International Journal of Computer Science Trends and Technology (IJCST) – Volume 2 Issue 5, Sep-Oct 2014

RESEARCH ARTICLE OPEN ACCESS

A Review of Object-Oriented Coupling and Cohesion Metrics

Sushma Yadav[1], Dr. Sunil Sikka[2], Uttpal Shrivastava[3]
M.Tech Research Scholar[1], Assistant Professor[2], Senior Lecturer[3]
Department of Computer Science & Engineering
Amity University
Haryana - India

ABSTRACT

Software metrics are essential to improve the quality of software during the development process. Coupling and cohesion measures are used in various activities such as impact analysis, assessing the fault proneness of classes, fault prediction, re-modularization, identifying of software component, design patterns, assessing software quality etc. Low coupling and high cohesion are better for good software quality. Coupling and cohesion metrics can be applied at the early phase of the software development process. This paper reviews various coupling and cohesion metrics for object-oriented software.

Keywords: - Software Metrics, Coupling Metric, Cohesion Metric**.**

I. INTRODUCTION

Success of any software system depends upon the properly used metrics. Coupling and cohesion measures can be defined as the indication of relationships among elements of the source code. Classes, methods and attributes in the object-oriented software systems can be consider as elements [1].

"Coupling is the measure of the strength of association established by a connection from one module to another". [2] If module A and B are strongly interconnected than modules are highly coupled while loosely coupled module have weak interconnections. If module A and B are independent than there is no interconnection [3]. Modification becomes simple, if modules are loosely coupled. Reusability and maintainability are the advantages of low coupling. [4]

Cohesion refers to the degree to which the elements of a module belongs together [5]. "Cohesion can be measured by inspecting the connection between all pairs of its processing elements"[2]. Degree of cohesion of software module is high when element of that module exhibit high degree of semantic relatedness [6]. Reusability, maintainability and extensibility are advantages of high cohesion [4].

This paper provides a review of various object-oriented coupling and cohesion metrics. Static coupling metrics of Chidamber and Kemerer[11][12], Li and Henry[14], Abreu et al[15][16], Martin[17],and Bansiya et al.[18] are discussed. Dynamic coupling metrics of Yacoub[19], Arisholm et al.[20], Hassoun et al.[22][23], Beszedes et al.[24] and Singh and Singh[25][26] are discussed. Static cohesion metrics of Chidamber and Kemerer[27][28], Briand et al.[29], Bieman and Kang [30], Li and Henry[31], Hitz and Montazeri[32], Henderson-Sellers[33], Xu and Zhou[34], Yang Metric[35], Badri et al[36], Bonja et al[37], Fernandez et al[38]., Jehad Al Dallal[39], Michael Bowman et al.[40] and Jehad et al.[41], are discussed. Dynamic cohesion metrics of Gupta et al.[43] and Mitchell [44][45]are discussed.

Rest of the paper is organised into two sections. Section 2 presents the review of various coupling and cohesion metrics cohesion. Section 3 concludes the paper and provides the scope of future work.

II. REVIEW OF COUPLING AND COHESION METRICS

Various coupling and cohesion metrics have been proposed by many researchers in the literature. Coupling and cohesion metric can be classified into two categories - static and dynamic. The static software metrics are obtained from static analysis, whereas the dynamic software metrics are computed on the basis of data collected during run time execution of the software [7]. Static metrics are easy to calculate while dynamic metrics are tough to calculate. Static metrics are useful when we need result from small programs. Ability of static metric is to quantify various aspects of the design complexity or source code of a software system, make them useful in software engineering [8]. Static metrics are inefficient to deal with object-oriented features such as polymorphism,

dynamic binding and the presence of unused code [7]. Inefficiency of static metrics to deal with object-oriented features raised the need of dynamic metrics [7].

1.1 STATIC COUPLING METRICS

Non object-oriented programming can be measured using static coupling metrics [9]. Some static coupling metrics are simple and some are more complex [9].

2.1.1 Chidamber and Kemerer (CK) Metrics – CK metrics are most discussed metrics in the world of software engineering. Among six CK metrics, two metrics Coupling Between Object Classes (CBO) and Response For a Class (RFC) are coupling metrics.

2.1.1.1 CBO: CBO [10] for a class is the count of the number of other classes to which it is coupled. A class is coupled with another class if the method declared in one class use method or instance variable defined by another class. CBO includes inheritance based coupling (i.e. coupling between classes related via inheritance). For good software quality, high CBO is undesirable. Coupling metric is good predictors for the maintainability of components in object oriented systems [11].

2.1.1.2 RFC: RFC [12] can be defined as the set of methods that can potentially be executed in response to a message received by an object of that class.

RFC = |RS|

Where, RS = the response set for the class.

The response set for the class can be defined as:

RS = { M } ∪ { Ri }for all i

Where, {Ri} is set of methods called by method i and {M} is set of all methods in the class. Complexity of the class increases as RFC increases [13].

2.1.2 Li and Henry [14] proposed various metrics, which includes two coupling metrics.

2.1.2.1 Message Passing Coupling (MPC): MPC is the total number of function and procedure calls made to external unit. Higher MPC leads to higher complexity and system is difficult to maintain [13].

2.1.2.2 Data Abstraction Coupling (DAC): DAC counts the total number of instances of other classes within a given class.

2.1.3 Abreu et al. [15][16] proposed Coupling Factor (COF) metric. COF used for the design quality evaluation of object-oriented software systems.

2.1.3.1 COF [46]: COF is defined as the ratio of the maximum number of possible couplings in the system to the actual number of couplings which are not related to inheritance[e]. COF also counts the number of inter-class communication.

$$\text{COF} = \frac{\sum_{i=1}^{tc}[\sum_{i=1}^{tc} client(CiCj)]}{tc^2 - tc}$$

Where client(C_iC_j) = $\begin{cases} 1 & \text{iff } C_i => C_j \wedge C_i \neq C_j \\ 0 & \end{cases}$

tc = Total number of classes in the system under consideration.

C_i = client class

C_j = server class

$C_i => C_j$ represents the relationship between client and server class.

Client Class contains at least one non-inheritance reference to a feature of the server class.

2.1.4 Martin Metrics [17] used to measure the quality of an object-oriented design in terms of the interdependence between the subsystems of that design.

2.1.4.1Afferent Coupling (Ca): Ca also known as incoming coupling. Ca is the number of packages which is depending on classes within the package. Ca points out package's responsibility.

2.1.4.2Efferent Coupling (Ce) – Ce also known as outgoing coupling. Ce is the number of packages in which the package depends upon. Ce points out package's independence.

2.1.5 Bansiya et al. QMOOD [18] proposed Direct Class Coupling Metric (DCC).

2.1.5.1 DCC: It is a count of different number of classes that a class is directly related to.

1.2 DYNAMIC COUPLING METRICS[8]

Actual coupling taking place between a pair of objects or classes at runtime can be measure using dynamic coupling metrics. These metrics are measured at object level and can be collect to class or system level.

2.2.1 Yacoub et al.[19] proposed two object level dynamic coupling metrics- Export Object Coupling(EOC) and Import Object Coupling (IOC).

2.2.1.1 EOCx(oi; oj): EOCx(oi; oj) for an object *oi* with respect to an object *oj*, is defined as the percentage of the number of messages sent from *oi* to *oj* with respect to the total number of messages exchanged during the execution of scenario *x*. A class's object with higher EOC to another specific object is more critical to changes due to maintenance [11].

2.2.1.2 IOCx(oi; oj): IOCx(oi; oj) for an object *oi* with respect to an object *oj* , is the percentage of the number of messages received by object *oi* that were sent by object *oj* with respect to the total number of messages exchanged during the execution of a scenario *x*. A class's object with higher IOC to another specific object is more likely to import changes due to maintenance in the class from which that specific object is instantiated [11].

2.2.3 Arisholm et al.[20][8] extended the concept of Yacoub[19]. They proposed number of dynamic coupling metrics. Some of them are defined at class level and some are defined at object level. Name of each dynamic metric starts with either IC or EC based on the direction of the method calls. 'IC' stands for import coupling and 'EC' stands for export coupling. Counting of messages which are sent from an object or class is defined by import coupling. Counting of messages which are received by an object or class is defined by export coupling. The third letter indicate the mapping level 'O' for object and 'C' for class. The last letters D, M and C denotes the strength of coupling. Strength of coupling measures the amount of association between the two objects. D denotes dynamic messages. M denotes distinct method invocation, and C denotes distinct classes.

C = counts the number of distinct classes that a method in a given class or object uses or is used by.

M = counts the number of distinct methods invoked by each method in each class or object.

D = counts the total number of dynamic messages sent or received from one class/object to or from other classes or objects. All import and export coupling metrics are defined as following:-

2.2.3.1 IC_OD: This metric count the total number of messages sent from one object to other objects.

2.2.3.2 IC _OM: This metric count the number of distinct methods invoked from one object to other objects.

2.2.3.3 IC_OC: This metric counts the number of distinct server classes used by the methods of the given object.

2.2.3.4 IC_CD: This metric counts the total number of messages sent by all methods in all objects of a class.

2.2.3.5 IC_CM: This metric counts the number of distinct methods invoked by all methods in all the objects of a class.

2.2.3.6 IC_CC: This metric counts the number of distinct server classes used by all methods of all objects of a class.

2.2.3.7 EC_OD: This metric counts the total number of messages received by one object from other objects.

2.2.3.8 EC_OM: This metric counts the number of distinct methods received by an object.

2.2.3.9 EC_OC: This metric counts the number of distinct client classes that in a given object are being used.

2.2.3.10 EC_CD: This metric counts the total number of messages received by all methods of all objects of a class.

2.2.3.11EC_CM: This metric counts the number of distinct methods received by all methods of all objects of a class.

2.2.3.12 EC_CC: This metric counts the number of distinct client classes that in all objects of a given class are being used.

2.2.4 Hassoun et al. proposed Dynamic Coupling Metric (DCM).

2.2.4.1 DCM: DCM measures the degree of interaction between object A and object B from a dynamic rather than static context [21]. "DCM defines the coupling between object A and object B as varying in time"[22]. Runtime complexity of a system may be predicted using this metric. DCM measures the coupling of specific objects and or the whole system at runtime [23].

2.2.5 Beszedes et al. [24] proposed Dynamic Function Coupling (DFC).

2.2.5.1 DFC: It is defined as the minimal level of indirection among all possible occurrences of the two functions in the traces.

2.2.6 Singh et al.[25][26] proposed metrics are used to evaluate the quality of object oriented software system.

2.2.6.1 Dynamic Afferent Coupling (DCa): It defines the percentage of number of classes accessing the methods of a class at runtime to the total number of classes.

2.2.6.2 Dynamic Key Server Class (DKSC): It defines the percentage of number of calls sent to a class at runtime to the total number of static calls sent to all the classes.

2.2.6.3 Dynamic Key Client Class (DKCC): It defines the percentage of number of calls sent by a class at

runtime to the total number of static calls sent by all the classes.

2.2.6.4 Dynamic Key Class (DKC): It defines the percentage of sum of calls sent out from the class and calls received by the class at runtime turned on the total number of static calls sent and received by all the classes.

2.2.6.5 Percentage Active Classes (PAC): It defines the percentage of number of classes sending or receiving at least one method calls from/to another class at runtime to the total number of classes.

1.3 STATIC COHESION METRIC

Static cohesion metrics are also used to measure non object-oriented programming. A large number of static cohesion metrics are proposed for measuring cohesion.

2.3.1 Chidamber and Kemerer proposed Lack of Cohesion in Methods (LCOM1)[27] and LCOM2[28].

2.3.1.1 *LCOM1* [27]: LCOM1counts the number of pairs of methods that do not share attributes. In LCOM1 if a class and substance are not clearly defined than software becomes harder to maintain [28].

2.3.1.2 LCOM2: LCOM2 [28] is defined as LCOM1 minus number of pairs of methods that use common attribute.

LCOM2= P – Q if P – Q ≥ 0

0 otherwise.

Where, P=number of pairs of methods that do not share attributes.

Q=number of pair of methods that share attributes.

2.3.2 Briand et al. [29] proposed high level design metrics for object-oriented system. These metrics are used to identify error prone software parts. All metrics discussed in this section satisfies the properties of normalization, monotonicity and cohesive modules.

2.3.2.1 Ratio of Cohesive Interactions (*RCI*): cohesive interactions only happen within modules, but not everywhere in modules. The Ratio of Cohesive Interactions for software part (sp) is -

$$RCI(sp) = \frac{|CI(sp)|}{|M(sp)|}$$

Where, CI(sp) = The set of cohesion interaction in a module M.

M(sp) = The maximal set of cohesive interaction of the software part(sp) i.e. the set of includes all of sp's possible cohesion interaction.

2.3.2.2 Neutral Ratio of Cohesive Interactions (*NRCI*): NRCI(sp) is undefined if and only if no information is available on cohesive interactions.

Unknown CIs are not taken into account.

$$NRCI(sp) = \frac{|K(sp)|}{|M(sp)|-|U(sp)|}$$

Where, K(sp) = The set of known interactions of a software part (sp).

M(sp) = The maximal set of cohesive interaction of the software part(sp).

U(sp) = The set of unknown interaction.

2.3.2.3 Pessimistic Ratio of Cohesive Interaction (*PRCI*): This metric is equal to RCI(sp). PRCI considers undefined CIs, if they were known not to be actual interactions.

$$PRCI(sp) = \frac{|K(sp)|}{|M(sp)|}$$

2.3.2.4 Optimistic Ratio of Cohesive Interactions (ORCI): If unknown CIs are known to be actual interaction than they may be consider.

$$ORCI(sp) = \frac{|K(sp)| + |U(sp)|}{|M(sp)|}$$

2.3.3 Bieman and Kang [30] proposed Tight Class Cohesion (TCC) and Loose Class Cohesion (LCC).

2.3.3.1 TCC: The measure TCC is defined as the percentage of pairs of public methods of the class with common attribute usage.

2.3.3.2 LCC: - LCC defines the relative number of directly or indirectly connected pairs of methods, wherein two methods are transitively connected if they are directly or indirectly connected to an attribute.

2.3.4 Li and Henry [31] proposed LCOM3.

2.3.4.1 LCOM3: It is the number of connected components in graph. To compute LCOM3 each method represented as a node and the use of at least one attribute as an edge.

2.3.5 Hitz and Montazeri [32] proposed LCOM4.

2.3.5.1 LCOM4: It is similar to LCOM3, To represent method invocation additional edges are used.

2.3.6 Henderson-Sellers [33] proposed a cohesion metric *LCOM*5.

2.3.6.1 LCOM5: LCOM5 = $(mh - a)(mh - h)$

Where, h=number of attributes

m=number of methods

a=summation of the number of definite attributes that are accessed by each method in a class.

2.3.7 Xu and Zhou [34] gave Improved Cohesion Based on Member Connectivity (ICBMC):

2.3.7.1 ICBMC:

$$\text{ICBMC (G)} = \text{Fc(G)} \times \text{Fs(G)}$$

$$\text{Where , Fc(G)} = \frac{|E(G)|}{|M(G)|}$$

E (G) = number of edges in the cut set of G,

M (G) = number of non-special methods represented in graph G multiplied by the number of attributes.

$$\text{Fs(G)} = [\sum_{ni} = \text{ICBMC}(Gi)] \,/2$$

2.3.8 Yang [35] gave their OLn metric. It satisfies all class cohesion properties [28].

2.3.8.1 OLn: This metric can be defined as the common strength of attributes; the strength of the attribute can be defined as the common strength of the methods that approach that attribute. Where n are the number of iteration which are used to calculate OL.

2.3.9 Badri et al.[36] proposed connectivity based metrics Degree of Direct Cohesion (DC_D) and Degree of Indirect Cohesion(DCi).

2.3.9.1 DC_D: It defines the relative number of straightly connected pairs of methods. It satisfies the condition of TCC metric.

2.3.9.2 DCi: It defines the relative number of straightly or transitively connected pairs of methods. It satisfies the condition of LCC metric.

2.3.10 Bonja et al.[37] proposed Class Cohesion (CC) metric.

CC: CC is the ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods. The similarity between methods m and n is:

$$\text{Similarity (M, n)} = |\text{Mm} \cap \text{Mn} / \text{Mm} \cup \text{Mn}|$$

where, Mm, Mn = sets of attributes that are referenced by methods m and n, respectively

2.3.11 Fernandez et al. [38] proposed Class Cohesion metric (SCOM).

2.3.11.1 SCOM: This metric is the ratio of the summation of the similarities between all pairs of methods to the total number of pairs of methods. The similarity between methods m and n is defined as:

$$\text{Similarity (m,n)} = |\text{Mm} \cap \text{Mn}|.|\text{MmUMn}|$$

2.3.12 Jehad Al Dallal Metrics [39] proposed Path Connectivity Class Cohesion (PCCC)

2.3.12.1 PCCC:

$$\text{PCCC} = \begin{cases} \text{PCCC} = 0 \text{ If l} = 0 \text{ and n} > m \\ 1 \; if \; m > 0 \; and \; n > m \\ \text{NSP(Gc)/NSP(FGc) oterwise} \end{cases}$$

Where, m = no. of attributes

n = no. of methods

NSP = number of simple paths in graph Gc

FGc = corresponding fully connected graph

2.3.13 Michael Bowman et al. [40] proposed a procedure based on a multi-objective genetic algorithm (MOGA) which uses class coupling and cohesion measurement for describing fitness functions. Their work has some similarity with refactoring. They did most of the work on source code refactoring, although it is concluded that refactoring obtains higher levels of abstraction, such as refactoring of UML models

2.3.14 Jehad et al. [41] proposed Method-Method through Attributes Cohesion (MMAC).

MMAC: The MMAC can be defined as the average cohesion of all pairs of methods

$$\text{MMAC(C)} = \begin{cases} 0 \text{ if } m = 0 and \; n = 0 \\ 1 \text{ if n} = 1 \\ \text{where } \frac{\sum_{i=1}^{m} xi(xi-1)}{mn(n-m)} \end{cases}$$

m = no. of attributes

n = no. of methods

xi = number of methods that have a or a return type j.

1.4 DYNAMIC COHESION METRICS

As above mentioned in this paper that dynamic cohesion metrics performs better than static cohesion metrics.

Dynamic metrics supports graphical user interface(GUI) also.

2.4.1 Gupta et al. Metrics [42, 43] Bieman and Ott [29] proposed the Strong Functional Cohesion (SFC) metric and Weak Functional Cohesion(WFC) metric. The work of Bieman and Ott is redefined by Gupta et al.. Gupta et al starts the dynamic cohesion measurement using program execution based approach on the basis of dynamic slicing.

2.4.2 Mitchell et al. [44][45] provide a new technique for collecting dynamic trace information from Java GUI programs and a number of simple runtime metrics are proposed.

2.4.2.1 The exPubMet.Ob: This metric gives an approximation of level of coupling present in a GUI program.

2.4.2.2 priMet.ob: This metric present that simple programs give a greater proportion of their method access to the internal working of their classes than the GUI program.

2.4.2.3 meth.ob: This metric used to measure the program size.

$$\text{meth.ob} = \frac{\text{Total number of methos called}}{\text{Total number of object created}}$$

2.4.2.4 meth.inst: GUI metric gives an estimation of the memory use of the methods.

$$\text{meth.inst} = \frac{\text{Number of methods called}}{\text{kilo byte code instruction execute}}$$

III. OBSERVATION AND FUTURE WORKS

This paper reviews various coupling and cohesion metrics. Metrics have been discussed with their definition and formulas. Only some class level and GUI based metrics have been discussed in this paper. Other metrics can be discussed in next paper.

REFERENCES

[1] Shweta Sharma, Dr. S. Srinivasan, "A review of Coupling and Cohesion metrics in Object Oriented Environment", International Journal of Computer Science &Engineering Technology, vol 4, no.8,2013.

[2]W.P. Stevens, G.J. Myers, and L. L. Constantine. Structured design., IBM Systems Journal, vol.13, no.2, pp.115-139, 1974.

[3] Pankaj Jalote, "An Integrated Approach to Software Engineering," Springer publication, Third edition , North and South America, Europe, Australia, parts of Asia/Africa, ISBN: 0-387-20881-X.

[4]Pressman,Roger, "Software Engineering - A Practitioner's Approach," Fourth Edition, 1982, ISBN 0-07-052182-4.

[5]Edward Yourdan and Larry L. Constantine, "Structural – Fundamentals of Discipline of Computer Program and Design.

[6]O.Ormandjieva, M.Kassab, C.Constantinides, "Measurement of Coupling and Cohesion in OO Analysis model: based on crosscutting concerns," IWSM, pp. 3-13, 2005.

[7]Varun Gupta, Phd Thesis, "Object-Oriented Static and Dynamic Software Metrics for Design and Complexity," National Institute of Technology, Kurukshetra-136119 India, June 2010.

[8] Jitender Kumar Chhabra et al., "A Survey of Dynamic Software Metrics," journal of computer science and technology, vol.25,no.5, Sept. 2010.

[9] Vasudha et al., "Static and Dynamic Coupling and Cohesion Measures in Object-Oriented Programming," International Journal of Engineering and Research, vol.2,issue 7,pp. 472-477

[10] S.R. Chidamber and C.F. Kemerer., "Towards a metrics suite for object-oriented design",In Object Oriented Programming Systems Languages and Applications, Phoenix, Arizona, USA, pp.197-211, November 1991.

[11] Melis Dagpiner and Jens H.Jahnke., "Predicting maintainability with onject-oriented metrics and empirical comparison", Proceeding of the 10th working conference on reverse engineering, 2003.

[12] S.R. Chidamber and C.F. Kemerer., "A metrics suite for object-oriented design", IEEE Transactions on Software Engineering, vol.20, no.6, pp.467-493, June1994.

[13] Jonas Lindell Mats Hägglund , "Maintainability metrics for object oriented systems".

[14] W. Li and S. Henry., "Object-oriented metrics that predict maintainability", The Journal of Systems and Software, vol.23, no.2, pp.111-122.

[15] F. Abreu, M. Goulo, and R. Esteves., " Toward the

design quality evaluation of object-oriented software systems," In Fifth International Conference on Software Quality, Austin, Texas, USA, pp.44-57,Oct 1995.

[16] Abreu and Melo, F. Brito e Abreu, Walcelio Melo., "Evaluating the impact of Object-Oriented Design on Software Quality," Proceedings of 3rd International Software Metrics Symp., Berlin, 1996.

[17]R. Martin. OO design quality metrics, "An analysis of dependencies", In Proceedings Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, 1994.

[18]Sonia Chawla, "Review of MOOD and QMOOD metric sets", International journal of advanced research in computer science and software engineering, vol.3, issue 3, March 2013.

[19]S.M. Yacoub, H.H. Ammar, and T. Robinson., "Dynamic metrics for object oriented designs", In Software Metrics Symposium, Boca Raton,Florida, USA, pp. 50-6, Nov 1999.

[20]E. Arisholm, L.C. Briand, and A. Foyen., "Dynamic coupling measures for object- oriented software", IEEE Transactions on Software,vol. 30,no.8,pp. 491-506,2004.

[21]S. M. Yacoub, T. Robinson, and H. H. Ammar, "Dynamic metrics for object oriented design," in Proc. International Symposium on Software Metrics, pp. 50–58, 1999.

[22] Y. Hassoun, R. Johnson, and S. Counsell,"A dynamic runtime coupling metric for meta-level architectures,"in Software Maintenance and Reengineering, CSMR Proceedings. Eighth European Conference on. IEEE, pp. 339–346, 2004.

[23] Y. Hassoun, R. Johnson, S. Counsell., "A Dynamic Runtime Coupling Metric for Meta Level Architectures", In Proceedings of Eighth Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '04), pp. 339, 2004.

[24]A. BeszedesT. Gergely S. Farago., "Dynamic Function Coupling Metric and Its Use in Software Evolution",In Proceedings of the Eleventh European Conference on Software Maintenance and Reengineering(CSMR),IEEE Computer Society, Washington DC, 2007,pp. 103-112

[25]P. Singh H. Singh., "Class-level Dynamic Coupling Metrics for Static and Dynamic Analysis of Object-Oriented Systems", International Journal of Information and Telecommunication Technology, vol.1,no.1, pp.16-28, 2010.

[26]Rani Geetika, Paramvir Singh, "Dynamic Coupling Metrics for Object Oriented Software Systems- A Survey", ACM SIGSOFT Software Engineering Notes, Vol. 39, March 2014.

[27]Chidamber, S.R., Kemerer, C.F., "Towards a metrics suite for object-oriented design." Object-Oriented Programming Systems, Languages and Applications (OOPSLA), vol. 26, pp.197–211, 1991.

[28] Chidamber, S.R., Kemerer, C.F., "A metrics suite for object oriented design.", IEEE Transactions on Software Engineering, vol. 20, pp. 476–493, 1994.

[29]L.C. Briand, S. Morasca, and V. Basili., " Defining and validating high-level design metrics",Technical Report CS-TR 3301, Department of Computer Science, University of Maryland, College Park, MD 20742, USA, 1994.

[30] Bieman, J., Kang, B., "Cohesion and reuse in an object-oriented system.", In: Proceedings of the 1995 Symposium on Software Reusability, Seattle, Washington, United States, pp.259–262, 1995.

[31]Li, W., Henry, S.M., "Maintenance metrics for the object oriented paradigm." In: Proceedings of 1st International Software Metrics Symposium, Baltimore, pp.52–60, 1993.

[32] Hitz, M., Montazeri, B., "Measuring coupling and cohesion in object oriented systems.", Proceedings of the International Symposium on Applied Corporate Computing, pp. 25–27, 1995.

[33] Henderson-Sellers, B., "Object-Oriented Metrics Measures of Complexity", Prentice-Hall, 1996.

[34] xu, b., zhou, y., "comments on 'a cohesion measure for object-oriented classes", software: practice & experience, vol. 31, pp.1381–1388, 2001.

[35] Yang, x., "research on class cohesion measures.", m.s. thesis. department of computer science and engineering, southeast university, 2002.

[36] Badri m. and badri l., "a proposal of a new class cohesion criterion: an empirical study", journal of object technology, vol. 3, pp.145-159, 2004.

[37] Bonja, C. et al.., "Metrics for class cohesion and similarity between methods.", In: Proceedings of the 44th Annual ACM Southeast Regional Conference, Melbourne, pp. 91–95, 2006.

[38]Fernández, l., pena, r., "a sensitive metric of class cohesion", international journal of information theories and applications, vol. 13, pp.82–91, 2006.

[39]Amandeep et al., "Class Cohesion Metrics in Object Oriented System",International Journal of Software and Web Sciences,vol 3,no-2, December, pp. 78-82,2013.

[40]Bowman m., briand l.c. and labiche y., ieee transactions on software engineering, pp.1-50, 2010.

[41] Jehad Al Dallal, "A Design Based Cohesion Metric for Object-Oriented Classes", World Academy of

Science, Engineering and Technology, vol.1, 21oct.2007.

[42] Bieman j m, ott l m., "measuring functional cohesion", ieee transactions on software engineering, vol.20(8), pp. 644-657, 1994.

[43]Gupta n, rao p. , "program execution based module cohesion measurement", in proc. the 16th international conference on automated on software engineering , San diego, USA, pp.144-153, , Nov. 2001.

[44] [34] Mitchell A, Power J F. Run-time cohesion metrics for the analysis of Java programs. Technical Report, SeriesNo. NUIM-CS-TR-2003-08, National University of Ireland, Maynooth, Co. Kildare, Ireland, 2003.

[45] Mitchell A, Power J F, Run-time cohesion metrics: An empirical investigation. In Proc. the International Conference on Software Engineering Research and Practice, Las Vegas,USA, Jun. 21-24, 2004, pp.532-537.

[46] Prof. Jubair JAL – JAFFER et al., "Metric for Object-Oriented design (MOOD) to assess java programs.