Towards a metric suite for OCL Expressions expressed within UML/OCL models

Luis Reynoso
Department of Computer Science, National University of Comahue
Neuquén, 8300, Argentina
and
Marcela Genero and Mario Piattini
Alarcos Research Group
Department of Computer Science, University of Castilla La Mancha
Ciudad Real, 13071, Spain

ABSTRACT

UML models quality is clearly a crucial issue that must be evaluated at the initial stages of object-oriented software development, in order to get software products with better quality. This fact is corroborated by the huge amount of metrics that have appeared in the literature related to the measurement of UML diagrams at a high level design stage. Most of these metrics are focused on the measurement of internal quality attributes such as structural complexity, coupling, size, etc. However, none of the proposed metrics take into account the added complexity involved when a UML model are complemented by expressions written in Object Constraint Language (OCL), that is a UML/OCL model. Due OCL is essential in building consistent and coherent platform-independent models we propose a metric suite for structural properties of OCL expressions. These metrics were proposed considering OCL concepts related to the "chunking" and "tracing" cognitive techniques. We believe that cognitive techniques affect the cognitive complexity, and by consequence the understandability and maintainability of expression of UML/OCL models. Therefore, the goal of this paper is to show how we defined these metrics in a methodological way, assuring thus their validity.

1. INTRODUCTION

The usage of metrics at the initial phases of the OO software life cycle can help modellers and designers to make better decisions, and to build OO software of a better quality, without unnecessary revisions at later development stages. Besides this fact, the early availability of metrics allows us to predict external quality attributes [22],[14], such as understandability and maintainability [17]. For that reason a huge amount of metrics were defined to be applied to any UML diagrams [1], [5], [9], [14], [18]. But, using UML alone for building a model [25], [27] is not enough to express many design decisions and constraints, because many essential aspect can not be specified using diagrammatic notations [15], [35]. However, after the introduction of OCL by OMG, the quality of a UML model can be improved specifying it in a combination of the UML and OCL languages, i.e., trough a UML/OCL combined model. Using a UML/OCL combined model the modeler can make decisions at a high level of abstraction producing a precise and complete model of a high level of maturity [35]. None of the proposed metrics take into account the added complexity involved when UML models are complemented by expressions written in OCL.

One of the key artefacts produced at the early development of OO software systems is the class diagram, because most of the work of design and implementation is based on it. For that reason, we started the definition of metrics for OCL expressions expressed within UML class diagrams. OCL enriches UML class diagrams with expressions that specifies semantic properties of the model [19] and improve the system precision, its documentation [34] and its understandability at early stages.

The theoretical basis for developing quantitative models relating to structural properties and external quality attributes has been provided by [9]. In this work we assume a similar representation to hold for OCL expressions. We implement the relationship between the structural properties on one hand, and external quality attributes on the other hand (see Fig. 1). We hypothesized that the structural properties (such as coupling, size, length, etc.) of an OCL expression have an impact on its cognitive complexity. By cognitive complexity we mean the mental burden of the persons who have to deal with the artefact (e.g. modellers, designers, maintainers). High cognitive complexity leads to an artefact reducing its understandability, and this conduce to undesirable external qualities, such as decreased maintainability. We suppose that OCL expression structural properties have an impact on the cognitive complexity of modellers, due to the fact that when developers try to understand an OCL expression, they apply cognitive techniques, such as "chunking" and "tracing" [11],[12],[16]. For that reason, it is important to define metrics related to these cognitive techniques referring to the structural properties of OCL expressions involved with them. Therefore, to accomplish this goal, the aim of this paper is two-fold:

1. Propose, in a methodological way, a set of metrics for measuring structural properties of OCL expressions, considering those OCL's concepts specified in its metamodel [26] related to the "chunking" and "tracing" cognitive techniques.
2. Assure that the proposed metrics measure what they purport to measure through their theoretical validation, following a property-based framework proposed by Briand et al. [6], [7], [8].

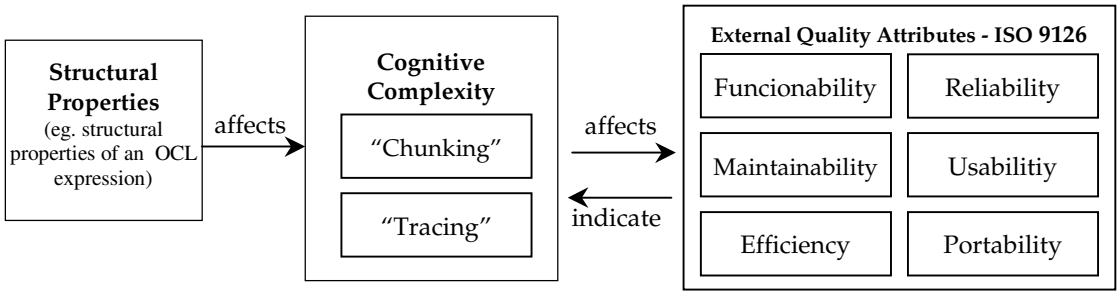


Fig 1: Relationship between structural properties, cognitive complexity, and external quality attributes (based on Briand et al [9] and ISO 9126 [22]).

This paper proceeds as follows. We start, in Section 2, describing some concepts of the cognitive model of Cant et al. [12], on which we are based on to define the metrics. Section 3 presents those OCL concepts related to "tracing" and "chunking". The proper definition of the metrics comes in section 4, meanwhile their theoretical validation is presented in section 5. Finally, in the last section some concluding remarks and future work are presented.

2. COGNITIVE TECHNIQUES OF CANT ET. AL

The Cognitive Complexity Model (CMM) defined by Cant et al. [12] gives a general cognitive theory of software complexity that elaborates on the impact of structure on understandability [16]. Although the study of Cant et al [12] has been considered a reasonable point of departure for understanding the impact of structural properties on understandability of code and the coding process, we believe that model can also be applied to UML developers when they try to understandand OCL expressions. The underlying rationale for the CMM argues that comprehension consists of two techniques: "chunking" and "tracing", that are concurrently and synergistically applied in problem solving [11], [12]:

- "Chunking" technique: a capacity of short term memory, involves recognizing groups of statements (not necessarily sequential) and extracting information from them which is remembered as a single mental abstraction: a "chunk" [12].
- "Tracing" technique: involves scanning, either forward or backwards, in order to identify relevant "chunks" [16], resolving some dependencies.

Cant et al. [12] argue that it is difficult to determine what constitutes a "chunk" since it is a product of semantic knowledge. Henderson-Sellers [20] notes that "tracing" disrupts the process of "chunking". The comprehension of a particular "chunk" is the sum of three components: (1) the difficulty of understanding the "chunk" itself; (2) the difficulty of understanding all the dependencies on the "chunks" upon which a particular "chunk" depends, and (3) the difficulty of "tracing" these dependencies to those "chunks" [16]. When a method calls for another method to be used in a different class, or when an inherited property needs to be understood [16], are typical examples of where "tracing" is applied. This cognitive process is also commonly done by UML developers during the understandability of OCL specifications. To understand an OCL expression, as a "chunk", UML developers must carry out "tracing" to understand some properties (attributes or operations) belonging to other classes, for example when a operation is referred through messaging, navigation of relationships, etc. These concepts will be explained in the following section.

3. OCL CONCEPTS RELATED TO COGNITIVE TECHNIQUES

OCL [26] is mainly used to add precision to UML element models beyond the capabilities of the graphical diagrams [19] on any UML model. Its main elements are OCL expressions that represent declarative and side effect-free textual descriptions that are associated to different features of UML diagrams [21], [26]. OCL expressions in UML class diagrams are used most importantly: to specify invariants on classes and types in the class diagram, to specify constraints on operations and methods, to describe pre- and post conditions on operations, to specify initial values and derivation rules for attributes, to specify query operations, and to introduce new attributes and operations [26].

An OCL expression is a suitable "chunk" unit, which modelers should understand as a whole declaration constraining an aspect of the system being modeled. Other kind of chunks are attributes, associations and operations which are accessed from an OCL expression.

The understanding of an OCL expression as a "chunk" involves a strong intertwining of "tracing" and "chunking" techniques. We had analyzed which OCL concepts, specified in its metamodel [26], are relevant to these cognitive techniques. The analysis of each of these techniques in turn leads us to identify the structural properties, which can be measured.

First we will briefly describe two important OCL concepts, that allow an expression to refers to model elements:

The Contextual Instance

Due to the fact that OCL is a typed language, it is important to notice that each OCL expression is written in the context of an instance of a specific type. The type, is referred inside an expression trough a reserved word, *self*, which represents the contextual instance.

Example 1. The following OCL expression states an invariant for the Company class. All companies must have a positive budget.

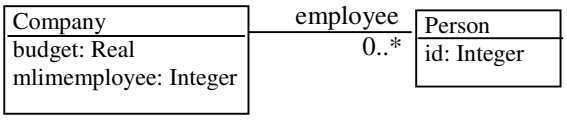
context Company
inv: self.budget > 0

Navigations

Starting from a specific object it is possible to navigate an association in the class diagram, to refer to other objects and their properties [26]. A relation is navigated when we use the rolename of the opposite association-end of a relation, that links the class where the expression is defined with another class in the diagram class (when the association-end is missing we can use the name of the type

at the association-end as the rolename). The result of a navigation is a single object or a collection of objects depending on the multiplicity of the association-end [31]. The syntax uses the dot notation followed by an association-end property. It is possible to navigate many relationships in order to access as many properties as needed in an expression.

Example 2. The following OCL expression, in the context of Company type, states that the number of employees must always be less than the maximal limit of employees of the company. The OCL expression is defined according to the UML class diagram of Fig. 2.



context Company
inv: self.mlimemployee > self.employee->size())

Fig 2: Example of Navigation

In the OCL expression self.employee represent a navigation from Company to Person class.
Cook et al [15] argues "that this kind of constraint on the multiplicity of an association occurs quite often in practice. The UML class model specifies multiplicity 'zero to many' in cases where the actual upper (or lower) bound of the multiplicity is not known beforehand, but depends on the value of an attribute of an object at runtime".

Example 3. An example of "chunking".
The contextual instance, i.e. *self*, provides a point of reference for the interpretation of an OCL expression. Whenever we use the self keyword, we are doing "chunking" of the expression.

Example 4. An example of "tracing".
If an OCL expression includes a navigation for accessing an attribute (or operation) belonging to a class, the proper evaluation of the navigation will disrupt the understanding of the OCL expression. The evaluation of the navigation, will involve the understanding of the rolenames (or class names if rolenames are missing in the diagram) mentioned in the navigation, to look for them in the class diagram, to look for the attribute (or operations) mentioned through the navigation, and eventually to understand the OCL expressions associated to this attribute (or operation), if they have OCL expressions attached.

In order to describe the OCL concepts which involve "chunking" we have basically considered those concepts which belong to one expression (the chunk) and which do not require solving dependencies to other chunks. Whilst to analyze "tracing", we have considered those OCL concepts that imply solving dependencies to other chunks. *Self* was used as the main concept related to the OCL expression itself, i.e. to the "chunking" technique. Other instances (object or object collections), whose types are different to the type represented by the contextual instance, are commonly accessed by the "tracing" techniques. Besides, two group of OCL concepts were defined for the "chunking" technique:

- "Chunking" Group 1 (CG1): this group of concepts includes OCL facilities related to the OCL language itself: variable definitions through *let* expressions, *if* expression condition, predefined iterator variables, literals, etc.
- "Chunking" Group 2 (CG2): this group of concept includes OCL concepts related to the contextual instance and some of its properties, values before the execution of an operation (that is, properties postfixed by @pre) of the contextual instance, variables defined through <<definition>> constraints in the type represented by the contextual instance, etc.

4. DEFINITION OF A METRIC SUITE FOR OCL EXPRESSIONS

In order to define valid and reliable metrics we have applied a method based on [10]; [13] which is composed of many steps beginning with the definition of the metrics goals and finishing with the acceptance of these metrics in real projects. Even though all the steps are equally important, in this paper we only address the definition of the metrics goals, the definition of the metrics and their theoretical validation. It is advisable to perform the theoretical validation of the metrics before the empirical validation. In the context of empirical studies, the theoretical validation of metrics demonstrate their construct validity. This means that the theoretical validation "proves" that they are valid measures to be used in empirical studies. The rest of the steps will be tackled in future works.

Fenton [17] suggested that it is not advisable to define a single measure for capturing different structural properties. For that reason we have defined a set of metrics, each of which captures different structural properties of an OCL expression, related to the cognitive techniques.

Using the GQM [2], [33] template for goal definition, the goal pursued for the definition of the metrics for OCL expression is shown in Table 1.

Table 1: Using GQM to define the metric's goal.

Analyse	OCL expression structures related to cognitive techniques.
for the purpose	of Evaluating
with respect to	their Understandability
from the point of view of	the OO Software modelers
in the context of	OO software organizations.

Even though our focus is on OCL expressions understandability, we will begin defining metrics for OCL expressions measuring their structural properties related to a cognitive technique, and afterwards ascertaining through experimentation if these metrics could be used as early understandability indicators.

Table 2 summarizes the metrics defined for OCL expression according to the cognitive technique they are related. A complete definition of the metrics of OCL expressions defined in terms of group 1 of "chunking", group 2 of "chunking", and "tracing" is included in [28], [29] and [30] respectively .

Table 2: Metrics for OCL expressions of UML/OCL models.

METRIC ACRONYM	METRIC GROUP	METRIC DESCRIPTION
NKW	"Chunking" CG1	Number of OCL KeyWords
NES	"Chunking" CG1	Number of Explicit *Self*
NIS	"Chunking" CG1	Number of Implicit *Self*
NVL	"Chunking" CG1	Number of Variables defined by *Let* expressions
NIE	"Chunking" CG1	Number of *If* Expressions
NSL, NOSL, NBL, NSQL, NTL	"Chunking" CG1	Number of *Set*, *OrderedSet, Bags*, *Sequence* or *Tuple* literals
NBO	"Chunking" CG1	Number of Boolean Operators
NCO	"Chunking" CG1	Number of Comparison Operators
NEI, NII	"Chunking" CG1	Number of Explicit/Implicit Iterator variables
NAS	"Chunking" CG2	Number of Attributes belonging to the classifier that Self represents
NOS	"Chunking" CG2	Number of Operations belonging to the classifier that Self represents
NVD	"Chunking" CG2	Number of Variables defined through <<Definition>> constraints
NIO	"Chunking" CG2	Number of oclIsTypeOf, oclIsKindOf or oclAsType Operations
N@P	"Chunking" CG2	Number of properties postfixed by @Pre
NNR	"Tracing"	Number of Navigated Relationships
NAN	"Tracing"	Number of Attributes referred through Navigations
WNON	"Tracing"	Weighted Number of referred Operations through Navigations
NNC	"Tracing"	Number of Navigated Classes
WNM	"Tracing"	Weighted Number of Messages
NPT	"Tracing"	Number of Parameters whose Types are classes defined in a class diagram
NUDTA	"Tracing"	Number of User-Defined Data Type Attributes
NUDTO	"Tracing"	Number of User-Defined Data Type Operations
WNN	"Tracing"	Weighted Number of Navigations
DN	"Tracing"	Depth of Navigations
WNCO	"Tracing"	Weighted Number of Collection Operations

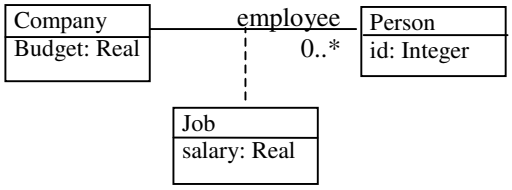
As an example of the defined metrics we show below a detailed definition of two metrics: NAN a defined metric related to the "tracing" technique, whereas NVL is a defined metric related to the "chunking" technique.

NAN Metric (Number of Attributes referred through Navigations)
DEFINITION: This metric counts the total number of attributes referred through navigations in an expression.
GOAL: NAN measures the extent of usage of attributes of other classes by the class where the expression is defined. The larger the set of attributes referred through navigations, the greater is the context to be understood. The understanding of attributes belonging to other classes involves the comprehension of them as a chunk, i.e. their meaning and the OCL specification associated to them.
EXAMPLE: In the expression of example 2 only one attribute is accessed by a navigation, the salary attribute is used, thus NAN = 1.

NVL Metric (Number of Variables defined by Let expressions)
DEFINITION: This metric counts the total number of variables defined by *Let* expressions in an expression. This metric does not take into account the number of times a variable is reused, but the quantity of different defined variables.

GOAL: NVL metric is related to the degree of reuse of the variables within an OCL expression. Although a low use of let expressions can improve the readability of an OCL expression, we believe that a higher number of variables defined through let can indicate that the OCL expression is reasonably complex.

EXAMPLE: In the OCL expression of fig. 3 a variable called *income* is defined to represent the expression: self.job.salary->sum(). The value of NVL is 1, as there is only one variable defined through a let-expression.



context Person inv:
let income : Integer = self.job.salary-> sum() in
if isUnemployed then income < 100
else income >= 100 endif

Fig 3: Example of Let expression.

5. THEORETICAL VALIDATION OF THE PROPOSED METRICS

Validating a software measure is the "process of ensuring that the measure is a proper numerical characterization of the claimed attribute by showing that the representation is satisfied" [17]. As we previously mentioned, the theoretical validation of metrics establishes their construct validity, i.e. it "proves" that they are valid measures for the constructs that are used as variables in the study.
To develop the theoretical validation of metrics we have used the property-based framework of Briand et al. Briand et al. ([7], [8]) and its adaptation for interaction-based metrics for coupling and cohesion [6].
Due to sake of brevity we will only show one example of the theoretical validation for each kind of measure. The theoretical validation of the rest of the metrics is summarized in Table 3.

NAN Properties as a Coupling (interaction-based) Metric
We will make some definitions prior to the application of properties of interaction-based metrics for coupling to the NAN (The number of attributes referred through navigations in an expression) metric:

- Relation: The relations are defined between a software individual part (in our context, an OCL expression) and its associated software system (attributes which it is possible to access through navigations in the NAN metric).
- DU-interaction: The interaction from Data declaration to Data Used (attributes used through navigations) in an OCL expression.
- Import coupling: Given a software part *sp* (an OCL expression), import coupling of *sp* is the number of DU-interactions between data declaration external to sp and data used within *sp*.

Our hypothesis is similar to the ISP-hypothesis of Briand et al. [6]: The larger the number of "used" software parts, the larger the context to be understood, the more likely the occurrence of a fault.

Following a similar approach applied in [6] the properties for interaction-based measures for coupling are instantiations, for our specific OCL context, of the properties defined in Briand et al. [7], [8] for coupling.

- Nonnegativity: Is directly proven, and it is impossible to obtain a negative value. An expression sp without navigation (referring to attributes) in its definition has NAN(sp) = 0.
- Monotonicity: Is directly verified, adding import interactions - in this case, DU-interactions of navigations referring to attributes- to an OCL expression cannot decrease its import coupling. If we add a new navigation referring to an attribute in an expression *sp*, two possible situations can happen: (1) the attribute referred to in the added navigation is an attribute already used by a DU-interaction. Thus the metric NAN applied to the new expression obtained, is equal to NAN(*sp*). (2) If the added navigation refers to a new attribute, then NAN applied to the new expression is greater than NAN (*sp*).
- Merging of Modules: This property can be expressed for our context in the following way: "the sum of the import coupling of two modules is no less than the coupling of the module which is composed of the data used of the two modules". The value of NAN for an expression which consists of the union of two original expressions, is equal to the NAN of each expression merged when the sets of attributes referred to in each original expression are disjointed, otherwise it is less than NAN of each expression merged.

In a similar way, it is possible to show that NNR, WNON, NNC, WNM, NPT, NUDTA, NUDTO, NAS, NOS, N@P and NIO are interaction-based measures for coupling.

NVL as a Size Metric
For our purpose and in accordance with the framework of Briand et al. [7], [8], we consider that an OCL expression is a system composed of variables defined through let expressions (the elements) and relationships are represented by the relation "belong to", which reflects that a variable of this type belongs to an OCL expression. A sub-expression will be considered a module. We will demonstrate that NVL fulfils all of the axioms that characterize size metrics, as follows:

- Nonnegativity: This property is directly proven because it is impossible to obtain a negative value.
- Null value: An expression *e* without a variable defined through let expressions, has a NVL(*e*) = 0
- Module Additivity: If we consider that an OCL expression is composed of modules with no variables (representing the same subexpression and defined through let expressions) in common, the number of variables of an OCL expression will always be the sum of the number of variables of its modules. In other words, when two modules without variables in common are merged then the new expression has as many variables as each of the merged expressions. But if the original merged modules (sub-expressions) have some variable (representing the same subexpression and defined through let expressions) in common, then the NVL of the resulting expression should be less than adding the NVL of the original expressions.

In a similar way, it is possible to show that WNN, WNCO, NKW, NES, NIS, NIE, NSL, NOSL, NBL, NSQL, NTL, NBO, NCO, NEI, NII, WNM and NVD are size metrics.

DN as a Length Metric
For our purpose and in accordance with the framework of Briand et al. [7], [8], we consider that an OCL expression is a system. The elements are the classes (to which the expression navigates) and the relationships are the navigations of a UML relationship. We will demonstrate that DN fulfils all of the axioms that characterize length metrics, as follows:

- Nonnegativity and Null Value are straightforwardly satisfied, the depth of a tree can never be negative, and an expression without navigation has an empty tree, and then DN is 0.
- Nonincreasing monotonocity for connected components: If we add relationships between elements of a tree (classes or interfaces) the depth of the tree does not vary.
- Nondecreasing monotonocity for non- connected components: Adding a relationship to two unconnected components (two trees) makes them connected, and its length is not less than the length of the two unconnected components.

· Disjoint modules: The depth of a tree is given by the component that has more levels from the root to the leaves.

6. CONCLUSIONS AND FUTURE WORK

This paper shows how we defined a metric suite for measuring structural properties of OCL expressions, considering OCL concepts related to the cognitive techniques of "tracing" and "chunking".

After performing the theoretical validation of the proposed metrics using the original framework of Briand et al., [7], [8] and its adaptation to interaction-based metrics [6], the results are summarized in table 3.

As Table 3 shows most of the metrics related to the "chunking" technique are size metrics, and most of the metrics defined the "tracing" technique are interaction-based metrics for coupling.

Table 3. Theoretical validation of metrics according to Briand et al. [6], [7] and [8]

Metric Classification	OCL expression metrics defined in terms of cognitive techniques					
	"Chunking" 1	"Chunking" 2		"Tracing"		
	NKW, NES, NIS, NVL, NIE, NSL, NOSL, NBL, NSQL, NTL, NBO, NCO, NEI, NII	NAS, NOS, NIO, N@P	NVD	NNR, NAN, WNON, NNC, WNM, NPT, NUDTA, NUDTO	WNM WNN WNCO	DN
Interaction-based metrics for coupling		X		X		
Length						X
Size	X		X		X	

This correlation between a cognitive technique and some kinds of metrics for internal attributes is significant, and the reason is because of the proper definition of the cognitive technique. In fact, Klemola [24] argues that "some traditional complexity metrics can be supported by the fact they are clearly related to cognitive limitations".

We believe that the metrics obtained for "tracing" technique will be more relevant than measures obtained for "chunking" technique, as the former cognitive technique has been observed as a fundamental activity in program comprehension [4], [24].

We are aware that it is necessary to provide information about the utility of the metrics in practice, through they empirical validation. As many authors mentioned [3]; [17]; [23], [32] the empirical validation employing experiments or case studies is fundamental to assure that the metrics are really significant and useful in practice. We are currently planning a controlled experiment for corroborating if the proposed metrics could be useful as early indicators of the OCL expression understandability. Moreover, once we have empirically validated these metrics at an expression level, we will be able to extend them at a class level.

7. ACKNOWLEDGMENTS

This work has been partially funded by the MUML project financed by "University of Castilla-La Mancha" (011.100623), the network VII-J-RITOS2 financed by CYTED, and the GIISCo research group (UNComa 04/E048 project) financed by "Subsecretaría de Investigación de la Universidad Nacional del Comahue". Luis Reynoso enjoys a postgraduate grant from the agreement between the Government of Neuquén Province (Argentina) and YPF-Repsol.

8. REFERENCES

[1]. J. Bansiya and C. G. Davis. "A Hierarchical Model for Object-Oriented Design Quality Assessment", IEEE Transactions on Software Engineering, 28(1), enero, 2002, pp.4-17.

[2]. V. R. Basili, and H. Rombach, (1998). The TAME project: towards improvement-oriented software environments. IEEE Transactions on Software Engineering. Vol. 14. Nº 6, pp. 758-773.

[3]. V. Basili, F. Shull and F. Lanubile. (1999). Building knowledge through families of experiments. IEEE Transactions on Software Engineering, Vol. 25. Nº 4, pp. 435-437.

[4]. D. A. Boehm-Davis, J. E. Fox, B. and Philips, (1996). Techniques for Exploring Program Comprehension. Empirical Studies of Programmers, Sixth Workshop. Eds. W. Gray and D. Boehm-Davis. Norwood, NJ: Ablex, pp. 3-21.

[5]. L. C. Briand, S. Arisholm, F. Counsell, F. Houdek y P. Thévenod-Fosse. "Empirical Studies of Object-Oriented Artefacts, Methods, and Processes: State of the Art and Future Directions", Empirical Software Engineering, 4(4), diciembre, 1999, pp. 387-404.

[6]. L. C. Briand, S. Morasca, and V. Basili, (1999). Defining and validating measures for object-based high level design. IEEE Transactions on Software Engineering. Vol. 25. Nº 5, pp. 722-743.

[7]. L. C. Briand, S. Morasca, and V. Basili, (1996). Property-based software engineering measurement. IEEE Transactions on Software Engineering, Vol. 22 . Nº 1, pp. 68-86.

[8]. L. C. Briand, S. Morasca and V. Basili, (1997). Response to: comments 'Property-Based Software Engineering Measurement': Refining the additivity properties. IEEE Transactions on Software Engineering, Vol. 22. Nº 3, pp. 196-197.

[9]. L. C. Briand. and J. Wüst (2001). Modeling Development Effort in Object-Oriented Systems Using Design Properties. IEEE Transactions on Software Engineering, Vol. 27. Nº 11, pp.963-986.

[10].C. Calero, M. Piattini, and M. Genero, (2001). Method for obtaining correct metrics. Proc. of the 3rd International Conference on Enterprise and Information Systems (ICEIS` 2001), pp. 779-784.

[11].S. N. Cant, B. Henderson-Sellers, and D. R. Jeffery (1994). Application of Cognitive Complexity Metrics to Object-Oriented Programs. Journal of Object-Oriented Programming, Vol. 7. Nº 4, pp. 52-63.

[12].S. N. Cant, D. R. Jeffery and B. Henderson-Seller, B. (1992) A Conceptual Model of Cognitive Complexity of Elements of the Programming Process. Information and Software Technology, Vol. 7, pp. 351-362.

[13].G. Cantone, and P. Donzelli, (2000). Production and maintenance of software measurement models. Journal of Software Engineering and Knowledge Engineering, Vol. 5, pp. 605-626.

[14].D. Card, K. El-Emam and B. Scalzo, "Measurement of Object-Oriented Software Development Projects", Software Productivity Consortium NFP, 2001.

[15].S. Cook, A. Kleepe, R. Mitchell, B. Rumpe, J. Warmer and A. Wills (2001). The Amsterdam Manifiesto on OCL. Tony Clark and Jos Warmer, editors, Advances in Object Modelling with the OCL, pp. 115-149. Springer, Berlin, LNCS 2263.

[16].K. El-Eman, (2001). Object-Oriented Metrics: A Review of Theory and Practice. National Research Council Canada. Institute for Information Technology.

[17].N. Fenton, and S. Pfleeger, (1997). Software Metrics: A Rigorous and Practical Approach. Chapman & Hall, London, 2nd Edition. International Thomson Publishing Inc.

[18].M. Genero, "Defining and Validating Metrics for Conceptual Models", PhD Thesis, Universidad de Castilla-La Mancha, 2002.

[19].M. Gogolla, and M. Richters (2001). Expressing UML Class Diagrams properties with OCL. Tony Clark and Jos Warmer, editors, Advances in Object Modelling with the OCL, pp. 85-114. Springer, Berlin, LNCS 2263.

[20].B. Henderson-Sellers (1996). Object-Oriented Metrics: Measures of Complexity. Prentice-Hall.

[21].R. Hennicker, H. Hussmann, and M. Bidoit (2001). On the Precise Meaning of OCL Constraints. Tony Clark and Jos Warmer, editors, Advances in Object Modelling with the OCL, pp. 69-84. Springer, Berlin, LNCS 2263.

[22].ISO/IEC 9126 (2001). Software Product Evaluation-Quality Characteristics and Guidelines for their Use. Geneva.

[23].B. Kitchenham, S. Pflegger, and N. Fenton, (1995). Towards a Framework for Software Measurement Validation. IEEE Transactions of Software Engineering, Vol. 21. Nº 12, pp. 929-943.

[24].T. Klemola, (2000). A cognitive model for complexity metrics. 4th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering. Sophia Antipolis and Cannes, France.

[25].Object Management Group. (2003a). UML 2.0 Infrastructure Final Adopted Specification. OMG Document ptc/03-09-15. [On-line] Available http://www.omg.org/cgi-bin/doc?ptc/2003-09-15.

[26].Object Management Group. (2003b). UML 2.0 OCL 2nd revised submission. OMG Document ad/2003-01-07. [On-line] Available: http://www.omg.org/cgi-bin/doc?ad/2003-01-07.

[27].Object Management Group. (2003c). UML Specification Version 1.5, OMG Document formal/03-03-01. [On-line] Available: http://www.omg.org/cgi-bin/doc?formal/03-03-01

[28].L. Reynoso, M. Genero and M. Piattini, (2003a). Métricas para expresiones OCL relacionadas con la técnica cognitiva de "chunking". CLEI 2003. La Paz, Bolivia.

[29].L. Reynoso, M. Genero and M. Piattini, (2003b). Métricas para propiedades estructurales de expresiones OCL relacionadas con la técnica de "chunking". CACIC 2003. La Plata, Argentina.

[30].L. Reynoso, M. Genero and M. Piattini, (2003c). Measuring OCL expressions: a "tracing"-based approach. Workshop on Quantitative Approaches in Object-Oriented Software Engineering. QAOOSE' 2003. Alemania.

[31].M. Richters. (2002). A Precise Approach to Validating UML Models and OCL Constraints. Biss Monographs Vol. 14. Gogolla, M., Kreowski, H.J., Krieg-Brückner, B., Peleska, J., Schlingloff, B.H. (series editors). Logos Verlag. Berlin.

[32].N. Schneidewind, (1992). Methodology For Validating Software Metrics. IEEE Transactions of Software Engineering, Vol. 18. Nº 5, pp. 410-422.

[33].R. Van Solingen, and E. Berghout, (1999). The Goal/Question/Metric Method: A practical guide for quality improvement of software development. McGraw-Hill.

[34].J. Warmer, and A. Kleppe, (1999). The Object Constraint Language. Precise Modeling with UML. Object Technology Series. Addison-Wesley. Massachusetts.

[35].J. Warmer and A. Kleppe (2003). The Object Constraint Language. Second Edition. Getting Your Models Ready for MDA. Object Technology Series. Addison-Wesley. Massachusetts.