A review of Coupling and Cohesion metrics in Object Oriented Environment

SHWETA SHARMA *
Phd.Research Scholar , MewarUniversity,
Rajsthan, India
bhardwaj.shweta28@gmail.com

Dr S. SRINIVASAN*
Head of Dept. PDM College
Haryana,India
dss_dce@yahoo.com

Abstract - Software metrics are used to check and evaluate various aspects of the complexity of a software product. Coupling and Cohesion are considered to be the most important attributes. The increasing need for software quality measurements has led to extensive research into software metrics and the development of software metric tools. Many Software Metrics have been proposed for object oriented paradigms to measure various attributes like complexity, cohesion, software quality, and productivity. As object oriented analysis and design appears to be at the forefront of software engineering technologies, many different object-oriented coupling and cohesion metrics have been developed. To maintain high quality software, developers' choice is always low coupled and highly cohesive design. The aim of the paper is to reinforce the existing coupling and cohesion metrics specifically used in object oriented environment, to analyze their significance in software development, expose their limitations and some suggestions for further investigations.

Keywords — Dependencies, Static and dynamic Coupling and Cohesion.

I. INTRODUCTION

There are mainly seven different levels which we can use to find the characteristics of complexity of software product by establishing the correlation and interdependence between them.

These levels are as follows:

Control Structure

Module Coupling

Algorithm

Code

Nesting

Module Cohesion

Data Structure

Among all of these, "Coupling" and "Cohesion" are considered to be the most important attributes. Coupling and cohesion are the attributes which measure the degree or the strength of interaction and relationships among elements of the source code, for example classes, methods, and attributes in object-oriented (OO) software systems. One of the main objectives behind Object Oriented analysis and design is to implement a software system where classes have high cohesion and low coupling amongst them.

Figure 1.1 illustrates the software quality model which depicts the relationship between these measures.

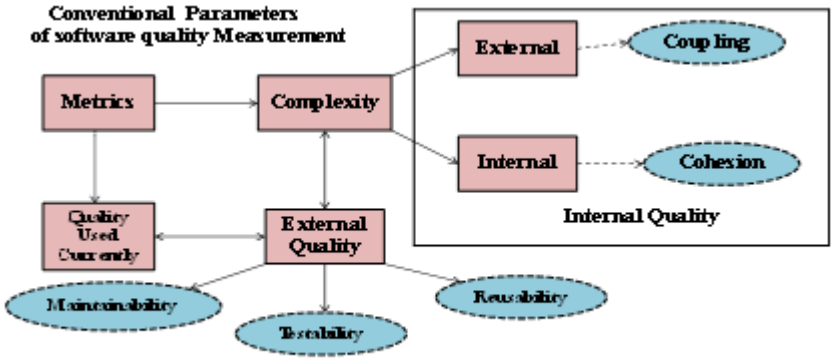

Fig 1.1 : External and Internal quality attributes of a Software Product

Now-a-days, the Object-Oriented design and development is becoming the first choice of a developer in today's software development environment. Object-oriented development needs not only a different approach to design and implementation but it requires a different approach to software metrics.

Coupling in existing literature

Coupling or dependency is the degree to which each program module relies on each one of the other modules. Stevens et al. [1] first introduced coupling in the context of structured development techniques. According to them "coupling is the measure of the strength of association established by a connection from one module to another". In their opinion, the complexity of the software product will be dependent upon the interconnection and interdependence between the modules.

Fig 1.2 shows variety of coupling and the interdependence among modules.

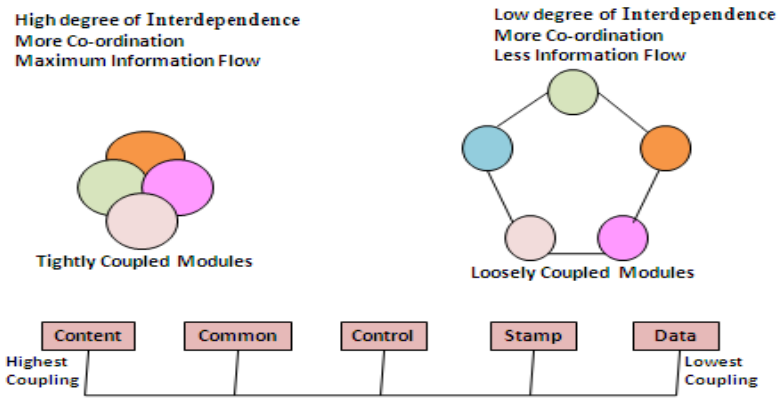

Fig 1.2 : Interdependence among modules according to different types of coupling

Existing Forms of Coupling

As coupling is the degree of interdependence among the modules so that degree can be high as well as low depending on their bonding level. The following is the set of different types of coupling in the order of the precedence from highest degree to the lowest one:

1. Content Coupling (high)
2. Common Coupling
3. External Coupling
4. Control Coupling
5. Stamp Coupling
6. Data Coupling
7. Message Coupling (low)

Cohesion in existing literature

Cohesion is a measure of degree of strength in terms of each piece of functionality expressed by the source code of a software module. The cohesion of a module is the extent to which its individual components are needed to perform the same task [2]. Cohesion was first introduced within the context of module design by Stevens et al. [1]. In their definition, the cohesion of a module is measured by inspecting the connection between all pairs of its processing elements.

Existing Forms of Cohesion

Cohesion is highly desired by the developers because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability whereas low cohesion is associated with undesirable traits such as being difficult to maintain, difficult to test, difficult to reuse, and even difficult to understand. Therefore the following are the different types of cohesion in the precedence of order from worst to the best:

1. Coincidental Cohesion (worst
2. Logical Cohesion
3. Temporal Cohesion
4. Procedural Cohesion
5. Communicational Cohesion
6. Sequential Cohesion:
7. Functional cohesion (best)

Here figure 1.3 reveals how degree of cohesion varies from its one form to the other.

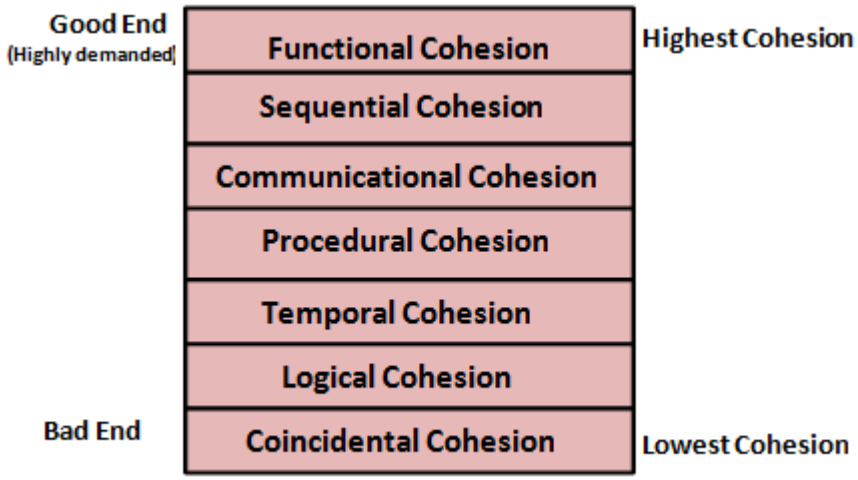

Fig1.3: Degree of Cohesion from low end to the high end

II. COUPLING MEASURES

To determine the complexity, it is very important and useful to measure the coupling between modules. The higher the inter object coupling, the more scrupulous the testing needs to be. There are several matrices using this concept . Number of children metric defines number of sub-classes subordinated to a class in the class hierarchy.

Coupling between Number of Objects is that two classes are said to be coupled if the methods of one class use the methods or attributes of the other class. Number of Dependencies IN is defined as the number of classes that depend on a given class [3]. Number of Dependencies OUT metric is defined as the number of classes on which a given class depends. Number of Association metric was suggested by Brian in which he stated that the number of association per class metric is the total number of associations a class has with other classes or with itself. Direct Dependency is direct association between services. This kind of dependency may exist between services explicitly when a service itself calls other services or a service is called by other services [6]. Indirect Dependency between services may occur in two cases. In the first case, when an indirect or transitive connection or association between the services is present. In the second case when the services share global data.

1.1 Static Coupling Metrics

There exists a large variety of measurements for coupling. A comprehensive review of existing measures performed by Briand et al. [4] found that more than thirty different measures of object-oriented coupling exist. The most prevalent ones are explained in the following subsections:

1.1.1 Chidamber and Kemerer suite of Metrics

Chidamber and Kemerer propose and validate a software metrics for object-oriented systems for the following basic purposes:

(a) To measure the unique aspects of Object Oriented approach.

(b) To measure the complexity of the design.

(c) To improve the development of the software.

The most accepted and commonly used coupling metrics amongst them are:

- Coupling Between Objects (CBO)
- Response for class (RFC)

Coupling Between Objects (CBO)

Chidamber and Kemerer first define a measure CBO for a class as, a count of the number of non-inheritance related couples with other classes [5]. If the methods of one class use the methods or attributes of the other that implies that the objects of both of the classes are coupled with each other. To improve the modularity of a software the inter coupling between different classes should be kept to a minimum. Beside reusability a high coupling also has a second weakness, a class that is coupled to other classes is susceptible to changes in those classes and as a result it becomes more difficult to maintain and becomes more error-prone. Additionally it is also harder to test a heavily coupled class in isolation. The class becomes so ambiguous that it is quite difficult to understand it. Therefore the number of dependencies should be kept at a minimum.

They further refined this definition by saying that CBO for a class is a count of the number of other classes to which it is coupled.

Response for class (RFC)

The response set (RS) of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. RFC is used to measure the number of different methods that can be executed when an object of that class receives a message (when a method is invoked for that object). There are some other important metrics discussed in this direction which measure the degree of coupling among different classes and hence are useful to determine the complexity of the software product. Message Passing Metrics (Li and Henry) recognizes a number of metrics that can predict the maintainability of a design [7]. There are two measures, message passing coupling (MPC) and data abstraction coupling (DAC). Message Passing Coupling measures the numbers of messages passing among objects of the class. Data Abstraction Coupling Metric (DAC) measures the number of Abstract Data Types defined in a class. This metric is used to measure the number of instantiations of other classes within the given class. Also the Afferent and Efferent Coupling Metric is given by Martin. Afferent coupling is harder to determine and much more valuable. It measures how many other classes use the current class. Efferent coupling determines how many number of classes the current class references. It is easy to find out via simple inspection: open the class in question and count the references (in fields and parameters) to other classes.

III. COHESION MEASURES

Cohesion can be defined as the intra-modular functional relatedness of a software module. As previously stated, we can categorize cohesion is into seven levels (ranging from low cohesion to high cohesion).

2.1 Static Cohesion Metrics

There are a lot of alternative measures which are being proposed for measuring cohesion. A broad survey on the current state of cohesion measurement is carried out by Briand et al. [8] in object-oriented systems and he provided fifteen separate measurements of cohesion. Following is a review of these measures in the following subsections.

2.1.1 Chidamber and Kemerer

The Lack of Cohesion in Methods (LCOM1) measure was first suggested by Chidamber and Kemerer [5].Given n methods M1, M2, …, Mn contained in a class C1 which also contains a set of instance variables {Ii}. Then for any method Mi we can define the partitioned set of

$P = \{(I_i, I_j) \mid I_i \cap I_j = \varphi\}$ and $Q = \{(I_i, I_j) \mid I_i \cap I_j \neq \varphi\}$

then LCOM = |P| - |Q|, if |P| > |Q| =0 otherwise

LCOM is a count of the number of method pairs whose similarity is zero.

Example: Consider a class C with three methods M_1,M_2 and M_3. Let $\{I_1\} = \{p,q,r,s,t\}$ and $\{I_2\} = \{p,q,t\}$ and $\{I_3\} = \{a,b,c\}$. $\{I_1\} \cap \{I_2\}$ is non-empty, but $\{I_1\} \cap \{I_3\}$ and $\{I_2\} \cap \{I_3\}$ are null sets. LCOM is the (number of null intersections - number of non-empty intersections), which is 1 in this case. LCOM is considered as an inverse cohesion measure. An LCOM value of zero specifies a cohesive class.

2.1.2 Other Static Cohesion Metrics

Briand et al. classify a set of cohesion measures for object-based systems [9,10] which are adapted in [11] to object-oriented systems. For this adaption a class is viewed as a collection of data declarations and methods. A data declaration is a local, public type declaration, the class itself or public attributes. There can be data declaration interactions between classes, attributes, types of different classes and methods.

They categorized the different cohesive metrics based on the above principle into following categories:

1. Ratio of Cohesive Interactions (RCI)
2. Neutral Ratio of Cohesive Interactions (NRCI)
3. Pessimistic Ratio of Cohesive Interactions (PRCI)
4. Optimistic Ratio of Cohesive Interactions (ORCI).

3.1 Run-time/Dynamic Coupling Metrics

While there has been significant work on static metrics there has been little research to date on run-time/dynamic coupling metrics. This section presents the two most relevant works. There are some metrics given by Erik Arisholm [12] which are based on the three below mentioned dimensions:

(i) Mapping – Object or Class

(ii) Direction – Import or Export

(iii) Strength – Number of dynamic messages, distinct methods, or distinct classes.

The following are the different classes of import and export coupling metrics based on the above dimensions.

3.1.1 Import Coupling Metric

Dynamic Import Coupling Metrics count the messages sent from an entity (object or class). Here the entity acts as a client. These metrics are defined next at the object level and the class level.

Object Level Import Coupling Metrics

Dynamic Import Level Object Coupling metrics quantify the extent to which messages are sent from objects to other objects and classes in the system at runtime.

(i) IC_OD Metric: This metric counts the number of messages sent from an object A during a given scenario. (A scenario is the context in which the metric is applicable. The scenarios are then extended to have an application scope.)

(ii) IC_OM : This metric counts the different methods used by an object A to send all the messages in a given scenario x.

(iii) IC_OC : This metric counts the number of distinct classes to which an object A has sent messages in a given scenario x.

Class Level Import Coupling Metrics

These metrics quantify the extent to which messages are sent from a class to other classes and objects in a system at runtime.

(i) IC_CD : This metric counts the number of messages sent from a class A during a given scenario x.

(ii) IC_CM : This metric counts the number of different methods used by a class A to send all the messages in a given scenario x.

(iii) IC_CC : This metric counts the number of distinct classes to which a class A has sent messages in a given scenario x.

3.1.2 Export Coupling Metrics

Dynamic Export Coupling metrics count the messages received by an entity (object or class). Here the entity acts as a server. These metrics are defined next at object-level and the class-level.

Object-Level Export Coupling Metrics

Dynamic Object Level Export Coupling metrics quantify the extent to which messages are received by objects from other objects and classes in the system at runtime.

Class-Level Export Coupling Metrics

Dynamic Class Level Export Coupling metrics quantify the extent to which classes receive messages from other objects and classes in the system at runtime.

(i) EC_CD : This metric counts the number of messages received by a class A during a given scenario x.

(ii) EC_CM : This metric counts the number of different methods used by a class A to receive all the

messages sent to it in a given scenario x.

(iii) EC_CC : This metric counts the number of distinct classes from which a class A has received messages in a given scenario x.

The above classes can also be summarized in the following table representing the variables and the description of every specific variable used over here.

4.1Run-time/Dynamic Cohesion Metrics

Despite extensive research work conducted in the measurement of static cohesion, only a few metrics have been proposed for the measurement of cohesion at runtime.

4.1.1 Gupta et al. Metrics

Bieman and Ott [13,14] proposed the concept of Strong Functional Cohesion (SFC) and Weak Functional Cohesion (WFC) and then Gupta et al.[15] redefined these module cohesion metrics. Gupta et al.[15] commence the dynamic cohesion measurement using program execution based approach on the basis of dynamic slicing (dynamic slice is the set of all statements whose execution had some effect on the value of a given variable). They use dynamic slices of outputs to measure module cohesion. According to them module cohesion metrics based on static slicing approach have got some insufficiencies in cohesion measurement.

Their approach addresses the limitations of static cohesion metrics by considering dynamic behavior of the programs and designing metrics based on dynamic slices obtained through program execution.

They defined SFC as module cohesion obtained from common defuse pairs of each type common to the dynamic slices of all the output variables and WFC as module cohesion obtained from defuse pairs of each type found in dynamic slices of two or more output variables.

4.1.2 Dynamic Metrics for GUI Programs

Though Graphical User Interfaces (GUIs) make the software easier to use from user's viewpoint however they increase the overall complication of the software since GUI programs unlike conventional software are event-based systems. The special characteristics of a GUI program imply that the traditional methods of evaluating complexity statically may not be the suitable ones as static analysis of source code emphasize only on the probability that what may happen when the program is executing whereas a dynamic analysis attempts to enumerate what actually happened during program execution. Mitchell and Power [16] outline a new technique for collecting dynamic trace information from Java GUI programs and a number of simple runtime metrics are proposed.

The exPubMet.Ob metric gives an estimation of level of coupling present in a GUI program and The priMet.ob metric shows that simple programs devote a greater proportion of their method access to the internal working of their classes than the GUI program.

exPubMet.Ob: measure of the level of coupling within a program at runtime.

= Number of External Public methods called/Total Number of Objects created

priMet.ob : measure of the level of cohesiveness within a program.

= Number of Private methods called/Total number of objects created

IV. CONCLUSION

This paper focuses on two very significant factors of complexity measurement of software which are coupling and cohesion. An extensive study of approximately all types of coupling and cohesion metrics has been reported in this paper. The major categories are static coupling and cohesion metrics and dynamic coupling and cohesion metrics. From this in-depth study we find that the static metrics are comparatively easy and simpler to collect because there is no need to execute the software. Static metrics can be obtained at very early stages of program development this is the reason these metrics are widely available. Static metrics are satisfactory to measure the quantity attributes such as size and complexity but as far as quality attributes such as reliability and testability are concerned, use of static metrics are not accurate because static metrics are evaluated only by means of static inspection of the software artifact. Dynamic metrics are calculated on the basis of the data collected during actual execution of the system, and thus reinforce the quality attributes explicitly such as chances of fault occurrences, performance. Thus keeping in view the above limitations of static metrics we see that the dynamic metrics are more precise to use for complexity measurements. However the computation process of dynamic metrics is difficult in comparison to the static once. Also very little work has been done in areas of dynamic coupling and cohesion metrics and need further more investigations. So we can conclude that to avoid computational efforts and also for qualitative measurements, a hybrid approach of static and dynamic metrics can proved to be beneficial one.

REFERENCES

[1] W.P. Stevens, G.J. Myers, and L. L. Constantine. Structured design. *IBM Systems Journal*, 13(2):115-139, 1974.

[2] N.E. Fenton and S.L. Peeger. Software Metrics: A Rigorous and Practical Approach. PWS Publishing Company, Boston, Massachusetts,USA,1997.

[3] Marcela Genero, Mario Piattini and Coral Calero, "A Survey of Metrics for UML Class Diagrams", *Object Technology Journal*, Vol. 4, No. 9, Nov-Dec 2005.

[4] L.C. Briand, J.W. Daly, and J.K. Wust. A unifed framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 25(1):91-121, Jan/Feb 1999.

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

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

[7] W. Li and S. Henry. Object-oriented metrics that predict maintainability. *The Journal of Systems and Software*, 23(2):111-122, 1993.

[8] L.C. Briand, J.W. Daly, and J.K. Wust. A unifed framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering: An International Journal*, 3(1):65-117, 1998.

[9] L.C. Briand, S. Morasca, and V. Basili. Measuring and assessing maintainability at the end of high-level design. In *International Conference on Software Maintenance*, pages 88-97, Montreal, Canada, 1993.

[10] 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.

[11] L.C. Briand, J.W. Daly, and J.K. Wust. A unified framework for cohesion measurement in object-oriented systems. *Empirical Software Engineering: An International Journal*, 3(1):65-117, 1998.

[12] E. Arisholm, L.C. Briand, and A. Foyen. Dynamic coupling measures for object- oriented software. IEEE Transactions on Software Engineering, 30(8):491–506, 2004.

[13] Bieman J M, Ott L M. Measuring functional cohesion. IEEE Transactions on Software Engineering, 1994, 20(8): 644-657.

[14] Ott L M, Bieman J M, Kang B K. Developing measures of class cohesion for object oriented software. In Proc. *The 7th Annual Oregon Workshop on Software Metrics*, Oregon, USA, 1995.

[15] Gupta N, Rao P. Program execution based module cohesion measurement. In Proc. the 16th International Conference on Automated on *Software Engineering (ASE 2001)*, San Diego, USA, Nov. 26-29, 2001, pp.144-153.

[16] Yacoub S, Ammar H, Robinson T. A methodology for architectural-level risk assessment using dynamic metrics. In Proc. 11th Int. Symp. *Software Reliability Eng*, San Jose, Oct. 8-10, 2000, pp.210-221.