A Review on various Cohesion and Coupling techniques for software quality improvement.

Er. Pawal Kaur
M.Tech Scholar
Guru Kashi University TalwandiSabo, Bathinda

Er.chamkaur Singh
Assistant Professor
Guru Kashi University TalwandiSabo, Bathinda

Abstract: with the increase in the use of the software product in daily life. The software development process improvement has been in focused of the researchers. Software development is always a human centric activity. There is higher level of chances of errors in the system. Various advance level techniques are required to identify the bugs and improve the quality of the product. There are two main issues that are considered one is cohesion and other is coupling. Cohesion denotes the modularity of the software products. Higher is the cohesion higher is the dependency of the modules to each other. Further various techniques are used based on FUP. It is frequently Usage pattern. This is the technique for identification of inter and intra dependency of modules. It can also be extended to check the inter module FUP for whole package or for whole class.

Keywords: FUB, Inter module, Cohesion, Coupling

I. INTRODUCTION

With increasing growth of software product use in industry and our day to day life, the software development process has gained popularity among researchers and other practitioners. Since software development is a human-centric activity, so, it is prone to undesirable performance and design defects [10]. So, software development process needs to be continuously assessed and evolved over time in order to fulfill customer's requirements and remove other identified defects .This helps in improving the software design and hence the quality of a software system. Cohesion and Coupling being the two important metrics that denotes the quality at structural design level of a software system. The term cohesion is originated from structural design and it refers to how much the various elements of a given modules are related to each other. It is an important indicator of software design quality and the modularity. A higher cohesion value of a module indicates that the given module is providing near single functionality, whereas, a lower value hinders the reuse of a software module. So, a module with higher cohesion is always desirable. Numerous cohesion metrics have been proposed already. These proposed metrics are based on measuring the method to method interaction and member variable references made by them. These metrics do not consider member variable references to outside modules and member variable references made due to nested member function calls, which in our idea is a research gap in accurately measuring cohesion of a module.

An approach to measure cohesion at module level is proposed. The proposed metric, Usage Pattern Based Cohesion (UPBC), measures the usage pattern of member variables among different member functions of a module. Later, based on the measured cohesion metric value, different modules are clustered by using the proposed clustering algorithm called FUP based Clustering (FUPClust).

The approach makes use of usage patterns present among different software elements. The usage patterns considered are extracted from the member function's usage pattern adopted for accessing different member variables present in the software system. The usage patterns extracted also considers the nested function calls statements present in any of the member function definition. The depth of the nested function calls is considered as the threshold parameter in the proposed methodology and it is user defined. The specified threshold value is used to extract the frequent usage (FUP's) patterns for different software elements. Based on the extracted FUP's, the cohesion among different software elements (class/package) is calculated based on the proposed cohesion metric. The calculated cohesion value among software elements is further used to perform clustering. It mainly consists of three steps and their structure is depicted in the figure-1. The first step in proposed methodology is to extract the FUP for each of the software element. The second step calculates cohesion of each software element using proposed cohesion measurement metric. The third step uses the calculated cohesion value to cluster elements into more cohesive elements using the proposed algorithm.

1.1 Frequent Usage Pattern Extraction

This step of the proposed methodology aims at identifying usage patterns present among software elements. The idea behind usage pattern identification is that in a more cohesive software element, the usage of member variables among the different member functions of the same software element is more as compared to outside elements. The

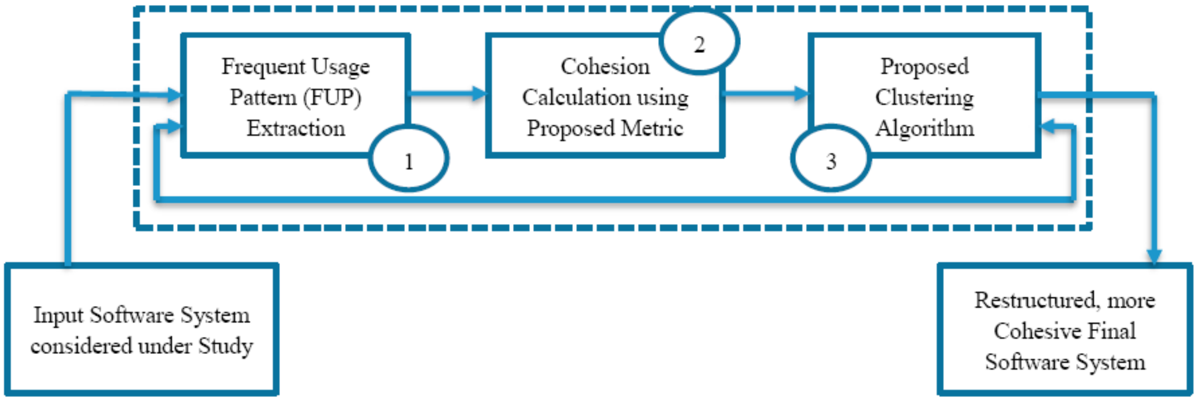
Frequent Usage Pattern (FUP) Extraction
1
Cohesion Calculation using Proposed Metric
2
Proposed Clustering Algorithm
3
Input Software System considered under Study
Restructured, more Cohesive Final Software System

Fig. 1.1 Proposed Methodology for Cohesion[1]

identification of the usage patterns is done by statically analyzing the source code of each of the software element. The usage patterns of a member function consist of a set of member variables directly or indirectly (through a call to other member functions of same of different software element) accessed and modified by it. During the process of extraction of usage pattern, a threshold limit is imposed on the depth of the indirect usage due to function calls. The usage pattern of a software element called FUP is obtained by grouping usage patterns of each of its member functions. Consider a software element Ei that consists of m member variables M1, M2… Mm and n member functions F1, F2… Fn. Suppose the usage pattern of F1 = {Mi, Mj … Mk}; F2 = Φ; -----; Fn = {Mp, Mq … Mt}, then, the frequent usage pattern of Ei is obtained as a set of distinct member variables, FUP = {F1 U F2 U …. U Fn}. It is illustrated by taking the following suitable hypothetical software system example as shown in figure-2. In this example, the usage pattern for member function F1 = {M1}, F2 = {M1, M2}, F3 = {M3}, F4 = {M4, M5, M7}, F5 = F6 = {M5, M7}. Here, the usage pattern for F4 consist of direct usage as M4 and indirect usage consisting of M5 & M7 due to nested functions call to F5 & F6. Similarly, the usage patterns for rest of the member functions can be defined based on same pattern, e.g. the usage pattern for F7 = Φ. Finally FUP of every software element are represented in the form of a vector Vi that denotes the usage pattern of a given software element inside the whole software system. The size of vector Vi is equal to the total number of member variables defined and used inside the considered system and is defined as follows

$$V_i[j] = \begin{cases} 1, & if\ M_j \in FUP\ (E_i) \\ 0, & if\ M_j \notin FUP\ (E_i) \end{cases} \quad \textbf{(1)}$$

Here, Mj is the member variable defined inside the software system.

II. LITERATURE SURVEY

Yourdon et al.(2015) define the coupling for an object-oriented software as the degree to which different modules are interdependent on each other.

Briand et al.(2014) propose a structural based unified framework to measure cohesion in an object-oriented software system and proposed a cohesion metric *Coh* that counts attribute references and sharing among the methods of a class.

Bansiya (2014) defines cohesion in terms of coupling by proposing a coupling metric Direct Class Coupling (DCC) which counts the total number of classes that are directly related to a given class.

Chidamber et al.(2015) propose a metric suite that also measures cohesion as LCOM (Lack of Cohesion among Methods) metric which measures the sharing of member variables among different pairs of methods of a class.

Li and Henry(2016) proposes a cohesion metric LCOM3 by extending the work and representing the system as an undirected graph. They represented each class method as a node in the graph and member variables sharing as an edge in the graph. They measured class cohesion as the total number of strongly connected components in MDG (Module Dependency Graph).

Hitz and Montazeri(2017) proposes another cohesion metric LCOM4 by representing the system as a graph in which the nodes represents the methods and edge between any vertices denote that they are accessing the same attribute.

Henderson et al.(2016) give the latest proposed metric LCOM5 in LCOM metric series. This metric gives cohesion value of zero (0) if methods use only member variables of the class and it gives a value of one (1) if every method uses only one member variable of the class.

Bieman and Kang's(2015) also proposes two sets of cohesion metrics known as tight class cohesion (TCC) and loose class cohesion (LCC). They calculated TCC as the ratio of a total number of pairs of member functions with no sharing of member variables to a total number of pair of direct member functions which share at least one member variable among

III. COMPARATIVE ANALYSIS

Author	Year	Technique	Future work
Yourdon et al.	2015	This paper has proposed a technique based on coupling on object oriented program.	This has not considered intra module inter dependency.
Briand et al.	2014	This paper has proposed a technique based on unified framework to measure cohesion.	This paper has not considered the independent module FUP.
Bansiya (2014)	2014	it simply count the number of classes which are directly be dependent on the given class.	this paper has not considered packages interdependency.
Chidamber et al.(2015)	2015	this paper has proposed a technique checks the sharing of the data members amongst different modules.	this paper has not considered the member functions.
Li and Henry(2016)	2016	this paper has used a technique which has evaluated the whole program by considering the undirected graph.	but have not being specific for any specific class or package.
Hitz and Montazeri(2017)	2017	this paper has enhanced approach in which graph edges are representing the sharing or attributes.	still further FUP can be evaluated.
Henderson et al.(2016)	2016	this paper has proposed a technique based on setting the cohesion value to either 0 or 1 depending upon the data members count.	but this have not considered any further enhancement for usage based metrics.

IV. Conclusion

From the study of various research papers it is clear that the complexity of the software can be reduced by systematically applying the techniques. In current research paper the FUP is calculate for both inter and intra module testing is done. So that those components which need to recognized and call again and again those repetitive components so that shifting time from one module to other module can be reduced. It is the system of recognize the module structure and making things to works smooth. So that the substantial time for system working can be reduced.

V. Future Work

Current research paper is based on evaluating the intermodal interdependency. In future this work can be extended by simply increasing the intra module dependency check.

REFERENCES

[1] L. Yourdon and M. Badri. (2015) "A Proposal of a new class cohesion criterion: an empirical study."Journal of Object Technology, 3 (4).

[2] J. Bansiya. (2014) "A Hierarchical Model for object-oriented Design Quality Assessment." IEEE Transaction on software engineering, 28(1). [3] J. M. Bieman and L. M.(1994) "Ott. Measuring functional cohesion." IEEE Transactions on Software Engineering, 20(8):644–657.

[4] J. Briand and B. Kang.(2014) "Cohesion and reuse in an object-oriented system." Proceedings of the 1995 Symposium on Software Reusability, Seattle, Washington, United States,259–262.

[5] C. Chidamber and E. Kidanmariam.(2015) "Metrics for class cohesion and similarity between methods." In Proceedings of the 44th annual Southeast regional conference, Florida, ACM, 91–95.

[6] Li. Briand, K. E. Henry, and S. Morasca.(2016) "On the application of measurement theory in software engineering." Empirical Software Engineering, 1:61–88.

[7] Hitz and Montazeri(2017) "A Unified Framework for Cohesion Measurement in Object-Oriented Systems." Software Metrics Symposium, Proceedings, Fourth International, 43-53.

[8] Henderson and C.F. Kemerer, C.F.(2016) "A metrics suite for object oriented design." IEEE Transactions on Software Engineering, 20, 476–493.

[9] Bieman and Kang's and S. L. Pfleeger.(2015) "Software metrics - a practical and rigorous approach." (2. ed.). International Thomson.

[10] A. Fuggetta.(2000) "Software process: a roadmap," in Proc. Conf. on The Future of Software Engineering, Limerick, Ireland, 25–34.

[11] M. Harman, S. Danicic, B. Sivagurunathan, B. Jones, and Y. Sivagurunathan.(1995) "Cohesion metrics." In 8th International Quality Week, San Francisco pages, 3(2), 1–14.