Joining ISO Model with Metrics Using Design Quality Properties

Zineb Bougroun, Adil Zeaaraoui, M.G. Belkasmi, T. Bouchentouf
Laboratory of Applied Mathematics
Signal Processing and Computer Science
Department of Computer Science
University Mohamed Premier
Oujda postal 60000, Morocco
bougroun.zineb@gmail.com

ABSTRACT: *In order to make software assessment focused on object oriented principals, we propose a classification based on ISO model using properties of the object-oriented design such as: abstraction, inheritance, encapsulation etc. In this paper we present some quality models, then we make the relationship between sub-characteristics of ISO model and properties of both the object-oriented design and the quality. At the end we present a classification for the most popular oriented object metrics based on design and quality properties.*

Keywords: Quality, Metric, ISO Model, Design Properties, Object-Oriented Model

Received: 21 August 2012, Revised 9 October 2012, Accepted 18 October 2012

© 2012 DLINE. All rights reserved

1. Introduction

To reduce the cost of software maintenance, it is essential to care about the quality of software. For measuring this quality, several metrics have emerged to assess it. Our research is included in this aspect, focussing on quality and quality model applied to object-oriented systems.

A quality model is generally defined by a several factors. Which are decomposed into several criteria and each criterion is defined by a set of metrics that assesses it. Among the quality models there is the ISO standard which structures quality by six characteristics and decomposes them into 22 sub-characteristics, which are on their turn connected to the metrics. In order to bring out the properties of the object-oriented design, we have included some criteria of quality and design between the sub-characteristics of ISO model and metrics.

In this paper, we begin by presenting the definition of quality and we present some quality models whose ISO model, in the second section we propose some properties of design and quality and we join them with ISO model, in the third part of the paper we present some popular metrics and discuss weaknesses and strengths of each one, then we classify them by properties.

2. Quality and model quality

2.1 Definition of quality

The word quality is often used to signify the relative worth of things in such sentences as "*good quality*", "*bad quality*". But

what do we know about the exact meaning of the word "*quality*"?

For JOSEPH M. JURAN. JURAN, this word has multiple meanings [20]. Two of those meanings dominate the use of the word:

1. Quality consists of those product features which meet the needs of customers and thereby provide product satisfaction.

2. Quality consists of freedom from deficiencies.

ARMAND V. FEIGENBAUM defines it as [20]: The total composite product and service characteristics of marketing, engineering, manufacture and maintenance through which the product and service in use will meet the expectations of the customer.

The concept of the quality in ISO 9126 combines the key point of the two previous definitions; quality is "*the totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs*". Otherwise, quality is determined by the presence or absence of the attributes. This definition was derived from the ISO 8402 (Quality vocabulary) definition of quality.

2.2 Quality models

2.2.1 McCall model

Among the oldest model we find the McCall model and several others derived from it. The McCall quality model has three perspectives: product revision, product transition and product operations. McCall started with 55 factors, and then he reduces them to eleven characteristics, as shown in figure 1:

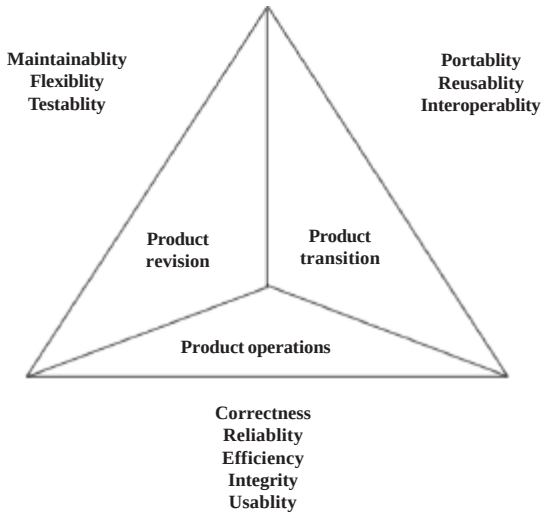

Figure 1. McCall model factors

The model factors are related to 23 quality criteria which are related to metrics that are used to provide a scale and method for measurement. The metric is achieved by answering yes and no questions. At this matter, answering an equally amount of "*yes*" and "*no*" to the questions measuring of a quality criterion; will achieve 50% of quality on that criterion [21].

2.2.2 Boehm model

The second model that will be presented in this paper is that of Barry W. Boehm [21]. Boehm addresses the shortcomings of models that automatically and quantitatively evaluate the quality of software. In essence his model attempts to qualitatively define software quality by a given set of attributes and metrics. Boehm's model is similar to the McCall Quality Model in that it also presents a hierarchical quality model. Boëhm lists seven factors: efficiency, reliability, usability, portability, understandability, flexibility, testability.

2.2.3. ISO model

The third model is the model of the standard ISO; this model was based on the McCall and Boehm models. It was structured in

the same manner as these models as shown in Figure 2.

The ISO/ IEC 9126-1 define a quality model with six characteristics [6]:

• **Functionality:** is the capability of the software to provide functions which meet the stated and implied needs of users under specified conditions of usage.

• **Reliability:** is the ability of the software to perform its required functions under stated conditions for a specified period of time, or for a specified number of operations.

• **Usability:** is the capability of the software to be understood, learned, used and liked by the user, when used under specified conditions.

• **Efficiency:** is the capability of the software to provide the required performance, relative to the amount of resources used, under stated conditions.

• **Maintainability:** is the ease with which a software product can be modified to correct defects, modified to meet new requirements, modified to make future maintenance easier, or adapted to a changed environment.

• **Portability:** is the ease with which the software product can be transferred from one environment to another.

The six characteristics of ISO model are related to twenty two sub-characteristics which are explained in the following table [7]:

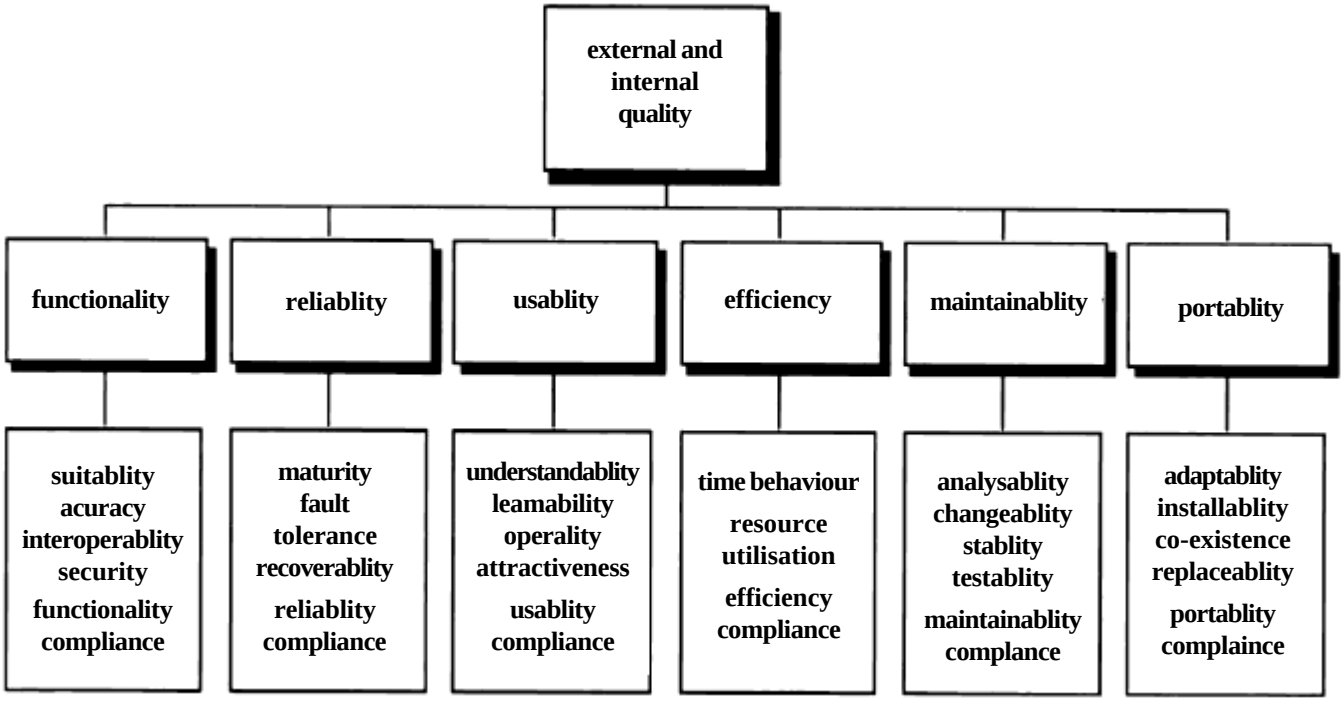

Figure 2. ISO model characteristics

ISO provides internal and external metrics for measuring the defined sub-characteristics quality. The internal metrics measure the software. They may be applied to a non executable software product during its development stages such as a specification design or source code. The external metrics measure the environment that includes the software. Our approach is to join the metrics to the sub-characteristics of ISO model using the properties of quality related to the oriented object system.

In the next section we will explain some properties of the oriented object model then we will present the relationship between ISO model and these properties.

2. Properties of design, of quality and their relation with ISO model

Functionality	Suitability	Allows drawing conclusions about the presence and appropriateness of a set of functions for specified tasks.
	Accurateness	The accuracy sub-characteristic allows drawing conclusions about how well software achieves correct or agreeable results.
	Interoperability	Measures the ability to interact with specified systems.
	Compliance	Measures the ability to adheres to application related standards, conventions, and regulations in laws and similar prescriptions
	Security	It measures the ability to prevent unauthorized access, whether accidental or deliberate, to programs or data.
Reliability	Maturity	Allows concluding about the frequency of failure by faults in the software.
	Fault tolerance	Allows to maintain a specified level of performance in cases of software faults or of infringement of its specified interface
	Recoverability	Attributes of software that relate to the capability to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort needed for it
Usability	Understandability	Attributes of software that relate to the users effort for recognizing the logical concept and its applicability.
	Learnability	Attributes of software that relate to the users effort for learning the application
	Operability	Attributes of software that relate to the users effort for operation and operation control
	Attractiveness	Attributes of software that relate to the capability of the software product to be attractive to the user.
Efficiency	Time Behaviour	Attribute software that allows concluding about the time behavior during testing or operating.
	Resource utilization	Attributes of software that relate to the amount of resources used and the duration of such use in performing its function.
Maintainability	Analyzability	Attributes of software that relate to the effort needed for diagnosis of causes of failures, or for identification of parts to be modified
	Changeability	Attributes of software that relate to the effort needed for modification, fault removal or for environmental change.
	Stability	Attributes of software that relate to the risk of unexpected effect of modifications.
	Testability	Attributes of software that relate to the effort needed for validating the modified software.
Portability	Adaptability	Measure attributes of software that allow concluding about the amount of changes needed for the adaptation of software to different specified environments.
	Installability	Attributes of software that relate to the effort needed to install the software in a specified environment
	Co-existence	The capability of the software to co-exist with other software in a common environment sharing common resources.
	Replaceability	Allows to draw conclusions about how well software can replace other software or parts of it

Table 1. The sub-characteristics of ISO model

2.1 Definition of properties used in classification

Design Size: measure the size of design elements, typically by counting the elements contained within.
Abstraction: is the manner that we can define a generic service.

Encapsulation: data hiding or data abstraction, the private members of a class, the encapsulated data can only be accessed by local methods.

Modularity: extends to which software is divided into components.

Coupling: is the degree of mutual interdependence between modules components.

Cohesion: is the degree of relationship between elements in a design unit (package, class etc.)

Composition: is to structure an object in parts (Jacobson).

Inheritance: is the process of creating new classes, called derived classes, from existing classes. The derived class inherits all the capabilities of the base class. In the mean time, new properties can be added.

Polymorphism: is the characteristic of being able to assign a different meanings or usage to something in different contexts.

Messaging: is the exchange of to a messaging server , which acts as a message exchange program for client programs

Complexity: measures the degree of connectivity between elements of a design unit.

2.2 Relationship between ISO characteristics and design properties

In our classification we are interested to the characteristics of the ISO model as much as combining them with object-oriented design and quality properties. In the following table we will explain this relationship.

3. Object-Oriented metrics and their classification by properties of design and quality

In this part we present some popular metrics with their classification using the properties already explained.

3.1 Metrics of Moreau and Dominick

The earliest researchers whose work in the Object-Oriented metrics were Moreau and Dominick[10] who defined three general metrics :

- **Message Vocabulary Size (MVS):** The number of different types of message sent by a particular object.
- **Inheritance Complexity (IC):** "*Compound"* and "*multilevel"* inheritance.
- **Message Domain Size (MDS):** The numbers of distinct procedures manipulate the state of object.

These metrics are not clear, such as what exactly is meant by "*sending messages*", and how the metrics are to be computed and they are not tested nor validated.

3.2 Moose metrics

The next metrics presented are those of Chidamber and Kemmerer introduced in the MOOSE (Metrics for Object Oriented Software Engineering) project [1], known as C.K. Many other OO metrics are built upon this metrics suite:

- Weighted Methods per Class: WMC measures the complexity of an individual class.

$$\mathrm{WMC} = \sum_{i=1}^{n} c_i$$

The sum of the complexity of each method contained in the class.

- **Depth of Inheritance Tree of a Class:** DIT is defined as the length of the longest path of inheritance ending at the current class. More the class is profound, more it was complex
- **Number of Children:** NOC represents the number of immediate subclasses subordinated to a class in question. A high value of NOC indicates an inappropriate abstraction in the design and high complexity.

characteristics	Sub- characteristics	Quality criteria	Design and quality properties
Functionality	Suitability	Ease to use (Check list)	
	Accurateness	user satisfaction (Check list)	
	Interoperability	Check list	
	Security	difficulty in accessing to the data	Abstraction, composition, coupling, inheritance, encapsulation
	Compliance	Quality of documentation (Naming, documentation)	
Reliability	Maturity	Modularity	Modularity
		Precision	Precision
		Consistency	consistency, cohesion
	Fault tolerance	Fault correlation	tolerance, instability, dependence
	Recoverability	memory efficiency	Complexity
		Execution efficiency	Complexity
		Response time	Complexity
Usability	Understandability	Documentation	
		Friendly interface (Check list)	
		Documentation quality	
		norms and standard (Naming)	
	Learnability	Quality of documentation	
	Attractiveness	unmarked data	Modularity
		unmarked communication	Modularity, messaging
Efficiency	Time behavior	Check list	
	Resource utilization	Check list	
Maintainability	analizability changeability	code architecture	coupling, inheritance, specialization of the class, Modularity
		size	Number of method, size method
		Documentation	
	Stability	Dependence interne	Dependence, stability, coupling
	Testability	code complexity	Complexity
		Modularity	Modularity
Portability	Adaptability	Documentation	
	Installability	auto-description	
	Conformance	Modularity	Modularity
	Replaceability	Machine independence	
		System independence	

Table 2. Relationship between ISO model and the properties of design and quality

• **Coupling between Objects:** CBO is defined as the count of the number of other classes to which it is coupled. A class is coupled to another class if it uses the member method and/or instance variables of the other class. A nonzero value of CBO indicates weakness of class encapsulation and may inhibit reuse.

• **Response for a Class:** RFC gives the number of methods that can be executed in response to a message received by an object of that class. If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated.

• **Lack of Cohesion in Method:** LCOM is the number of pairs of methods operating on disjoint sets of instance variables, reduced by the number of method pairs acting on at least one shared instance variable. For example, a class that has three instance variables (Ia , Ib ,Ic) and five methods (M1, M2,..., M5) where M1 and M2 only uses Ia, M3 and M4 only uses Ib and M5 uses Ia and Ic.This class will get a LCOM value of two (2).

Chidamber and Kemmerer metrics are focused on class level. Churcher and Shepperd [12] note that some definitions are imprecise. Such as the decision to count inherited methods as belonging to the class, whether to count methods with the same name (but different signatures), whether operators should be considered in the count, and so on.

In a reply to these remarks, Chidamber and Kemerer[13] clarify their position by stating that "*the methods that require additional design effort and are defined in the class should be counted, and those that do not should not*".

3.3 Metrics of Li and Henry

Li and Henry [2,3] present ten metrics in their system; five metrics of CK: DIT, NOC, RFC, LCOM, WMC, and they define five more metrics:

• Message-Passing Coupling (MPC) measures the complexity of message passing among classes. MPC is the (static) number of send statements defined in a class, where a send statement is a message sent out from a method in a class to a method in another class.

• Data Abstraction Coupling (DAC) is the number of abstract data types used as instance variables by a class.

• The Number of Methods (NOM) = number of local methods.

• The number of semicolons (SIZE1) in a class is a LOC traditional metric.

• Number of properties (SIZE2) is the number of attributes plus the number of local methods.

The DIT metric is used as a measure of complexity, more the value of DIT increases more the system is complex. Trying to minimize DIT (in order to decrease complexity) leads to the disuse of inheritance, which is one of the major advantages of the OO paradigm.

3.4 Metrics of Chen & Lu

Chen and Lu [14] present eight new metrics for OO design:

- **Operation Complexity (OpCom) of a class:** The OpCom is the sum of the complexity of each operation in the class. OpCom = ΣO(i) where O(i) is operation "*i*" complex value, and is evaluated from Table 3.
- **Operation Argument Complexity (OAC):** is the sum of the value of each argument in each operation in the class. OAC = ΣP(i) where P(i) is the value of each argument "*i*" in each operation in the class; it is evaluated from Table 4.
- **Attribute Complexity (AC) metric:** is the sum of the value of each attribute in the class. AC = ΣR(i) where R(i) is the value of each attribute in the class, and is also evaluated from Table 4
- **Operation Coupling (OpCpl) metric:** Measures the coupling between operations in the class and operations in other classes.
- **Class Coupling (ClCpl) metric:** Measures the coupling between a class and other classes
- **Cohesion (Coh):** Consider a class with N operations (function) : F(1), F(2), ..., F(N), with N sets of arguments I(1), I(2), ... , I(N); M is the number of disjoint sets of arguments formed by the intersection of these N sets. The cohesion metric is (M/N) *100%
- **Class Hierarchy (CH):** is the sum of the following:
    - The depth of the class in the inheritance tree.

- The number of sub-classes of the class.
- The number of direct super classes of the class.
- The number of local or inherited operations available to the class.

▪ **Reuse (Re):** It measures whether a class is a reused one. A value of 1 is given to the class if it is reused from the current or from another project and 0 otherwise.

About this suite metrics there is some ambiguity surrounding the difference between the "*operation coupling*" and "*class coupling*" metrics. The latter metric uses the number of accesses from a class to another; but it is not specified what constitutes a class access, thus the value of the class coupling metric is either 0 or 1 when taken between two classes; it does not take into consideration the number of messages passed between the class.

Rating	Complexity value
Null	0
Very low	1-10
Low	11-20
Nominal	21-40
High	41-60
Very high	61-80
Extra high	81-100

Table 3. Operation complexity value (from Chen and Lu [18])

Type	Value
Boolean or integer	1
Char	1
Real	2
Array	3-4
Pointer	5
Record, Struct, or Object	6-9
File	10

Table 4. Argument/attribute value (from Chen and Lu [18])

3.5 MOOD Metrics

Brito and Abreu[15] defined six metrics in the MOOD (Metrics for Object Oriented Design) project.

- **Method Hiding Factor (MHF):** is the sum of the invisibilities of all methods defined in all classes. The invisibility of a method is the percentage of the total classes from which this method is not visible.

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)}$$

The number of methods defined in class C_i

$$M_d(C_i) = M_v(C_i) + M_h(C_i)$$

$M_v(C_i)$ number of visible methods (interface) in class C_i

$M_h(C_i)$ number of hidden methods (implementation) in class C

- **Attribute Hiding Factor (AHF):** is the sum of the invisibilities of all attributes defined in all classes. The invisibility of an attribute is the percentage of the total classes from which this attribute is not visible.

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)}$$

the number of attributes defined in class C_i by :

$$A_d(C_i) = A_v(C_i) + A_h(C_i)$$

$A_v(C_i)$ number of visible methods (interface) in class

$A_h(C_i)$ number of hidden methods (implementation) in class C

• **Method Inheritance Factor (MIF):** is the sum of inherited methods in all classes of the system under consideration.

$$AHF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

The number of available method defined in class C_i by

$$M_a(C_i) = M_d(C_i) + M_i(C_i)$$

$M_d(C_i)$ is the number of the defined method

$M_i(C_i)$ is the number of inherited method

• **Attribute Inheritance Factor (AIF):** is the sum of inherited attributes in all classes of the system under consideration.

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

The number of available attribute defined in class Ci by:

$$A_a(C_i) = A_d(C_i) + A_i(C_i)$$

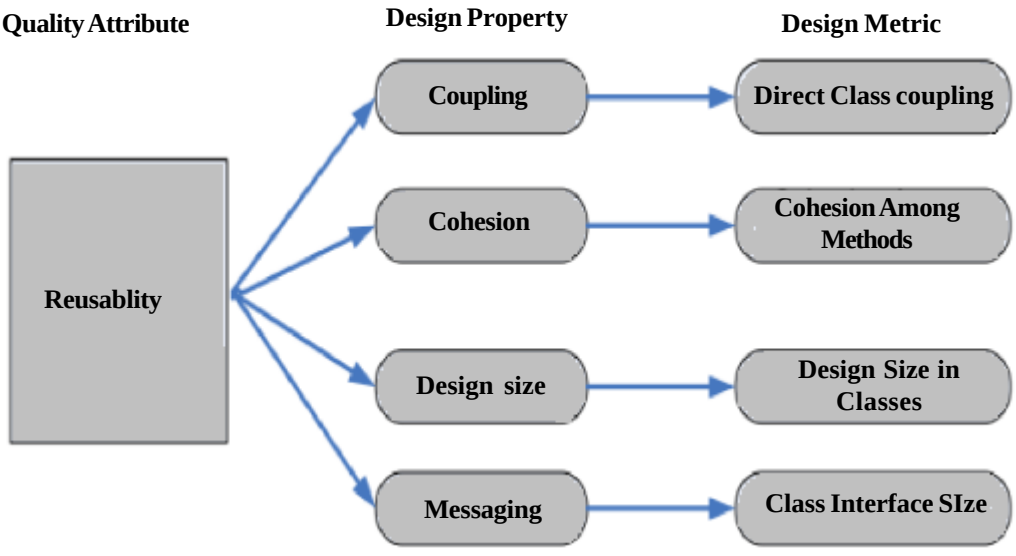

Figure 3. The design Metric of the reusability

Properties	Metrics
Design Size	NOM (Number Of Methods) SIZE1 (number of semicolons) SIZE2 (number of properties)
Abstraction	NOC (Number Of Children) DAC (Data Abstraction Coupling)
Encapsulation	AHF (Attribute Hiding Factor) MHF(Mathod Hiding Factor)
Inheritance	DIT (Depth of Inheritance Tree of a class) MIF (Method Inheritance Factor) AIF (Attribute Inheritance Factor) CH (Class Hierarchy)
Coupling	CBO (Coupling Between Object) COF (Coupling Factor) OpCpl (Operation Coupling) ClCpl (Class Coupling)
Cohesion	LCOM (Lack of Cohesion in Method) Coh (Cohesion)
Complexity	WMC (Weighted Methods per Class) NOC (Number Of Children) MPC (Message-Passing Coupling) OpCom (Operation Complexity) OAC (Operation Argument Complexity) AC (Attribute Complexity)
Messaging	RFC (Response For a Class) MPC (Message-Passing Coupling)
Polymorphism	POF (Polymorphism Factor)

Table 5. The classification of some popular metrics by properties of quality and design

$A_d(C_i)$ is the number of the defined attribute

$A_i(C_i)$ is the number of inherited attribute

- **Polymorphism Factor (POF):** represents the actual number of possible different polymorphic situations.
- **Coupling Factor (COF):** is the maximum possible number of couplings in a system.

This metrics are validated and applied to eight projects representing eight variations of the design for the same requirements document [9]. The results show that most of the metrics were good predictors of the quality measures: complexity, size and coupling.

3.6 QMOOD Metrics

The QMOOD [16] (Quality Model for Object-Oriented Design) project is a comprehensive quality model that establishes a clearly defined validated model to assess objects oriented design quality attributes, and relates it through mathematical formulas, with structural OOD. The QMOOD model consists of six equations that establish relationship between six OOD quality attributes (reusability, flexibility, understandability, functionality, extendibility, and effectiveness) and eleven design properties.

For example (see figure 3), reusability is a function of the coupling measure, cohesion measure, messaging measure, and the

design size. The coupling measure is quantified using Direct Class Coupling (DCC) metric. The cohesion measure is quantified using Cohesion Among Methods in a class (CAM) metric. The Messaging metric is quantified by Class Interface Size (CIS) metric. Finally, design size is quantified by Design Size in Classes (DSC) metric.

QMOOD distinguishes itself by providing mathematical formulas that links design quality attributes with design metrics. This allowed computing a Total Quality Index (TQI). Then the QMOOD suite is the most complete, comprehensive, and supported suite.

3.7 Classification of the metrics by properties of design

In the following table we present the classification of the metrics cited above by the properties of design already explained.

4. Conclusion

There are several metric models for software components, and many classifications are used to organize it, but in these classifications some quality attributes are unknown. It is unclear whether any fundamental of objectoriented, such as neglecting the use of inheritance to reduce complexity. To progress in this area, it is essential to rely on a model that allows classifying and systematizing metric use.

The work we are presenting in this paper is classified in this approach. It allows us to evaluate a software project taking as basis the ISO Model as much as taking in consideration the principal characteristics of the objectoriented design. For these reasons, we recovered properties concerning quality and object-oriented design from the characteristics of the ISO model. This work is our database that will be used to evaluate software projects in future works.

References

[1] Hoyer, R. W., Hoyer, B. B. Y. (2001). What is quality?.

[2] Chidamber, S. R., Kemerer, C. F. (1994). A Metrics Suite for Object Oriented Design, IEEE Transactions on Software Engineering, 20 (6), June.

[3] Li, W., Henry, S. (1993). Object-oriented metrics that predict maintainability. *Journal of Systems and Software*.

[4] Li, W., Henry, S., Kafura, D., Schulman, R. (1995). Measuring object-oriented design. *Journal of Object Oriented Programming*.

[5] ISO 8402. (1994). Quality management and quality assurance -Vocabulary.

[6] ISO/IEC 9126. (1991). Software product evaluation - Quality characteristics and guidelines for their use.

[7] ISO/IEC FCD 9126-1. (1998). Software product quality - Part 1: Quality model.

[8] Mohamed El-Wakil, Ali Fahmy. Object–Oriented Design Quality Models: A Survey and Comparison, IEEE.

[9] Jagdish Bansiya, Carl G. Davis. (2002). A Hierarchical Model for Object-Oriented Design Quality Assessment, IEEE, January, QMOOD.

[10] Brito, F., e Abreu, Melo, W. (1996). Evaluating the impact of object-oriented design on software quality. March, IEEE.

[11] Moreau, D. R., Dominick, W. D. (1989). Object-oriented graphical information systems : Research plan and evaluation metrics. *Journalof Systems and Software*.

[12] Hatton, L. (1998). Does OO sync with how we think? *IEEE Software*, 15 (3).

[13] Churcher, N. I., Shepperd, M. J. (1995). Comments on : a metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, March.

[14] Chidamber, S.R., Kemerer, C.F. (1995). Authors' reply to : comments on : A metrics suite for object oriented design. IEEE, March.

[15] Chen, J.-Y., Lu, J. F. (1993). A new metric for object-oriented design. *Information and Software Technology*, April.

[16] Brito e Abreu, F. (1992). Object-oriented software design metrics. *In*: Proc.of the OOPSLA' 92 workshop on OO metrics.

[17] Bansiya, J., Davis, C. (2002). A hierarchical model for object-oriented design quality assessment. *Transaction on Software Engineering*, IEEE.

[18] Fernando Brito e Abreu, Rogério Carapuça. Object-Oriented Software Engineering: Measuring and Controlling the Development Process.

[19] Chen, Lu, J. F. (1993). A new metric for object-oriented design. *Information and Software Technology*, 35 (4) 232-240, April.

[20] Hoyer, R. W., Hoyer, B. B. Y. (2001). What is quality?.

[21] Ronan Fitzpatrick. (1996). Software Quality: Definitions and Strategic Issues, April.