International Journal of Scientific Research in Computer Science and Engineering

Review Paper

Volume-1, Issue-2, March-April-2013

ISSN No. 2320-7639

Available online at www.isroset.org

Metrics for Software Components in Object Oriented Environments: A Survey

Aanchal[#1], Sonu kumar[#2]

[#1]Dept. of Comp. Engineering, ACE, Ambala, Haryana, India, aanchalprajapat@gmail.com

[#2]Dept. of Comp. Engineering, ACE, Ambala, Haryana, India, sumit.damla@gmail.com

***Abstract*— Object-oriented Software Engineering, classically refers to OOSE, is the object modeling methodology in software architectures. In this paper, we present obtainable and few Software metrics useful in the different phase of the Object-Oriented Software Development Life Cycle. Metrics have been used progressively in making quantitative and qualitative decisions as well as in risk assessment and reduction. They give software professionals the ability to evaluate software process. Metrics are used by the software industry to enumerate the development, operation and maintenance of software. The practice of applying software metrics to a software process and to a software product is a complex task that requires study and restraint, which brings knowledge of the status of the process and / or product of software in regard to the goals to accomplish. In this paper, metrics for Object Oriented Software Systems are presented. They provide a basis for measuring all characteristics.**

Keywords- Object Oriented, Cohesion, Coupling

I. INTRODUCTION

Object oriented (OO) approach is invented to remove some of flaws encountered in procedural approach. In a large program it is difficult to identify what data is used by which function as a result if we need to modify data, then we also need to modify all function that access that data. Also, it does not model real world problems very well. This is because functions are action-oriented and do not really corresponding to elements of the problem. Five characteristics of Object Oriented Metrics are as follows [5]:

- Localization operations used in many classes
- Encapsulation metrics for classes, not modules
- Information Hiding should be measured & improved
- Inheritance adds complexity, should be measured
- Object Abstraction metrics represent level of abstraction

Object oriented programming treats data as an important element and therefore does not allow it to flow freely around the system. It binds data more closely with the function which operate on them and hence able to protect them from accidental modification from other outside functions. Object oriented program is viewed as collection of interacting objects, objects are instance of class. Each object is capable of receiving messages, processing data, and sending messages to other objects.

II. COMPONENTS OF OBJECT ORIENTATION

A. Data Abstraction:

It is an important property of OO through which the background or unessential details are hiding and only essential part is viewed. OO classes use this concept and are defined as a list of abstract attributes, called data member, and functions, called methods or member functions.[6]

B. Encapsulation:

Encapsulation is the word discover from the word "CAPSULE" which mean to put something in a kind of shell. Binding of data and function into a single unit, called class, is termed as encapsulation. By this data is not accessible by outside function directly. Function inside a class can access that data and modify it.[6]

C. Inheritance:

It is the process through which object of one class can acquire the properties of objects of other class. Main concept in inheritance is that a derived class share common characteristics with the class from which it is derived. It provides the idea of reusability. That is a segment of source code that can be used again to add new functionalities with slight or no modification to new derived class.[6]

D. Polymorphism:

It is a Greek term which means the ability to take more than one form. It is of two types that are: operation overloading and function overloading. It means an operation can perform different operations at different instance of time depending upon on the type of data used. Polymorphism allows object having different internal structure to share the same external interface.[6]

III. COUPLING AND COHESION IN OBJECT ORIENTED ENVIRONMENTS

Coupling between modules / components is their degree of mutual interdependence; lower coupling is better. It is an indication of the strength of interconnections between program units. Highly coupled have program units dependent on each other. Loosely coupled are made up of units that are independent or almost independent. Modules are independent

Corresponding Author: *Aanchal*[#1]

if they can function completely without the presence of the other. Obviously, we can't have modules completely independent of each other. They must interact so that they can produce desired outputs. The more connections between modules, the more dependent they are in the sense that more information about one module is required to understand the other module. Modules tightly coupled if they use shared variables or if they swap control info. Loose coupling if info held within a unit and interface with other units via parameter lists. Tight coupling if shared global data. [2]

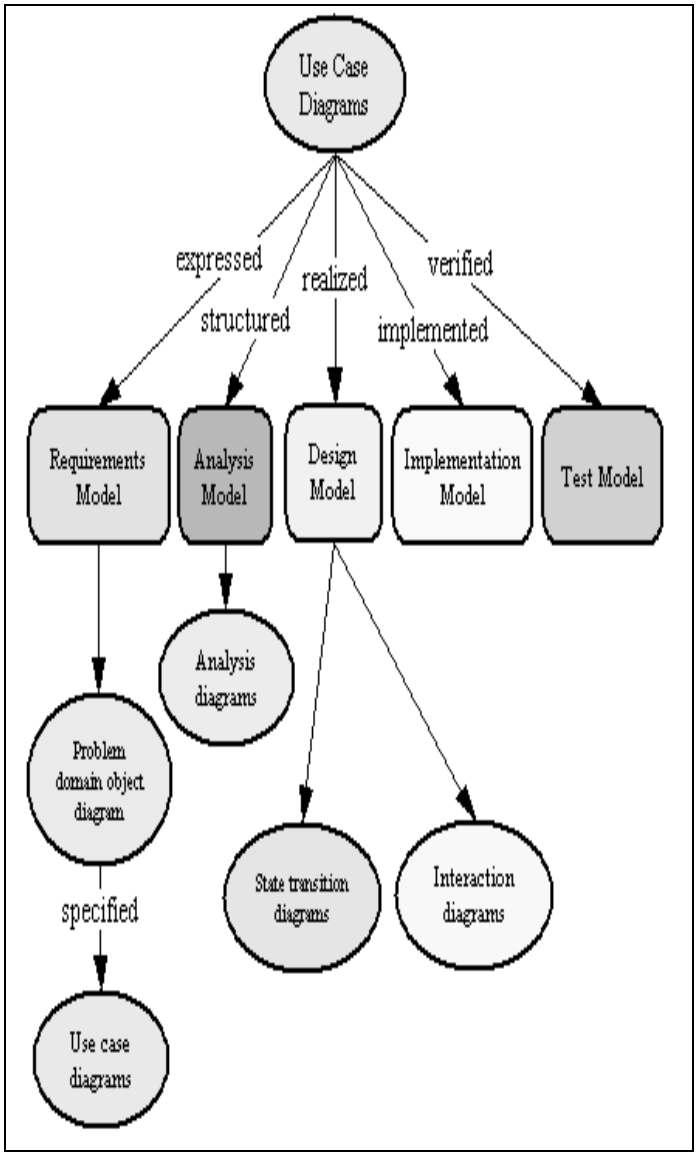

Figure 1 : Object-Oriented Software Engineering

Cohesion of a single module/component is the degree to which its responsibilities form a meaningful unit; higher cohesion is better. It is a Measure of how well module fits together. A component should implement a single logical function or single logical entity. All the parts should contribute to the implementation. Modules with high cohesion tend to be preferable because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understand-ability 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. [2]

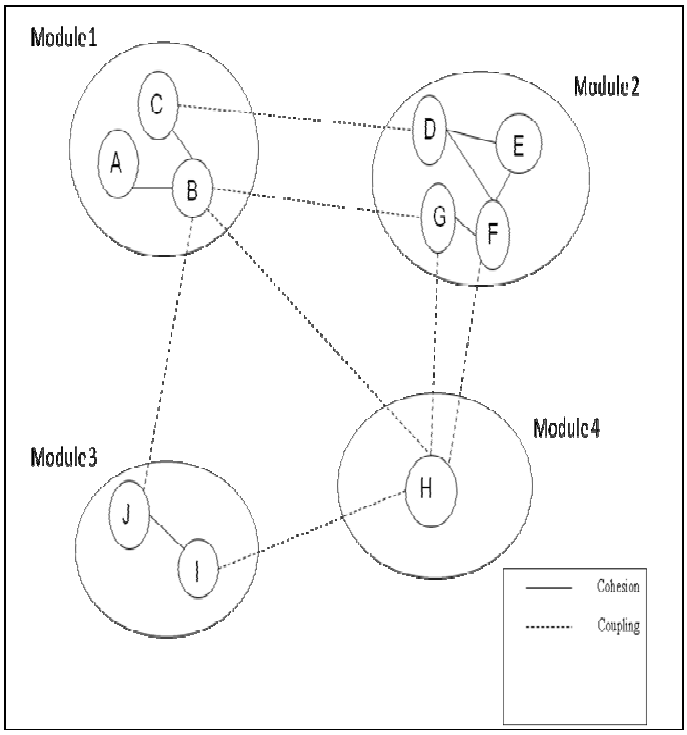

Fig. 2: Coupling and cohesion

In above figure modules are interconnected via coupling and component within the modules are interconnected with each other via cohesion.

IV. Review Of Metrics Usedin Objectoriented Environments

Various object oriented metrics are formed for the object oriented software development. Some of these metrics are CK Metrics, MOOD Metrics, EMOOSE, LI Metrics

A. Shyam R. Chidamber and Chris F. Kemerer (CK) Metrics [7]

Chidamber and Kemerer (CK) et al. [7] proposed metrics suite that have generated a significant amount of interest and are currently the most well known object-oriented suite of measurements for Object-Oriented software. The CK metrics suite consists of six metrics that assess different characteristics of the object-oriented design are-

1) Weighted Methods per Class (WMC): This measures the sum of complexity of the methods in a class. A predictor of the time and effort required to develop and maintain a class we can use the number of methods and the complexity of each method. A large number of methods in a class may have a potentially larger impact on the children of a class since the methods in the parent will be inherited by the child. Also, the complexity of the class may be calculated by the cyclomatic complexity of the methods. The high value of WMC indicates that the class is more complex as compare to the low values.

2) Depth of Inheritance Tree (DIT): DIT metric is used to find the length of the maximum path from the root node to the

end node of the tree. DIT represents the complexity and the behavior of a class, and the complexity of design of a class and potential reuse. Thus it can be hard to understand a system with many inheritance layers. On the other hand, a large DIT value indicates that many methods might be reused. A deeper class hierarchy indicates that the more methods was used or inherited through which this making more complex to predict the behavior of the class and the deeper tree indicates that there is high complexity in the design because all of the facts contained more methods and class are involved. A deep hierarchy of the class may indicates a possibility of the reusing an inherited methods.

3) Number of children (NOC): Number of Children (NOC) metric may be defined for the immediate sub class coordinated by the class in the form of class hierarchy [14, 15]. These points are come out as NOC is used to measure that "How many subclasses are going to inherit the methods of the parent class". Greater is the number of children, greater the potential for reuse, since inheritance is a form of reuse. Also greater is the number of children, the greater the likelihood of improper abstraction of the parent class. The number of children also gave an idea of the potential influence for the class which may be design.

4) Coupling between Objects (CBO): CBO is used to count the number of the class to which the specific class is coupled. The rich coupling decrease the modularity of the class making it less attractive for reusing the class and more high coupled class is more sensitive to change in other part of the design through which the maintenance is so much difficult in the coupling of classes. The coupling Between Object Classes (CBO) metric is defined as "CBO for a class is a count of the number of non-inheritance related couples with classes". It claimed that the unit of "class" used in this metric is difficult to justify, and suggested different forms of class coupling: inheritance, abstract data type and message passing which are available in object oriented programming.

5) Response for class (RFC): The response set of a class (RFC) is defined as set of methods that can be executed in response and messages received a message by the object of that class. Larger value also complicated the testing and debugging of the object through which, it requires the tester to have more knowledge of the functionality. The larger RFC value takes more complex is class is a worst case scenario value for RFC also helps the estimating the time needed for time needed for testing the class.

6) Lack of Cohesion in Methods (LCOM): This metric is used to count the number of disjoints methods pairs minus the number of similar method pairs used. The disjoint methods have no common instance variables in the methods, while the similar methods have at least one common instance variable. It is used to measuring the pairs of methods within a class using the same instance variable. Since cohesiveness within a class increases encapsulation it is desirable and due to lack of cohesion may imply that the class is split in to more than two or more sub classes. Low cohesion in methods increase the complexity, when it increases the error proneness during the development is so increasing.

B. Metrics for Object-Oriented Design (MOOD):

F.B. Abreu et al. [3] defined MOOD (Metrics for Object-Oriented Design) metrics. MOOD refers a structural mechanism of the object oriented paradigm like encapsulation as (MHF, AHF), inheritance (MIF, AIF), polymorphism (POF), and message passing (COF). In MOOD metrics model, there are two main features methods and attributes. Attributes are used to represent the status of object in the system and methods are used to maintained or modifying several kinds of status of the objects [5]. Metrics are defined as:

1) Method Hiding Factor (MHF): MHF is defined as the ratio of the sum of the invisibilities of all methods defined in all classes to the total number of methods defined in the system under consideration. The invisibility of a method is the percentage of the total classes from which this method is not visible. Here inherited methods are not considered.

2) Attribute Hiding Factor (AHF): AHF is defined as the ratio of the sum of the invisibilities of all attributes defined in all classes to the total number of attributes defined in the system under consideration. (iii)Method Inheritance Factor (MIF): MIF is defined as the ratio of the sum of the inherited methods in all classes of the system under consideration to the total number of available methods (locally defined plus inherited) for all classes. It is used to measure the inheritance of the class & also provide the similarity into the classes.

3) Attribute Inheritance Factor (AIF): AIF is defined as the ratio of the sum of inherited attributes in all classes of the system under consideration to the total number of available attributes (locally defined plus inherited) for all classes. It is used to measure the inheritance of the class & also provide the similarity into the classes.

4) Polymorphism Factor (PF): PF is defined as the ratio of the actual number of possible different polymorphic situation for class C_i to the maximum number of possible distinct polymorphic situations for class C_i. Polymorphism potential of the class are used to measure the polymorphism in the particular class & also arise from inheritance.

5) Coupling Factor (CF): CF is defined as the ratio of the maximum possible number of couplings in the system to the actual number of couplings not imputable to inheritance. CF is used to measure the coupling between the classes. the coupling are of two types static & dynamic coupling, due to which is increase the complexity of the class & reduce the encapsulation & potential reuse that provide better maintainability. Software developers for the object-oriented system always avoid the high coupling factor.

C. Extended Metrics for Object-Oriented

Software Engineering Emoose: W. Li et al. [4] proposed this metrics of the Moose model. They may be described as

1) Message Pass Coupling (MPC): It means that the number of message that can be sent by the class operations.

2) Data Abstraction Coupling (DAC): It is used to count the number of classes which an aggregated to current class and also defined the data abstraction coupling.

3) Number of Methods (NOM): It is used to count the number of operations that are local to the class i.e. only those class operation which can give the number of methods to measure it.

4) Size1: It is used to find the number of line of code.

5) Size2: It is used to count the number of local attributes & the number of operation defined in the class.

D. LI Metrics

Li et al. [6] proposed six metrics, these are:

1) Number of Ancestor Classes (NAC): The Number of Ancestor classes (NAC) metric proposed as an alternative to the DIT metric measures the total number of ancestor classes from which a class inherits in the class inheritance hierarchy. The theoretical basis and viewpoints both are same as the DIT metric. In this the unit for the NAC metric is "class", justified that because the attribute that the NAC metric captures is the number of other classes environments from which the class inherits.

2) Number of Local Methods (NLM): The Number of Local Methods metric (NLM) is defined as the number of the local methods defined in a class which are accessible outside the class. It measures the attributes of a class that WMC metric intends to capture. The theoretical basis and viewpoints are different from the WMC metric. The theoretical basis describes the attribute of a class that the NLM metric captures. This attribute is for the usage of the class in an object oriented design because it indicates the size of a class's local interface through which other classes can use the class. They stated three viewpoints for NLM metric as following:

a) The NLM metric is directly linked to a programmer's effort when a class is reused in an Object-Oriented design. More the local methods in a class, the more effort is required to comprehend the class behavior.

b) The larger the local interface of a class, the more effort is needed to design, implement, test, and maintain the class.

c) The larger the local interface of a class, the more influence the class has on its descendent classes.

3) Class Method Complexity (CMC): The Class Method Complexity metric is defined as the summation of the internal structural complexity of all local methods. The CMC metrics theoretical basis and viewpoints are significantly different from WMC metric. The NLM and CMC metrics are fundamentally different as they capture two independent attributes of a class. These two metrics affect the effort required to design, implement, test and maintain a class.

4) Number of Descendent Classes (NDC): The Number of Descendent Classes (NDC) metric as an alternative to NOC is defined as the total number of descendent classes (subclass) of a class. The stated theoretical basis and viewpoints indicate that NOC metric measures the scope of influence of the class on its sub classes because of inheritance. Li claimed that the NDC metric captures the classes attribute better than NOC.

5) Coupling through Abstract Data Type (CTA): The Coupling through Abstract Data Type (CTA) is defined as the total number of classes that are used as abstract data types in the data-attribute declaration of a class. Two classes are coupled when one class uses the other class as an abstract data type [16]. The theoretical view was that the CTA metric relates to the notion of class coupling through the use of abstract data types. This metric gives the scope of how many other classes' services a class needs in order to provide its own service to others.

6) Coupling through Message Passing (CTM): The Coupling through Message Passing (CTM) defined as the number of different messages sent out from a class to other classes excluding the messages sent to the objects created as local objects in the local methods of the class. Two classes can be coupled because one class sends a message to an object of another class, without involving the two classes through inheritance or abstract data type. Theoretical view given was that the CTM metric relates to the notion of message passing in object-oriented programming. The metric gives indication of how many methods of other classes are needed to fulfill the class own functionality.

V. DISADVANTAGES OF METRICS

A. Disadvantages of CK Metrics

1) Weighted Methods per Class (WMC): WMC break an elementary rule of measurement theory that a measure should be concerned with a single attribute. This is also not clear whether the inherited method is to be counted in base class (which defines it), in derived classes or in both.

2) Response for a Class (RFC): Because of practical considerations, Chidamber and Kermerer recommended only one level of nesting during the collection of data for calculating RFC. This gives incomplete and ambiguous approach as in real programming practice there exists "Deeply nested call-backs" that are not considered here.

3) Depth of Inheritance Tree (DIT): But the definition should measures the maximum ancestor classes from the class-node to the root of the inheritance tree.

4) Number of children (NOC): The definition of NOC metric gives the distorted view of the system as it counts only the immediate sub-classes instead of all the descendants of the

class. So the NOC value of a class should reflect all the subclasses that share the properties of that class.

5) Coupling between Object Classes (CBO): As Coupling between Object classes increases, reusability decreases and it becomes harder to modify and test the software system. But for most authors coupling is reuse, which raises ambiguity. So there is the need to find out the coupling level that implies the goodness of design

6) Lack of Cohesion in Methods (LCOM): The high value of LCOM indicates that the methods in the class are not really related to each other and vice versa. According to definition of LCOM the high value of LCOM implies low similarity and low cohesion, but a value of LCOM = 0 doesn't implies the reverse. So the definition of CK metric for LCOM is not able to distinguish the more cohesive class from the less ones. This is simple violation of the basic axiom of measurement theory, which tells that a measure should be able to distinguish two dissimilar entities. So this deficiency offends the purpose of metric.

B. Disadvantages of MOOD Metrics

1) Method Inheritance Factor: Definition of the MIF is inconsistent with the 0-1 scale.

2) Attribute Inheritance factor: The metric Ai (Ci) is meaningless in the sense that the concept of the inheritance concerns the behavior defined in a method, an attribute does not have behavior, and thus cannot be overridden or inherited.

3) Method Hiding Factor: It is recommended that MHF should not be lower than a particular (as yet undefined) value but suggest that there is no upper limit, thus implying that it is 'good' for all methods in a class to be hidden (private). However, the number of private methods in a class doesn't tell us anything about the degree of information hiding in a class. It may tell us that a particular method (or methods) has been broken down into a number of smaller methods to avoid duplication or for clarity of understanding. Such methods would only need to be visible to the containing class. But whether or not a method is broken down this way the containing class's implementation is still hidden.

4) Attribute Hiding Factor: This is a clearly defined metric with no apparent inconsistencies. Its use is in determining the level of visibility of a class's data.

5) Polymorphism factor: It is possible a sub-system will consist of a set of classes that extends a framework. This may be a set of library classes or a framework of low(er) level system classes. When measuring the sub-system it should be only the classes that belong to the sub-system that are measured; classes outside of its boundaries (which are where the framework or library classes will lie) should not be considered. In such cases the denominator for the POF measure may be less than the numerator, resulting in a value greater than one

6) Coupling Factor: This metric is intended to count all client-supplier relationships in a system. The important point here is that the relationship between any two classes in a system is not constrained to just one or the other of these relationship types.

VI. CONCLUSIONS

In this paper, we have presented all of the software metrics for object oriented development. They provided a basis for measuring all of the characteristics like size, complexity, performance and quality. In rely of some notions the quality may be increased by added some features like abstraction, polymorphism and inheritance which are inherent in object orientation. This paper provides some help for researchers and practitioners for better understanding and selection of software metrics for their purposes.

REFERENCES:

[1] Amandeep Kaur, Satwinder Singh, Dr. K. S. Kahlon and Dr. Parvinder S. Sandhu "Empirical Analysis of CK & MOOD Metric Suit", International Journal of Innovation, Management and Technology, Vol. 1, No. 5, December 2010, pp. 447-452

[2] Adam Carlson "Coupling and Cohesion http://www.cs.washington.edu/education/courses/ cse403/96sp/coupling-cohesion.html"

[3] B. F. Abreu: "Design metrics for OO software system", ECOOP'95, Quantitative Methods Workshop, 1995.

[4] W. Li, Sallie, Henry "Metrics for Object-Oriented system", Transactions on Software Engineering, 1995.

[5] Seyyed Mohsen Jamali "Object Oriented Metrics (A Survey Approach)" January 2006

[6] Li W., "Another Metric Suite for Object-oriented Programming", The Journal of System and Software, Vol. 44, Issue 2, December 1998, pp. 155-162.

[7] Shyam R. Chidamber and Chris F. Kemerer "A Metrics Suite for Object Oriented Design" IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994, pp. 476-493.

[8] E. Balagurusamy. "Object-Orinted Programming With C++" Second Edition ,2004

