International Journal of Information Technology and Knowledge Management
July-December 2011, Volume 4, No. 2, pp. 613-621

MERGING OF OBJECT-ORIENTED PARADIGM AND THE SOFTWARE PRODUCT MEASUREMENT PROCESS: AN OVERVIEW

Shubha Jain[1], Praveen Kumar Tripathi[2] & Raghuraj Singh[3]

In the mid 1980s, a new paradigm began to affect the way the software developers viewed software analysis and design. Recent years witnesses the increasing use of this paradigm, the object-oriented paradigm which added a new level of complexity to the study of software measures. The whole software development process has shifted from the procedure oriented to object-oriented environment. It is quite clear that measurement is necessary for the software development process to be successful. This paper provides an overview of the merging of the object-oriented paradigm and the software product measurement process. A survey of most of the existing object oriented metrics has been presented. A classification of object-oriented software measures is created on the basis of their features which will help in choosing the correct set of metrics depending upon the requirements or objectives during the development process.

Keywords: Object Orientation, Measures, Software Metrics, Characteristics, Coupling, Cohesion, Inheritance, Classification of Metrics.

1. INTRODUCTION

1.1 Measures Vs Metrics

Measurement can be considered as mapping from the empirical world to the formal and relational world. It is the number or symbol assigned to an entity in order to characterize an attribute. It is the process of empirical, objective, assignment of numbers to properties of objects or events of the real world in such a way as to describe them. Software plays an important role in our life. We need quality software.

To determine quality of the software we must have some way to measure quality. Metric is a measurement function. A software quality metric is a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality.

Software metrics play a central role in the planning and control of software development projects. Metrics are used to provide a quantitative basis for the development and validation of models of the software development process. It can be used to improve software productivity and quality and identify potential areas of problems that may lead to problems or errors.

[1,2]Department of Computer Science & Engineering, Kanpur Institute of Technology, Kanpur (India)

[3]Department of Computer Science & Engineering, Harcourt Butler Technological Institute, Kanpur -208001(India)

E-mail: [1]shubhj@rediffmail.com, [2]praveentri81@gmail.com, [3]raghurajsinghj@rediffmail.com

1.2 Object Oriented Paradigm

Object orientation has rapidly become accepted as the preferred paradigm in industrial software development environments for large scale system design [10]. This paradigm, the object-oriented paradigm, has compounded the study of software measures because of the multiplicity of interrelated elements.

2. TRADITIONAL VS OBJECT ORIENTED TECHNIQUES

Traditional design techniques separate data and procedures while object-oriented designs combine them. Traditional metrics at high level are not capable to measure a large-scale system's complexity. There are multiple dimensions that an OO metric must have if it is to provide accurate effort prediction or productivity tracking. Objected-oriented software products are different from earlier or conventional software products as they use procedures and subprograms.

The conventional technique of structured programming is procedure-oriented, but is supported by programming languages that support separate compilation of modules, data abstraction, strong data typing, and data encapsulation. This emphasizes more on implementation phase in the life cycle of software. In contrast, object-oriented programming places greater emphasis on the design phase of the software life cycle. The essence of the object-oriented design is that it decomposes the system into object classes, gathers together the data and the functions to be performed on the data and encapsulates the data and functions (methods) within the class.

The basic building block of the object-oriented approach is the object class. Features of the object class

that become measurable are the number of attributes the class contains, the number of methods the class has, the number of methods called from other classes, the number of methods outside the class that are called, and the placement of the particular class in the class inheritance tree.

One of the key differences between the procedure-oriented implementation and the object-oriented implementation is the concept of Inheritance. Inheritance is a relationship among classes in which a class inherits or shares the attributes and methods of another class.

The methods of a class are similar to the functions, programs, or subprograms that are used in conventional programming. Functionality in classes is gained through message passing both within classes and between classes. A class's methods are measurable. Methods can be measured by the earlier, more conventional measures. Examples of such measures are lines of code, Halstead's software science metrics, McCabe's cyclomatic complexity metric, and Albrecht's function points.

3. TERMINOLOGY FOR OBJECT-ORIENTED APPROACHES

The term object is considered to be a primitive term. Objects have attributes, methods, and an identity (a name). The following terminology is a partial adaptation of Booch's set of terms [25].

Abstraction: The essential characteristics of an object that distinguish it from all other kinds of objects, and thus provide, from the viewer's perspective, crisply-defined conceptual boundaries; the process of focusing upon the essential characteristics of an object.

Aggregate object (aggregation): An object composed of two or more other objects.

Attribute: A variable or parameter that is encapsulated into an object.

Class: A set of objects that share a common data structure called attributes and a common behaviour manifested by a set of methods; the set serves as a template from which objects can be created.

Cohesion: The degree to which the methods within a class are related to one another. Here, cohesion is limited to cohesion within a class.

Collaborating classes: If a class sends a message to another class, the classes are said to be collaborating.

Coupling: Class X is coupled to class Y if and only if X sends a message to Y.

Depth: The depth of a class is the length of the longest path from the root of the inheritance tree to the class in question.

Encapsulation: The process of bundling together the elements of an abstraction that constitutes its structure and behaviour.

Information hiding: The process of hiding the structure of a class and the implementation details of its methods. A class has a public interface and a private representation; these two elements are kept distinct.

Inheritance: A relationship among classes, wherein one class shares the structure or methods defined in one other class (for single inheritance) or in more than one other class (for multiple inheritance).

Inheritance Tree: A directed graph in which the nodes represent classes and the edges represent base-class/derived-class dependencies. The graph may not be a tree if multiple inheritances are permitted.

Instance: An object with specific structure, specific methods, and an identity.

Instantiation: The process of filling in the template of a class to produce a class from which one can create instances.

Message: A request made of one class to another, to perform an operation.

Method: An operation upon a class, defined as part of the declaration of a class.

Polymorphism: The ability of two or more objects to interpret a message differently at execution, depending upon the superclass of the calling object.

Superclass: The class from which another subclass inherits its attributes and methods.

Uses: If class X is coupled to class Y and class Y is coupled to class Z, then class X uses class Z.

It has been a common goal of many researchers to agree upon a common set of terminology that encompasses the object-oriented analysis and design methods used in the object-oriented community.

4. A SUGGESTIVE CLASSIFICATION OF OBJECT-ORIENTED SOFTWARE METRICS BASED ON THE OBJECT ORIENTED CONCEPTS

There has been little agreement among authors as to the characteristics that identify the object-oriented approach. For e.g., Henderson-Sellers [26] list information hiding, encapsulation, objects, classification, classes, abstraction, inheritance, polymorphism, dynamic binding, persistence, and composition as a defining aspect of object-orientation. Different authors group and create their own individual groupings, each with a unique aspect name. It should be clear that there are many dimensions to O-O software. It should also be noted that this list should not be exhaustive.

Ladan Tahvildari and Ajit Singh [2] propose a taxonomy that places a metric in one or more of five category, viz., system, coupling and uses, inheritance, class Interface, and class implementation.

The object oriented design approach begins with the high level characteristics of an object oriented system and moves down to the low-level characteristics. It can be classified in following hierarchical manner as:

System: In OO design, the system and its components are at highest level. Any system can be further divided into components which can also be treated as a system. The characteristics of a good component are those of a good system and vice versa.

Coupling and Uses: A subsystem is made up of by several classes and they can interact with each other. The interaction of different classes indicate a complexity resulting from too much coupling, or from using objects derived from objects that have been obtained from yet another object. This type of complexity results in making the process more complex. Coupling and Uses are related to each other. The Use is defined in terms of coupling and hence both can be grouped in a single unit.

Inheritance: Classes are found in a class structure diagram, often called an inheritance tree or class hierarchy graph. Visible in the graph are the inheritance relationships between classes and their parents—the properties shared by both. Such relationships may indicate to a designer where changes would improve the development.

Class: The class is the main feature of object oriented concept that describes one central location the state and the generic behavior of a set of objects. Classes have many characteristics that are measurable and may have characteristics that make them excellent candidates for inclusion in a library for reuse.

Method: Attributes and methods occur at the finest level of detail in the class. Methods are usually implemented much like procedures are in structured programming. However, in object-oriented products methods have the additional complexity of message passing. Messages can be passed to objects in the same class or to objects in different classes. Message passing involves accessing features of other objects that are visible (public) and some that are invisible (private) to the object. Such accesses should be measured or recorded.

A useful software measures classification should clearly differentiate among category so that there is no ambiguity as to the category to which a measure belongs. This classification attempts to cover all the characteristics of object oriented software concept and to capture the features of the design from the system level down to the class level.

5. CLASSIFICATION OF EXISTING MEASURES

5.1 System Measures

1. OC (Object Counts): This metric is defined as count of object instances in the system. [3]
2. OP (Object Points): This metric counts the weighted instances of an object. [3]
3. RL (Reuse Leverage): This metric is defined as the ratio of total of objects in the system to the number of unique objects built for the system. [3]
4. Size (Size of Object-Oriented system): This metric is defined as a statistical estimate based on a model developed by the author. [1b]
5. HC(Hierarchy Complexity): This metric is used to define the hierarchical complexity of the system of system. [7]
6. PC(Program Complexity): This metric is defined as the sum of the complexity of the main program and the complexity of the class hierarchies in the system) [7]
7. CBC (Count of Base Classes): This metric defines the number of base classes. [9]
8. CSC (Count of Standalone Classes): This metric is used to define the number of standalone classes. [9]

5.2 Coupling and Uses Measures

1. OCM (Operation Coupling Metric): This metric is defined as a count of the number of operations that access other classes, are accessed by other classes, and cooperate with other classes. [4]
2. CCM (Class Coupling Metric): This metric is used to define the counts accesses between classes. [4]
3. CBO (Coupling Between Object classes): This metric is defined as the count of coupling, where coupling is considered as bi-directional. [5]
4. MPC (Message Passing Coupling): This metric defines the number of send statements in a class. [6]
5. NOT (Number of Tramps): This metric is used to count of extraneous parameters that are not involved in any message passing. [8]
6. VOD(Violations of the Law of Demeter) [8]
7. CCR (Counts of Contains Relationship): This metric is used to count number of contains relationships in a class. [9]
8. COU(Count of Uses) [9]

9. GSDM(Graph of the source and Destination of all Messages) [11]
10. CF (Coupling Factor): This metric represents the percentage of couplings among classes, not imputable to inheritance, with respect to the maximum possible number of couplings in the class diagram. The CF is computed by dividing the number of associations, not related to inheritance, between all classes by the number of classes squared minus the number of classes. [12], [13], [14]
11. CTM (Coupling through message passing): 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. [16]
12. DCC (Direct Class Coupling): This metric counts the different number of classes that a class is directly related to. The metric includes classes that are directly related by attribute declarations and message passing (parameters) in the methods.[18]
13. CPM(Coupling Metric): This metric count of the different number of classes that a class is directly related to [19].

5.3 Inheritance Measures

1. AID (Average Inheritance Depth): This metric is used to give the average depth of inheritance.[1b]
2. CHM (Class Hierarchy Metric): This metric is defined as the summation of a specific class in the inheritance tree, the number of subclasses of the class, the number of 'direct' superclasses of the class, and the number of local or inherited operations available to the class.
3. DIT (Depth of Inheritance Tree): This metric is defined as the maximum length from the node to the root of the tree. [5]
4. NOC (Number of Children): This metric defined as the number of immediate subclasses. [5]
5. IL(Inheritance Lattice) [11]
6. MIF (Method Inheritance Factor): This metric represents the percentage of effective inheritance of methods. The MIF is computed by dividing the number of all inherited methods in all classes by the sum of all methods available (inherited and locally defined) of all classes. [12], [13], [14]
7. AIF (Attribute Inheritance Factor): This metric gives the percentage of effective inheritance of attributes. The AIF is computed by dividing the number of all inherited attribute in all classes by the sum of all attributes.[14]
8. PF (Polymorphism Factor): This metric represents the actual number of possible different polymorphic situations with respect to the maximum number of possible distinct polymorphic situations. The PF is computed by dividing the total number of overridden methods in all classes by the result of multiplying the number of new methods times the number of descendants for all classes, respectively. [12], [13], [14]
9. NMI Number of Methods Inherited): This metric measures the number of methods inherited by a subclass. No mention is made as to whether that inheritance is public or private. [15]
10. NMO (Number of Methods overridden): This metric give the number of overridden methods. A large number of this metric indicates a design problem, indicating that those methods were overridden as a design afterthought. [15]
11. SIX (Specialization Index): This metric is calculated as (NMO*DIT)/NM This metric looks at the quality of the classes use of inheritance. This measure to what extent subclasses redefine the behaviour of their super classes. [15]
12. NNA (Number of New Methods): The normal expectation for a subclass is that it will further specialize (or add) methods to the super class object. A method is defined as an added method in a subclass if there is no method of the same name in any of its super classes. [15]
13. NAC(Number of ancestor classes): This metric is 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. [16]
14. NDC (Number of descendent classes): 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.
15. NOH(Number Of class Hierarchies): This metric counts the number of class hierarchies in the design.[18]
16. ANA (Average Number of Ancestors): This metric gives the average number of classes from which a class inherits information. It is computed by determining the number of classes along all paths from the root class(es) to all classes in an inheritance structure.[18]

17. MFA(Measure of Functional Abstraction): This metric give is the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class.[18]
18. REM (Reuse Metric): This metric counts number of class hierarchies in the design. [19]

5.4. Class Measures

1. RFC1 (Raw Function Counts): This metric is used to represent a simple count of the five function types from function point's analysis. [3]
2. OXM(Operation Complexity Metric): This metric gives the Operation complexity within a class.[4]
3. RM (Reuse Metric) [4]
4. OACM(Operation Argument Complexity Metric) [4]
5. ACM(Attribute Complexity Metric) [4]
6. CM(Cohesion Metric) [4]
7. RFC2(Response For a Class): This metric gives the cardinality of the set of all methods that can potentially be executed in response to a message received by an object of the class being measured.[5]
8. LCOM (Lack of Cohesion of Methods): This metric defines the number of different methods within a class that reference a given instance variable. [5]
9. WMC (Weighted Methods per Class) : This metric gives the sum of the complexities of the methods in a class. [5]
10. CC (Class Complexity): This metric give the sum of the individual method complexities. [7]
11. DAC (Data Abstraction Coupling): This metric is used to count number of abstract data types.[6]
12. NOM (Number of local Methods): This metric is used to count of number of methods in a class. [6]
13. Size2(number of attributes + number of local methods)[6]
14. WAC(Weighted Attributes per Class) [8]
15. MHF (Method Hiding Factor): This metric represents the percentage of invisibilities of methods in a class diagram (degree of methods encapsulation). The MHF is computed by dividing the number of all visible methods in all classes by the number of all methods in all classes. [12], [13], [14]
16. AHF (Attribute Hiding Factor): This metric represents the percentage of invisibilities of attributes in a class diagram (degree of attributes encapsulation). The AHF is computed by dividing the number of visible attributes in a class diagram by the number of all attributes in a class diagram. [12], [13], [14]
17. NPM (Number of Public Methods): This is a count of the number of public methods in a class. It is used to help estimating the amount of work to develop a class. [15].
18. NM (Number of Methods): This metric is used to count the total number of methods in a class counts all public, private and protected methods [15].
19. NPV (Number of Public Variables per class): This metric counts the number of public variables in a class. [15].
20. NV (Number of Variables per class): The total number of variables including public, private and protected variables. [15].
21. NCM (Number of Class Methods) [15]
22. NCV (Number of Class Variables) [15].
23. CTA (Coupling through abstract data type): 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. [16].
24. DSC (Design Size of Class): This metric is a count of the total number of classes in the design. Data Access Metric (DAM): This metric is the number of private (protected) attributes to the total number of attributes declared in the class [18].
25. MOA (Measure Of Aggregation): This measure the extent of part-whole relationship, realized by using attributes. The metric is count of the number of data declarations whose types are user defined classes.[18]
26. CIS(Class Interface Size): This metric is count of the number of public methods in a class.[18]
27. NOM (Number of Methods): This metric is a count of all the methods defined in the class[18]
28. CAM (Cohesion among Methods of class): This metric computes the relatedness among methods of a class based upon the parameter list of the methods. The metric is computed using the summation of the interaction of parameters of a method with the maximum independent set of all parameter types in the class.[18]

29. ENM(Encapsulation Metric): This metric counts all the methods defined in a class.[19]
30. COM (Cohesion Metric): This metric computes the relatedness among methods of a class. [19]
31. LORM (Logical Relatedness of Methods): It is defined as total number of relations in the class/ total number of possible relations in the class. [20]
32. CDC (Class Domain Complexity): This metric measures the domain complexity of a class. [20]
33. RCDC (Relative Class Domain Complexity): It is defined as CDCI maximum CDC for any class in the same system. [20]
34. CIC (class interface complexity) and SCIDE (Semantic Class Interface Definition Entropy) : These metrics measures the complexity of the interface methods of a class. [20]
35. COa (Class Overlap A), Cob (Class Overlap B): These metrics determine the overlap of functionality between two classes by examining the concepts that appear in more than one class. [20]
36. OCDQa (Overall Class Documentation Quality A), OCDQb (Overall Class Documentation Quality B), CCQ (Class Comment Quality) and CIQ (Class Identifier Quality): The documentation quality of a class can be measured by examining the level of useful information provided by the identifiers of the class (names of methods, attributes, etc.), and by examining the level of useful information provided by the comments in the class. [20]
37. RCI (Ratio of Cohesive Interactions): The RCI metric is defined as the ratio of the number of cohesive interactions of a module to the total number of possible cohesive interactions. [21]
38. CAMC (Cohesion among Methods in a Class): The CAMC metric is defined as the ratio of the total number of 1s in the matrix to the total size of the matrix.[21]
39. NHD (Normalized Hamming Distance): The metric calculates the average of the parameter agreement between each pair of methods. The parameter agreement between a pair of methods is defined as the number of places in which the parameter occurrence vectors of the two methods are equal.[21]
40. NHDM (NHD Modified): This metric counts the method pairs which agree on a 0, as a disagreement. [23]

5.5 Method Measures

1. FP (Function Points][13]
2. SSM (Software Science Metrics)[1c]
3. MCC (McCabe's Cyclomatic Complexity): Complexity metrics can be used to calculate essential information about constancy and maintainability of software system from source code. It also provides advice during the software project to help control the design.
4. MC (Method Complexity): This metric is used for computing the complexity. It involves length of method, number of arguments, coupling to other methods, etc. [7]
5. Size1: This metric is used to count the number of semi-colons in a class [6]
6. LOC (Lines of executable source Code): There are many interpretations of this measure. Park is one of the best sources of guidelines for collecting this data.
7. APM (Average parameters per Method): This metric defines the total Number Parameters in a class / Total number of methods. Lorenz and Kidd argue that APM should not exceed. [15].
8. NLM (Number of local methods): 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. [16]
9. CMC (Class method complexity): The Class Method Complexity (CMC) metric is defined as the summation of the internal structural complexity of all local methods.[16]
10. Comment Percentage (CP): The CP metric is defined as the number of commented lines of code divided by the number of non-blank lines of code. The comment percentage is calculated by the total number of comments divided by the total lines of code less the number of blank lines. [17]
11. NOP (Number Of Polymorphic methods): This metric is a count of the methods that can exhibit polymorphic behavior.[18]

6. SUMMARY OF MEASURES FOR EACH CATEGORY

Table 6.1 provides a summary of the measures given for the five categories. The table contains the category names by column and the authors by row. This table contains almost all the measures proposed in the literature.

Table 6.1
Classification of Metrics by Category

Sr. No	Author	System	Coupling & Uses	Inheritance	Class	Method
1.	Laranjeira 90	Size				
2.	Moreau 90 ab	GSDM		IL		SSM, MCC
3.	Banker 91	OC, OP, RL			RFC1	FP
4.	Coppick 92					SSM, MCC
5.	Tegarden 92					SSM, MCC LOC
6.	Lee 92	HC, PC			CC	MC
7.	Chen et al. 1993		CCM, OCM	CHM	OXM, RM, OACM, ACM, CM	
8.	Li, Wei, Henry, Salley 1993	HC, PC			CC	MC
9.	Li, Wei, Henry, Salley 1993		MPC		DAC, Size 2, NOM	SIZE 1
10.	Sharble 93		NOT VOD		WAC	
11.	Williams 93	CBC, CSR	CCR, COU			
12.	Yap 93a			AID		
13.	Chidamber 94 (MOOSE)		CBO	DIT, NOC	WMC, RFC, LOCM	
14.	Fernando Brito Abreu and Rogerio Carpuca in 1994 (MOOD)		CF	MIF, AIF PF	MHF, AHF	
15.	Lorentz & Kidd in 1994, (L&K)			NMI, NMO SIX, NNA	NPM, NM NPV, NV NCM, NCV	APM
16.	Rosenberg et al. in 1997		CBO	DIT, NOC	WMC, RFC, LOCM	CC, SLOC, CP
17.	Wei Li in 1998 (Li)		CTM	NAC, NDC	CTA	NLM, CMC
18.	J Bansiya, 1999				CAMC	
19.	Letha Etzkom et al. in 2000 (semantic metric suite)				LORM, CDC RCDC, CIC SCIDE, COa, COb, OCDQa, OCDQb, CIQ, CCQ	
20.	J. Bansiya et al. in 2002 (QMOOD)		DCC	NOH, ANA, MFA	DSC, DAM, MOA, CIS NOM, CAM	NOP
21.	Counsell, S 2006				NHD, SNHD	
22.	Raees A. Khan et al. 2009 (MTMOOD)		CPM	REM	ENM, COM	
23.	Kuljit Kaur, Hardeep Singh, 2011				NHDM	

7. CONCLUSIONS

It has been pointed out that any single metric is inadequate to capture all aspects of software development process and respective product [27]. Many researchers have contributed towards collections of metrics which measure the various dimensions of the software development effort thus adapted to the characteristics of object technology. But metrics have not been validated theoretically and most of the metrics are accepted by practitioners on heavy usages and popularity and by academic experts on empirical post development validation [10]. Thus many of the available metrics may not be used and discarded. Moreover, OO metrics try to capture different aspects of software product [5]. Some of the metrics also try to capture the same aspects of software e.g. there are number of metrics to measure the coupling between different classes. We need to identify the necessary metrics that provide useful information, otherwise the developer will be lost into so many numbers and the purpose of metrics would be lost [28].

For about a decade, researchers have been discussing whether a separate set of object oriented software metrics is needed and what unique measures this set should include. This classification of existing metrics based on their characteristics help the practitioners to develop their own suite by choosing the appropriate metrics to evaluate the quality of an object oriented system to meet their objectives or requirements.

REFERENCES

[1a] Yap, L.M. & Henderson-Sellers., Brian., "Consistency Considerations of Object-Oriented Class Libraries", (Research Report 93-3). Sydney, Australia: University of New South Wales, 1993.

[1b] Laranjeira, Luiz., "Software Size Estimation of Object-Oriented Systems", IEEE Transactions on Software Engineering 16, 5 (May 1990): 510-522.

[1c] Tegarden, David P.; Sheetz, Steven D.; & Monarchi, D.E., "Effectiveness of Traditional Metrics for Object-Oriented Systems", pp. 359-368, Proceedings 25th Hawaii International Conference on System Sciences 4. Kauai, HI, January 7-10, 1992. Los Alamitos, CA:IEEE Computer Society Press, 1991.

[2] Laden Tahvildari., "Categorization of Object Oriented Software Metric", 2000.

[3] Banker, Rajiv D.; Kauffman, Robert J.; & Kumar, Rachina., "An Empirical Test of Object-based Output Measurement Metrics in a CASE Environment", Journal of Management Information Systems 8,3 (Winter 1991): 127-150.

[4] Chen, J-Y. & Lum, J-F., "A New Metric for Object-Oriented Design", Information of Software Technology 35, 4 (April 1993): 232-240.

[5] Chidamber, Shyam & Kemerer, Chris F., "A Metrics Suite for Object-Oriented Design", IEEE Transactions on Software Engineering 20, 6 (June 1994): 476-493.

[6] Li, Wei & Henry, Salley., "Maintenance Metrics for the Object Oriented Paradigm", pp. 52-60., Proceedings: First International Software Metrics Symposium. Baltimore, MD, May 21-22, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

[7] Lee, Yen-Sung; Liang, Bin-Shiang; & Wang, Feng-Jian., "Some Complexity Metrics for Object-Oriented Programs Based on Information Flow", pp. 302-310. Proceedings: CompEuro. Paris-Ivry, France, May 24-27, 1993. Los Alamitos, CA: IEEE Computer Society Press, 1993.

[8] Sharble, Robert C. & Cohen, Samuel S., "The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods", SIGSOFT Software Engineering Notes 18, 4 (April 1993): 60-73.

[9] Williams, John D., "Metrics for Object Oriented Projects," pp. 13-18. Proceedings: Object Expo Euro Conference, London, U.K., July 12-16, 1993. New York, NY: ACM SIGS Publications, 1993.

[10] Khan R. A, Mustafa, K., "Metric Based Testability Model for Object Oriented Design (MTMOOD)", IGSOFT Software Engineering Notes, March 2009, 34 Number 2.

[11a] Moreau, Dennis R. & Dominick, Wayne D., "A Programming Environment Evaluation Methodology for Object-Oriented Systems: Part I-The Methodology", Journal of Object-Oriented Programming 3, 1 (May/June 1990): 38-52.

[11b] Moreau, Dennis R. & Dominick, Wayne D., "A Programming Environment Evaluation Methodology for Object-Oriented Systems: Part II - Test Case Application", Journal of Object-Oriented Programming 3, 3 (September/October 1990): 23-32.

[12] Fernando Brito e Abreu and Rogério Carapuça,1994,"Object-Oriented Software Engineering: Measuring and Controlling the Development Process", 4th Int. Conf. on Software Quality, McLean, VA, USA, 3-5 October 1994.

[13] Fernando Brito e Abreu Miguel Goulão, Rita Esteves,1995, "Toward the Design Quality Evaluation of Object-Oriented Software Systems", 5th International Conference on Software Quality, Austin, Texas, 23 to 26 October 1995.

[14] Fernando Brito e Abreu and Walcélio Melo,1996, "Evaluating the Impact of Object-Oriented Design on Software Quality", 3rd Int'l S/W Metrics Symposium, March 1996, Berlin, Germany.

[15] Mark Lorenz, Jeff Kidd,1994, "Object-Oriented Software Metrics", Prentice Hall; ISBN: 013179292X.

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

[17] L. H. Rosenberg and L. Hyatt, "Software Quality Metrics for ObjectOriented Environments", Crosstalk Jounal, 1997.

[18] J. Bansiya and C.G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transactions on Software Engineering, 28, No. 1, 2002.

[19] Khan Raees A., Mustafa Khurram, "Metric Based Testability Model for Object Oriented Design (MTMOOD)", ACM SIGSOFT Software Engineering Notes 34(2): 1-6 (Feb 2009)

[20] Etzkorn, L, Delugach, H, "Towards a Semantic Metrics Suite for Object-Oriented Design", 34th International

Conference on Technology of Object-Oriented Languages and Systems, 2000. Proceedings, Object Identifier: 10.1109/TOOLS.2000.868960, Page(s): 71 – 80.

[21] Counsell, S., Swift, S., and Crampton, J., "The Interpretation and Utility of Three Cohesion Metrics for Object-Oriented Design," ACM Trans. Software Eng. and Methodology, 15, no. 2, pp. 123- 149, 2006.

[22] Bansiya, J., Etzkorn, L., Davis, C., and Li, W, "A Class Cohesion Metric for Object Oriented Designs", Journal of Object Oriented Programming, 11, No. 8, pp 47-52, 1999.

[23] Dallal, J., "A Design-Based Cohesion Metric for Object-Oriented Classes", Proceedings of the International Conference on Computer and Information Science and Engineering (CISE 2007), pp 301-306, 2007.

[24] Kuljit Kaur, Hardeep Singh,"Towards a Valid Metric for Class Cohesion at Design Level", Second International Conference on Emerging Applications of Information Technology, 2011.

[25] Booch, G., "Object-oriented Analysis and Design with Applications", Second Edition, Benjamin Cummings, 1994.

[26] Henderson-Sellers, B., "A Book of Object-Oriented Knowledge, Prentice Hall, NY, 1992.

[27] Jitender Kumar Chhabra*, Ravinder Singh Bhatia* & V. P. Singh, "Object-oriented Metrics Suites & Complexity: A SURVEY", International Journal of Information Technology and Knowledge Management January June 2009, 2, No. 1, pp. 25-31.

[28] Kaur Amandeep, Singh Satwinder and Kahlon K. S, "Evaluation and Metrication of Object Oriented System", Proceedings of the International MultiConference of Engineers and Computer Scientists 2009, I IMECS 2009, March 18 - 20, 2009, Hong Kong].