Enhanced Cohesion Of A Software By Performing Usage Pattern Of Inter Module Based Clustering

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

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

Abstract: with the advancement in the computer technology new and advanced tools are coming into picture. Because the new software with advanced development tool perform various complex tasks efficiently. Software development is human centric task. It involves large number of people who contribute to the development. Because it involves multiple persons can be prone to various types of discrepancies into the system. These inefficiencies are various design defects. 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. These metrics do not consider member variable references to outside modules and member variable references made due to nested member function calls, in proposed system various types of inter module dependencies can be checked. So that one module can be shifted from one class to other class. Will reduces the overall module execution time and also reduces the memory space requirement. So in proposed system the inter module and Inter package frequent usage pattern can be identified.

Keywords: FUP, Inter, Intra, Time, Space.

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

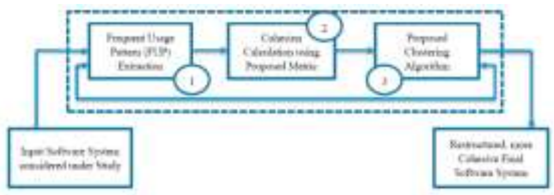
Proposed Usage Pattern (FUP) Extraction
Cohesion Calculation using Proposed Metric
Proposed Clustering Algorithm
1
2
3
Input Software System considered under Study
Restructured, more Cohesive Final Software System

Fig. 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

As the popularity of object-oriented software development is increasing, there is a greater need for software design metrics which are capable of measuring the software design quality. Cohesion is one such key design principle in software engineering and in this direction, numerous cohesion metrics have been already proposed.

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

[2] 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.

[3] 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.

[4] 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.

[5] 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).

[6] 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.

[7] 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.

[8] 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. PSEUDO CODE

INPUT

1. Total number of software elements in Original system is N.

2. Vector representation of FUP of different Modules

3. Compute the Cohesion Values of different modules.

STEPS:

```
1.    Iterate=true
2.    Cohesion= Cohesion(Ei)/|E|
3. Total_Package = Total_Package – 1
4. Compute new overall cohesion value of a software system i.e. cohesion’
5. If (Previous_Cohesion == COHESION’ )
6. ITERATE = FALSE
7. Else
8. Previous_Cohesion = COHESION’ }
```

IV. ALGORITHM

Step1 Input the large program taken from the Github library. These software are JUnit and Hospital Automation.

Step2 Break the whole large program into various packages. Later on each package will be sub divided into various modules and classes.

Step3 Identify the Inter and Intra module dependency Frequent usage pattern.

Step4 Shift the module from one class to other class or from one package to other package to reduces the execution time and memory space requirement.

Step5 Measure the FUP for improved system.

Step6 Evaluate the time and space parameters.

V. FLOWCHART

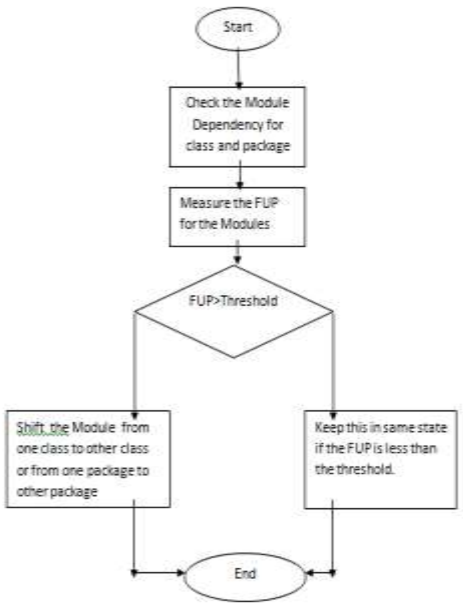
Start
Check the Module Dependency for class and package
Measure the FUP for the Modules
FUP>Threshold
Shift the Module from one class to other class or from one package to other package
Keep this in same state if the FUP is less than the threshold
End

Fig. 2 Flowchart

VI. RESULTS AND DISCUSSIONS

6.1 Intra Module complexity of different parameter

Table 1 Intra Module Complexity

Intra module Complexity						
used variables	Unused Variables	Loops	Tokens	Lines	Packages	Total
2.7	0	4.5	30.6	23.4	1.8	63
3.6	0	0.9	20.7	16.2	1.8	43.2
1.8	0	0.9	15.3	12.6	0.9	31.5
3.6	0	1.8	22.5	17.1	1.8	46.8
2.7	0	4.5	26.1	18.9	1.8	54
2.7	0	2.7	22.5	17.1	0.9	45.9
9.9	0	0	26.1	16.2	1.8	54
8.1	0	0	26.1	18	1.8	54
0.9	0	1.8	21.6	18.9	0.9	44.1
					Average	48.5

Table shows the Intra module complexity based on different parameters. These parameters are like for used variables, Unused variables, Loops, Tokens, Lines, Packages etc. These intra module dependency will be measuring the discrepancies into the designing the modules of the same software.

6.2 Inter Module complexity of different parameter

Table 2 Inter module Complexity

Inter Module Complexity						
used variables	Unused Variables	Loops	Tokens	Lines	Packages	Total
3	0	5	34	26	2	70
4	0	1	23	18	2	48
2	0	1	17	14	1	35
4	0	2	25	19	2	52
3	0	5	29	21	2	60
3	0	3	25	19	1	51
11	0	0	29	18	2	60
9	0	0	29	20	2	60
1	0	2	24	21	1	49
					Average	53.88889

Table shows the Inter module complexity based on different parameters. These parameters are like for used variables, Unused variables, Loops, Tokens, Lines, Packages etc. These intera module dependency will be measuring the discrepancies into the designing of the the modules of the same software.

6.3 Complexity comparison

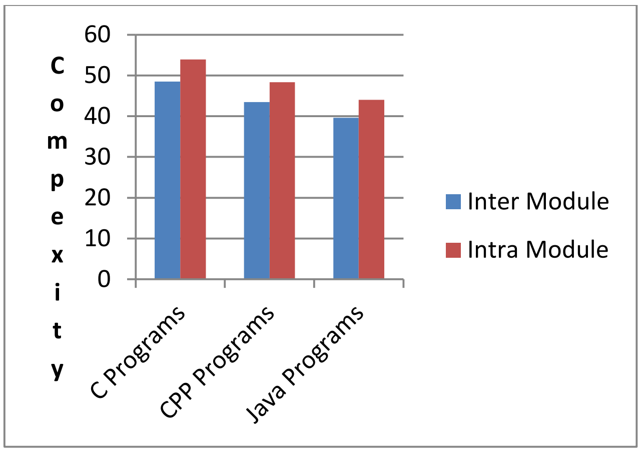

Fig. 3 Complexity Comparison

This graph shows the complexity comparison for the inter and Intra modules dependency. Using Inter modules dependency FUP the complexity of the software can be reduced substantially. That means the software discrepancies is more dependent on the inter module integrity compared to the intra module. This graph shows the average complexity of the inter module and intra module complexity of different types of c, c++ and java programs. Inter module dependency check has less complexity than the inter module.

VI. CONCLUSION

Software designing is the primary issue for the system designing. The correct system design with less inter and intra module cohesion and coupling will reduces the complexity of the algorithm. Also it will reduces the design time complexities of the software cohesion and coupling. These complexities are main reason for the deteriorated performance of the software. Various modules lies into the same class or different class are having higher FUP. Means for there collective execution they are dependent on each other. This type of cohesion is called as intra cohesion. In some cases the module cohesion also stands for two different packages. That can be called as inter module. This type of relative cohesion entity leads to more time and storage wastage. In current research inter module complexity and FUP has to be evaluated. once the relative strength will be improving the system performance will also be improving. So the time and space complexity reduction by considering inter module FUP has been more success full compared to intra module dependency.

VII. FUTURE WORK

In current research software complexity will directly be dependent on the system integration. How there FUP for inter and intra module cohesion will increases the complexities. In future a integrated approach can be considered. Which can collectively increases the performance of the software.

VIII. 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.

[12] B. Henderson-Sellers.(1996) “Object-Oriented Metrics Measures of Complexity.” Prentice-Hall, Inc., Upper Saddle River, NJ.

[13] M. Hitz and B. Montazeri.(1995) “Measuring coupling and cohesion in object-oriented systems.” Proceedings of the International Symposium on Applied Corporate Computing, 25–27.

[14] B. Kitchenham and S. Pfleeger.(1996) “Software quality: the elusive target,” IEEE Softw., 13(1), 12–21.

[15] Kalantari, S., Alizadeh, M. and Motameni, H..(2015) “Evaluation of the reliability of object-oriented systems based on Cohesion and Coupling Fuzzy Computing”. Journal of Advances in Computer Research, **6(1)**, 85-99.

[16] D. I. K. Sjoberg, T. Dyba, and M. Jorgensen.(2007) “The future of empirical methods in software engineering research.” in Future of Software Engineering (FOSE), Minneapolis, 358–378.