Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 125 (2018) 740–746

Procedia Computer Science

www.elsevier.com/locate/procedia

6th International Conference on Smart Computing and Communications, ICSCC 2017, 7-8 December 2017, Kurukshetra, India

Improving Cohesion of a Software System by Performing Usage Pattern Based Clustering

Amit Rathee[a*], Jitender Kumar Chhabra[b]

[a,b]*Department of Computer Engineering, National Institute of Technology, Kurukshetra- 136119*

Abstract

Increasing the software design quality is a key research challenge in object-oriented software development system. Cohesion is one of the key spect that helps to evaluate the quality and modularity of a software system at the design level. It helps to create software components that are directly reusable to the industry because of their less dependence on other components. In this paper, a new cohesion metric for object-oriented software, named as Usage Pattern Based Cohesion (UPBC), is proposed which is computed at the module level. This paper considers class as a module initially and subsequently group of classes (i.e. a package) is considered as a module with an aim of improving overall cohesion. This metric utilizes the Frequent Usage Patterns (FUP) extracted from different member functions interactions to capture the cohesiveness of the module. Further, the measured cohesion value is used to perform clustering of modules in order to increase cohesion and decrease coupling among modules simultaneously. The clustering is performed by using a newly proposed clustering algorithm called FUPClust (Frequent Usage Pattern based Clustering) based on FUP interactions among modules. The proposed approach is applied to two Java software systems and the results obtained show a significant improvement in the cohesiveness of the software system.

Peer-review under responsibility of the scientific committee of the 6th International Conference on Smart Computing and Communications.

Keywords: : Software Engineering; Clustering; Frequent Usage Patterns (FUP); Cohesion.

1. Introduction

With increasing growth of software product use in industry and our day to day life [14], the software development process has gained popularity among researchers and other practitioners [16] [20]. Since software development is a

* Corresponding author. Tel.: +91-999-177-1143
E-mail address: [a]amit1983_rathee@rediffmail.com, [b]jitenderchhabra@gmail.com

1877-0509
Peer-review under responsibility of the scientific committee of the 6th International Conference on Smart Computing and Communications
10.1016/j.procs.2017.12.095

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 [10]. 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 [15]. The term cohesion is originated from structural design [21] 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 [7]. Numerous cohesion metrics have been proposed already [3] [5] [6] [9] [11] [17]. 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.

In this paper, 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 research contributions of this paper include:

1. To propose a cohesion metric UPBC, that measures the cohesion at module level by utilizing the FUP's identified for the given module. The FUP measures the extent to which a given module references the member variables inside and outside to it. The FUP is calculated at method level and overall FUP for the module is calculated based on FUP calculated for the member functions defined inside the module.
2. To propose a clustering algorithm FUPClust that regroup the modules by doing clustering based on the measured cohesion values of different modules.

The rest of this paper is structured as follows: Section 2 gives description regarding the literature survey, section 3 gives detailed description of the proposed. Section 4 describes the experimentation and results and finally section 5 describes the concluding remarks and future works.

2. 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 [3] [5] [6] [9] [11] [17]. Yourdon et al. define the coupling for an object-oriented software as the degree to which different modules are interdependent on each other [22]. Briand et al. [7] 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 [2] 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. [8] 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 [18] proposes a cohesion metric LCOM3 by extending the work of [8] 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 [13] 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. [12] 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 [4] 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

them. Similarly, they calculated LCC as the ratio of a total number of different pairs of member functions that share common attribute usage either directly or indirectly to the total number of pair of member functions defined inside a class. The authors have considered only public member functions while measuring these metrics. Bonja et al. [5] similarly introduced a cohesion metric called the Class Cohesion (CC) Metric based on the similarity among the pair of member functions of a class in using member variables of the class. The authors have also validated their metric by fulfilling Weyuker's set of properties. Similarly, Badri et al. [1] propose DCD and DCI cohesion metrics by measuring method invocations based on member variable usage among them either directly or transitively through method calls.

In summary, various cohesion metrics are proposed in the literature that are structure and/or graph based. Most of the proposed metrics consider member variables used only inside the given module and they do not consider references to member variables originally belonging to outer modules while computing cohesion. The metrics like LCOM1, LCOM2 & LCOM3 are non-standard in nature as their upper bound is not fixed and it depends on the total number of methods in the class. Also, they do not consider the nesting sequence of calls made to other function into account. These limitations are solved in our proposed approach.

3. Proposed Methodology

This section of the paper gives a detailed description of the proposed methodology. The proposed approach makes use of usage patterns present among different software elements. The usage patterns considered in this paper 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 as discussed below further in this section. The calculated cohesion value among software elements is further used to perform clustering based on the steps of the proposed clustering algorithm as mentioned below in this section. The proposed methodology 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 algorithmic steps. These three Steps are repeated recursively and are discussed in detail in the following three subsections:

3.1. 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

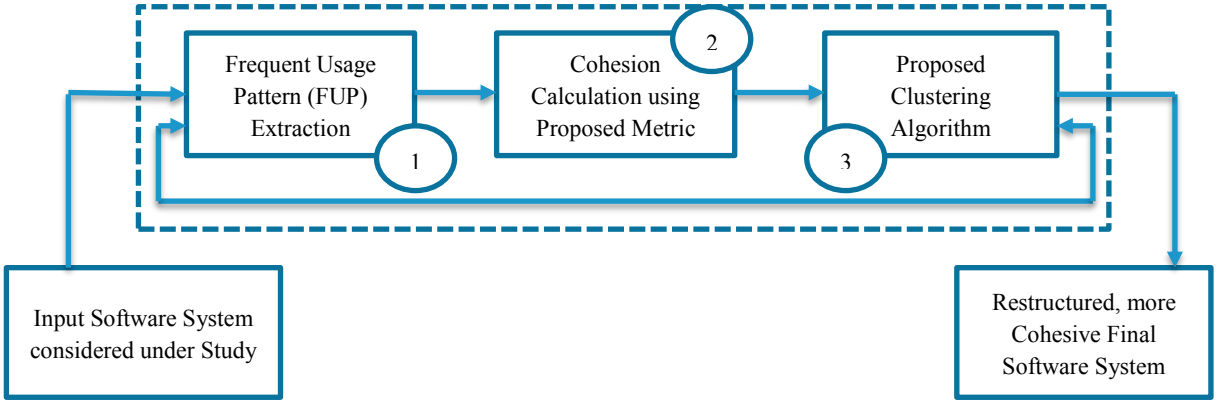

Figure-1: Proposed Methodology for Cohesion

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 F_1 = {M1}, F_2 = {M1, M2}, F_3 = {M3}, F_4 = {M4, M5, M7}, $F_5 = F_6$ = {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 V_i that denotes the usage pattern of a given software element inside the whole software system. The size of vector V_i is equal to the total number of member variables defined and used inside the considered system and is defined as follows:

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

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

3.2. Cohesion Calculation

This step of the proposed work aims at measuring cohesion value of different software elements of the software system. To measure it, a cohesion metric derived from FUP of the software element is proposed. The proposed metric is based on two measurements: IN (Ei) and OUT (Ei) degrees. The in-degree IN (Ei) measures the degree to which a given software element makes use of FUP's within itself. For a given software element that contains total m member functions: M1, M2 … Mm, the in-degree is defined as a count of the elements of the set consisting of FUP's of every member function that is defined inside the given software element. Mathematically it is represented as follows:

$$IN(E_i) = \bigcup_{i=1}^{m} FUP\ (M_i) \quad (2)$$

Here, FUP (Mi) is the set of member variables that represent the frequent usage pattern of method Mi.

Similarly, the out-degree OUT (Ei) measures the degree to which a given software element makes use of FUP's defined in other software elements. For a given software element that contains total m member functions: M1, M2…

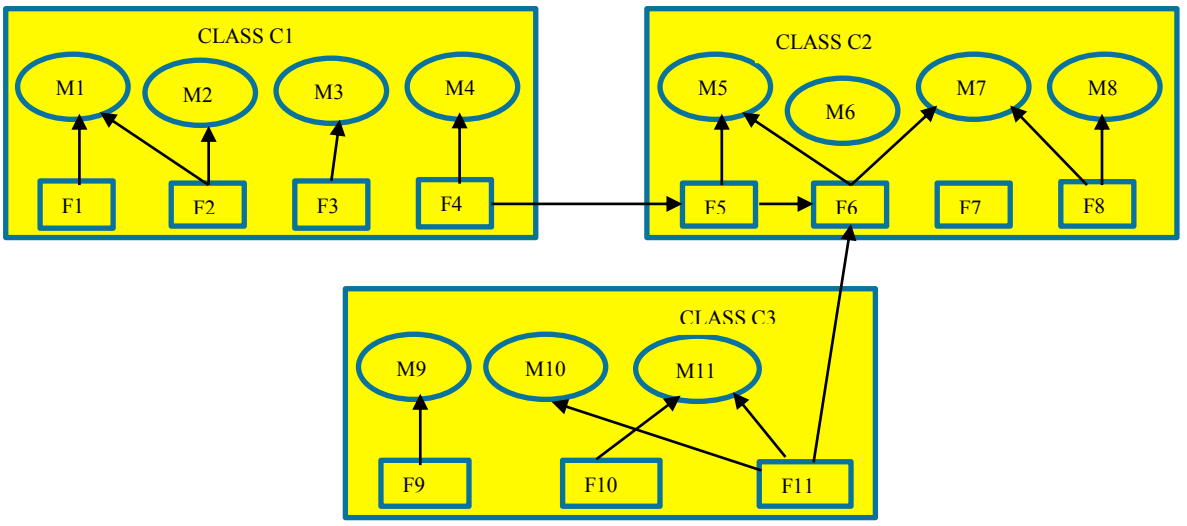

Figure-2: Sample classes showing interactions among them.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11
C1	1	1	1	1	1	0	1	0	0	0	0
C2	0	0	0	0	1	0	1	1	0	0	0
C3	0	0	0	0	1	0	1	0	1	1	1

Figure-3: Vector Representation of FUP's of different modules of considered example

Mm, the out-degree is defined as a count of the elements of the set consisting of FUP's of every member function defined outside the given software element. Mathematically it is represented as follows:

$$OUT(E_i) = \bigcup_{i=1}^{m} FUP\ (M_i) \tag{3}$$

Based on these two measurements, the proposed metric is defined and formulated as shown in the equation below:-

$$Cohesion\ (E_i) = \frac{IN(E_i)}{IN(E_i) \ + \ OUT(E_i)} \tag{4}$$

Here, the idea is that a software element is supposed to have higher cohesion if and only if it makes use of FUP's within itself as compared to the use of FUP's outside its boundary. The overall cohesion of a software system is calculated as the average of cohesion values of each module of the software and it is represented as follows:-

$$UPBC = \sum_{i=1}^{|E|} Cohesion\ (E_i) \Big/ |E| \tag{5}$$

Using the above defined metric and vector representation of FUP's as shown in figure-3, the cohesion value for C1 considered in figure-2 is = $4/(4+2) = 0.66$. Similarly, the cohesion value of other classes can be calculated.

FUPClust Algorithm

INPUT:

1. Total Number of Software Elements in original system i.e. N
2. Vector representation of FUP of different modules
3. Computed Cohesion Values of different modules

STEPS:

// examine the vector representation of FUP's of every software element in order to find pair of vector having maximum usage pattern similarity in order to cluster them together

1. ITERATE = TRUE
2. // Compute overall cohesion of a given software system as the average of individual module's cohesion value

$$\text{COHESION} = \sum_{i=1}^{|E|} \text{Cohesion}\ (E_i) \Big/ |E|$$

3. Total_Package = N *// total number of software elements in the original system*
4. Previous_Cohesion = COHESION
4. While (ITERATE == TRUE) {
7. MAX_LEN = 0; PAIR = Φ *// set representing two most similar* elements in terms of FUP's
8. For V_i : 1 to Total_Package {
9. For V_j : 1 to Total_Package {

// calculate the pair of vector representing two software elements that are most similar in the usage of patterns

10. Perform AND operation between V_i and V_j and calculate the length L as:
L = total number of 1's present in new vector obtained after performing AND operation. *// more the value of L, more is the closeness among the usage of FUP's*
11. If (MAXLEN < L) {
12. MAXLEN = L; PAIR = {i, j}
13. } } }
14. *// create new vector representing the new cluster Vector* V'_{ij}

To obtain new Vector $V'_{ij,}$ perform the XOR operation between two vectors represented by set PAIR as:

$$V'_{ij} = V_i \oplus V_j$$

15. Total_Package = Total_Package – 1
16. Compute new overall cohesion value of a software system i.e**. COHESION**'
17. If (Previous_Cohesion == COHESION')
18. ITERATE = FALSE
19. Else
20. Previous_Cohesion = COHESION' }

Figure-4: Proposed Clustering Algorithm.

3.3. Proposed Clustering Algorithm

This step of the proposed approach aims at clustering different software elements based on their measured cohesion value into more cohesive structure represented in the form of another cluster obtained after performing clustering. This paper proposes a new clustering algorithm which is based on obtained cohesion scores obtained in the previous step. The proposed algorithm is recursive in nature and in each iteration, it combines two most similar software elements together based on the usage pattern similarity. The different steps of the proposed clustering algorithm are as shown in figure-4.

4. Experimentation & Results

This section of the paper discusses the experimentation carried out in order to validate the proposed. The results obtained after applying the proposed approach are also discussed in this section. In order to carry out the experimentation, two standard Java projects are downloaded from popular open source code repository GitHub. The considered software systems are related to the different domain and are of different sizes. Table-1 gives a description of the considered software system for carrying out the experimentation.

Table-1: Software System under Study.

S.NO.	SYSTEM NAME	VERSION	SIZE (KLOC)	# CLASSES
1	Junit	4.5	3.5	22
2	HospitalAutomationWithJavaEE	1.0	5.9	53

In order to carry out the experiment, the software systems are studied manually and FUP data is extracted. Initially, each class is considered as an individual module and based upon the calculated cohesion value the modules are grouped to have new modules in a software system. Again, the cohesion value is calculated and clustering algorithm's steps are followed to regroup modules. These two processes are iteratively repeated until there is no significant improvement in the overall cohesion of the software system. To evaluate the software system's quality, TurboMQ [19] measure criteria is used. Table-2 shows the TurboMQ values before and after applying our proposed approach. The results obtained justify our proposed approach due to significant improvement in TurboMQ score.

Table- 2: Results for different Software Systems.

S.NO.	SYSTEM NAME	TurboMQ VALUES	
		BEFORE	AFTER
1	Junit	0.456	0.612
2	HospitalAutomation WithJavaEE	0.354	0.471

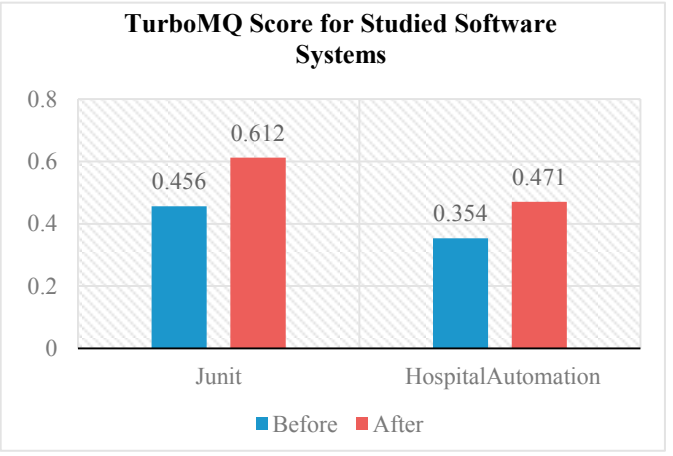

Figure-4: TurboMQ Score for Studied Software Systems.

Figure-4 displays the results obtained after experimentation in bar-chart form. From the chart, it is clear that the proposed new schemes is capable to improve overall cohesion of an software system significantly. In case of *Junit*, the overall cohesion of the software, calculated as the average of the cohesion values of individual modules of the corresponding software systems, is increased by around 34%. Similar is the case of *HospitalAutomation* in which the overall improvement in cohesion is around 33%.

5. Conclusion And Future Scope

This paper has proposed a new approach to measure cohesion of an object-oriented software at module level, where module indicates a single class initially and set of classes in subsequent sections. The new cohesion metric is proposed to be used with a new clustering methodology, which clusters multiple classes in a single group based on the proposed cohesion metric. It is based on the use of FUP computation at module level by analyzing the usage pattern used by different methods of a given software module. Based on FUP computed, cohesion score of each module is calculated and this score is used for clustering modules into more cohesive ones. The proposed approach improves the cohesion of the object-oriented software iteratively and repeatedly calculates the cohesion score and uses it to perform clustering until there is no significant improvement in overall cohesion of the software system.

The future work of this research paper can be extended in many ways. First of all, the proposed cohesion metric can be used as a fitness function in search-based clustering algorithms. Another possibility is to carry out an empirical study to compare its performance with other cohesion based metric in literature in order to find out its significance in research. Further, based on the obtained results, an automated tool can also be proposed.

References

[1] 1. L. Badri and M. Badri. (2004) “A Proposal of a new class cohesion criterion: an empirical study.”Journal of Object Technology, **3 (4).**

[2] J. Bansiya. (2002) “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. Bieman and B. Kang.(1995) “Cohesion and reuse in an object-oriented system.” Proceedings of the 1995 Symposium on Software Reusability, Seattle, Washington, United States,259–262.

[5] C. Bonja and E. Kidanmariam.(2006) “Metrics for class cohesion and similarity between methods.” In Proceedings of the 44th annual Southeast regional conference, Florida, ACM, 91–95.

[6] L. Briand, K. E. Emam, and S. Morasca.(1996) “On the application of measurement theory in software engineering.” Empirical Software Engineering, 1:61–88.

[7] L. C. Briand, J. W. Daly and J. Wust.(1997) “A Unified Framework for Cohesion Measurement in Object-Oriented Systems.” Software Metrics Symposium, Proceedings, Fourth International, 43-53.

[8] S. R. Chidamber and C.F. Kemerer, C.F.(1994) “A metrics suite for object oriented design.” IEEE Transactions on Software Engineering, 20, 476–493.

[9] N. E. Fenton and S. L. Pfleeger.(1996) “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.

[17] A. Lakhotia.(1993) “Rule–based approach to computing module cohesion.” In Proceedings of the 15th Conference on Software Engineering (ICSE-15), 34–44.

[18] W. Li, and S. M. Henry, S.M.(1993) “Maintenance metrics for the object oriented paradigm.” Proceedings of 1st International Software Metrics Symposium, Baltimore, 52–60.

[19] B. S. Mitchell, M. Traverso and S. Mancoridis.(2001) "An architecture for distributing the computation of software clustering algorithms", In Proc. IEEE/IFIP Conf. on Software Architecture.

[20] W. Scacchi.(2001) “Process models in software engineering.” Encyclopedia of Software Engineering, 993–1005.

[21] W. Stevens, G. Myers, and L. Constantine.(1974) “Structured Design.” IBM Systems J., **12(2)**.

[22] Yourdon, E and Constantine, L.L.(1979) “Structured design.” Prentice Hall, Englewood Cliffs, NJ.

[23] Amarjeet, Jitender Kumar Chhabra.(2017) “Improving Modular Structure of Software System using structural and lexical dependency”, Information & Software Technology (Elsevier, SCI), 82, 96-120.

[24] Amarjeet, Jitender Kumar Chhabra.(2015) “Preserving Core Components of Object-oriented Packages while Maintaining Structural Quality”, Procedia Computer Science (Elsevier), 46, 833-840.