dissimilar from entities in other clusters. As the software changes and evolves over time, it is inevitable that the undisciplined approach to software maintenance will have negative effect on the quality of the software. Eventually the system structure might change. Appropriate abstractions are needed to understand the structure and to cluster it. Architectural level views must be created directly from the source code. A graph of entities and relations in the source code are produced by bunch which is also explained in this paper.

To determine the components and relations in the source code, design extraction starts at parsing the source code. To produce views of the software structure, the parsed code is then analyzed. When the software engineer isolates the subsystem the software structure is taken into consideration, whichever is relevant to his work? The quality of graph partition is evaluated by the approach given that represents the software structure and uses heuristics to navigate through the search space of all the possible graph partition. Several possible heuristic approaches are possible to solve the problem and are surveyed in this paper by taking the referred papers. Software clusters are independent of any programming language and to achieve this we need the source code analysis tool in which directed graph can be obtained from source code.

2. Survey Among Various Modularization Criteria

2.1. Package Coupling

Abdeen H, Ducasse S, Sahraoui HA and Alloui I [1] propose this approach where coupling takes place inside the package. When software evolves to meet environment changes, modularization quality degrades due to environmental factors. Inter package connectivity is optimized automatically by designing and implementing a search-based approach and improve the quality of software modularization. This approach is inspired by the technique simulated annealing which is also search-based. This is similar to annealing process in metallurgy. Now, the objective is local optimization, so simulated annealing is recommended. Automated Object Oriented class design improvement is well suited for its performance. Connectivity among packages is decreased, specially cyclic connectivity among packages. New modularizations from the existing ones were exploited along with several principles of package design quality. Classes are always public and it is a well-known fact that they can be transferred from one package to another. Classes can be interchanged between packages.

Kuhn A, Ducasse S and Girba T [2] say that when the formal information is considered the informal information that is the semantics contained in the source code is overlooked. Developer information is hidden in the code naming and the software as a whole should be understood and the software should be enriched with the developer as well. Linguistic information can be found in the source code and this paper exploits that by using information retrieval system. Identifier names and comments can be found out by using this. Semantic clustering is a technique in Latent semantic indexing which groups the source artefacts by looking at the language containing the similar vocabularies. These groups that are formed are called semantic clusters and they reveal the intention of the code. When two code segments have similar semantic groups, they may have similar tasks to accomplish. The topics are then compared with each other and the similar links between them are found out and some labels are automatically retrieved. As it is based on identifier names, they are language independent. Software analysis is done based in informal information and does not cover up in depth.

Poshyvanyk D, Marcus A, Ferenc R and Gyimothy T [3] propose an approach for impact analysis which uses information retrieval techniques. When there is coupling, it directly impacts the program comprehension and when the strength of coupling is measured it is a direct predictor of fault-proneness, ripple effects and external software quality. Coupling measures are introduced which investigates a new set of conceptual coupling measures during impact analysis and take into consideration how much identifiers and comments relate to each other. Information retrieval techniques are used here for conceptual coupling between the classes where by information we mean the language of the program. New dimensions are captured as a part of coupling measures where Conceptual Coupling Between Classes are indicators of the change in ripple effects in the software. Classes can be effectively ranked using CCBC.

2.2. Software Restructuring

M. O. Keeffe and M. O. Cinneide [4] say that the cost of the software can be reduced by keeping the behaviour of the software intact and changing the design in an improved way. A software tool is proposed which is capable of refactoring object oriented programs which confines to the quality model and formulates the task as a search problem. Refactoring is applied to increase the flexibility, reusability and understandability of the software. This is defined by a contemporary quality model. Well defined quality models can be used to refactor the object oriented programs and one such model is the QMOOD that defines functions from Quality Attribute Indices. However, this was not found to be suitable in softwares that had large number of featureless classes.

C.-H. Lung, X. Xu, M. Zaman, and A. Srinivasan [5] suggest a similar technique as to reduce the cost of software and at the same time keep up the quality of the software. Understandability must be restored and it should be flexible as well. In this paper, cohesion is the major concern and an approach is presented at the functional level. Automatic support is given to identify ill-structured and low cohesive function. The heuristic advice given helps designers to establish how and why to restructure the program. High cohesive sub functions inside a low cohesive function can be identified. This reveals the potential problems in existing code. Functional clusters and non-functional clusters are identified by the developers and the software designers. Singleton clusters have also to be placed and it is upto the developers to decide as to where to place them. Big data structures are used to group different functional activities together.

M. Harman and L. Tratt [6] propose an approach at the design level. As the software system evolves, its structure degrades and as this happens, refactoring aims at improving the quality of the system. Current system combine metrics in a complex fashion and a single sequence of refactoring is produced. Pareto front is produced when multiple runs of search based refactoring system make up a pareto front. Multiple metrics are used to determine the refactorings. Users should come up with a value that maximises the trade-off between metrics most appropriate to them. Direct and Indirect approaches are proposed that define the search based refactoring. It shows that the existing system relies on complex fitness functions and metrics. It is always optional to not to use complex fitness functions to evaluate the metrics, so if we have an alternative to it, it is better to overcome the existing system. Therefore refactoring of software makes the working of software simpler yet effective.

2.3. Modularization Approach

B. S. Mitchell and S. Mancoridis [7] introduce a new concept called bunch. The detailed discussion on bunch is given here in this paper. Appropriate abstractions are created from the structure of the software to simplify the software maintenance activities. There are documented versions of the abstractions but are sometimes out of date and are no longer used. The search space of several open source systems is studied in detail. Bunch's clustering results are highlighted in several aspects where individual clustering results are considered. It points out as to why bunch's clustering results were not at all obvious. Results produced by Bunch were common and structural properties were independent of whether the MDGs represented real systems or not. Large landscapes can be modelled by using large clustering results. Search–based clustering algorithms like Bunch can be evaluated using search based algorithm. Practitioners have to be reliable as to whether they are working perfectly and this is done using systems such as bunch.

Ducasse S, Pollet D, Suen M, Abdeen H and Alloui I [8] show us how packages can be related among each other and how the relations can be demonstrated. Large software are constituted by large number of packages. Many developers fail to understand how packages are positioned and related to each other. Package surface blueprint shows a relationship that a package has with another and makes the job of the developer easier. Packages are represented under the notion of package surfaces. Package surfaces are the group of relationships according to the package they refer to. Inheritance structure of the package is shown along with references made by the packages. Visualization of the packages was successfully done where large applications were given as inputs. Badly designed packages were pointed out. Tests were conducted with several software maintainers.

Mitchell BS and Mancoridis S [9] have given an insight into automatic modularization of software using bunch tool. Appropriate abstraction is needed for the software structure as these systems are large and complex and we need to make it more understandable. Architectural level views are produced by abstraction in the system level directly from its source code. Bunch clustering system is examined in this paper which uses the search technique to perform clustering. Subsystem decomposition is performed by Bunch by partitioning a graph of entities and relations in a given source code. To evaluate the quality of the graph partition, a fitness function is used and a satisfactory solution is found out. The making of views of the software system is demonstrated by Bunch. Simulated annealing cooling schedules and MQ measurement functions are not included.

2.4. Software Clustering

Wu J, Hassan AE and Holt RC [10] compare the clustering algorithms in the context of software evolution. Meaningful subsystems have been obtained to form clusters for which softwares are partitioned to aid maintenance and analysis tasks. Meaningful clusterings for real life softwares are obtained for achieving growth and continual change. Six software clustering algorithms have been considered here. Stability, authoritativeness and extremity of cluster distribution are the three criteria on the basis of which comparisons have been conducted. Clustering techniques are inspired by various batch processing techniques which produce the implementation of the software tasks. The six algorithms are found not to be matured enough for the production of software representing large evolutionary changes. Before these clustering algorithms are ready to be widely adopted, more work needs to be done.

Harman M, Swift S and Mahdavi K [11] present the search based software engineering which are based on the nature of the fitness function that is used to guide the search. The nature of the search space was given an insight into when these highly robust search spaces were given. Search based module clustering was the aim of this paper. Fitness function that is used for software module clustering are compared here. Another fitness function that is applied here is the EVM. But the results show that both metrics are relatively robust in the presence of some external factors such as noise. Highly tentative observations were made when the when the two fitness functions were tested on entirely random graphs. Software engineering can be used as a vehicle to in order to improve the understanding of problems that are usually faced by software engineering.

Andreopoulos B., An A., Tzerpos V., Wang X. [12] propose an approach which have a better idea than the previous works that rarely incorporate in the clustering process of dynamic information such as the function invocation that take place during the runtime. As the software architecture most of the times are multilayerd, but then the clustering algorithms consist of flat system decomposition. In this paper, a clustering algorithm called MULICsoft has been introduced which incorporates both staitic and dynamic information for clustering process to take place. The core elements of each cluster are assigned to the top layer. Experimental results are produced by testing the components in large open source systems. MULICsoft was successful in coming up with decompositions that had meaningful results and were close to the clusters that were produced by experts. And the most important thing was that MULICsoft did not compromise with the quality of the software system.

Kishore C. and Srinivasulu A [13] give us better results on unweighted MDGs because the previous works showed us results only for weighted MDGs. This was because of the low modularization quality. The technique used here was to maximize the number of clusters with the same number of modules. MQ value was increased due to this technique. This paper also describes the Pareto optimality approach for multi objective clustering. This was considered better for unweighted graphs.

2.5. Architecture Reconstruction

Ponisio and Nierstrasz [14] tackle complexity by organizing classes into packages. For a given developer, a particular package may be neither straight forward nor obvious. Misplaced classes are detected by the technique proposed by this paper, by analyzing how client packages access the given provider package. Locality is considered as the degree to which classes are reused by the common clients that appear in the software. A virtualization layout technique is done to support the locality of the classes into packages.

Pollet, Ducasse, Poyet, Alloui, Cˆımpan, and Verjus [15] present an approach which aids in knowing the large applications and maintain them. Some of the software evolves, so the architecture drifts inevitably. Checking the architecture is therefore important. This paper presents us with the technique to reconstruct the architecture.

3. Conclusion

There has been an interest in search based formulation of this problem that captures the twin objective of high cohesion and low coupling. Two novel multi-objective formulations of the software module clustering problem has been taken into consideration in which several different objectives have been represented separately. First Pareto multi objective formulation is presented which shows how the approach can yield superior results. Richer solution spaces afforded by Pareto optimal approach can be used for the task of restructuring and improving modular cohesion and coupling and is able to produce better solutions than existing single objective solution. Increase in performance is obtained by increased computational cost.

4. References

[1] Abdeen H, Ducasse S, Sahraoui HA, Alloui I (2009) Automatic package coupling and cycle minimization.

[2] Kuhn A, Ducasse S, Gîrba T (2007) Semantic clustering: identifying topics in source code. Inf Softw Technol 49(3).

[3] Poshyvanyk D, Marcus A, Ferenc R, Gyimóthy T (2009) Using information retrieval based coupling measures for impact analysis. Empir Software Eng 14(1).

[4] M. O'Keeffe and M. O. Cinn´eide. Search-based refactoring for software maintenance. Journal of Systems and Software, 81(4).

[5] C.-H. Lung, X. Xu, M. Zaman, and A. Srinivasan. Program restructuring using clustering techniques. J. Syst. Softw., 79(9).

[6] M. Harman and L. Tratt. Pareto optimal search based refactoring at the design level. In GECCO'07.

[7] Mitchell BS, Mancoridis S (2006) On the automatic modularization of software systems using th bunch tool. IEEE Trans Softw Eng 32(3).

[8] Ducasse S, Pollet D, Suen M, Abdeen H, Alloui I (2007) Ackage surface blueprints: visually supporting the understanding of package relationships. In: Proceedings of international conference on software maintenance. Paris, France.

[9] Mitchell BS, Mancoridis S (2006) On the automatic modularization of software systems using th bunch tool. IEEE Trans Softw Eng 32(3).

[10] Wu J, Hassan AE, Holt RC (2005) Comparison of clustering algorithms in the context of software evolution.

[11] Harman M, Swift S,Mahdavi K (2005) An empirical study of the robustness of two module clustering fitness functions. In: Proceedings of the 2005 conference on genetic and evolutionary computation. ACM Press, Washington DC, USA.

[12] Andreopoulos, B., An, A., Tzerpos, V., Wang, X (2005) Multiple layer clustering of large software systems. In: Proceedings of 12th Working Conference on Reverse Engineering.

[13] Kishore, C., Srinivasulu, A (2012) Multi-objective approach for software moduleclustering. International Journal of Advanced Research in Computer Engineering& Technology (IJARCET) 2 (3).

[14] Ponisio and Nierstrasz (2006) Using context information to re-architect a system. In Soft. Measurement Eur. Forum.

[15] Pollet, Ducasse, Poyet, Alloui, Cˆımpan, and Verjus (2007) Towards a process-oriented software architecture reconstruction taxonomy.