Measuring OCL expressions: a "tracing"-based approach

Luis Reynoso, Marcela Genero, and Mario Piattini

Abstract— Owing that class diagrams constitute the backbone of object-oriented (OO) software development, many metrics were proposed to measure internal quality attributes such as the structural complexity, coupling, size, etc. But none of the proposed metrics take into account the added complexity when class diagrams are complemented by Object Constraint Language (OCL) expressions. OCL expressions improve class diagrams enhancing their semantic properties, adding precision, improving their documentation and understandability. The importance of OCL and the lack of defined metrics for OCL expressions motivate us to propose a set of metrics for structural properties of OCL expressions. The first set of metrics we propose considers only those OCL concepts related to "tracing" technique. We believe that "tracing" technique affects the cognitive complexity, and by consequence the understandability of OCL expressions, and the maintenance of a UML class diagram. Therefore, the main goal of this paper is to show how we defined a set of metrics for structural properties of OCL expressions in a methodological way. We will also present the theoretical validation of these metrics according to a property-based framework proposed by Briand et. al..

Index Terms—Software measurement, OO measures, measure properties, cognitive complexity, tracing technique, understandability, maintainability, coupling.

1 Introduction

IN software engineering it is widely acknowledged that 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. This being when changes are more expensive and also more difficult to perform[1], [4], [9], [11], [17], [22], [32]. Besides this fact, the early availability of metrics allows us to predict external quality attributes [12],[27], such as understandability and maintainability [21]. 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. Hence class diagram quality is a crucial issue that must be evaluated (and improved if necessary) in order to get quality OO software. Many measures have been provided in literature that can be applied to a UML [34] class diagram at a high level design stage [5],[6],[18],[19],[23],[24],[26]. Most of these works 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 when class diagrams are complemented by OCL expressions. OCL, defined by OMG [31], has became a fundamental language in developing OO software using UML, because it allows a precise UML modelling, providing the modeller with an expressive notation to capture the essential properties of the systems [30]. OCL enriches UML class diagrams with expressions that specifie semantic properties of the model [26] and improve the system precision, its documentation [36] and its understandability at early stages of OO software development. This led us to think about the necessity of having metrics to measure structural properties of OCL expressions.

The theoretical basis for developing quantitative models relating to structural properties and external quality attributes has been provided by Briand et al [10]. In this work we assume a similar representation to hold for OCL expressions. We implement the relationship between the structural complexity 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" [14],[15],[20]. For that reason, it is important to study and 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 only those OCL's

- *Luis Reynoso is with the Department of Computer Science, National Unversity of Comahue, Neuquen, 8300, Argentina. E-mail: lreynoso@uncoma.edu.ar.*
- *Marcela Genero and Mario Piattini are with the Department of Computer Science, Castilla La Mancha University, Ciudad Real, España. E-mail: {Marcela.Genero, Mario.Piattini@uclm.es}.*

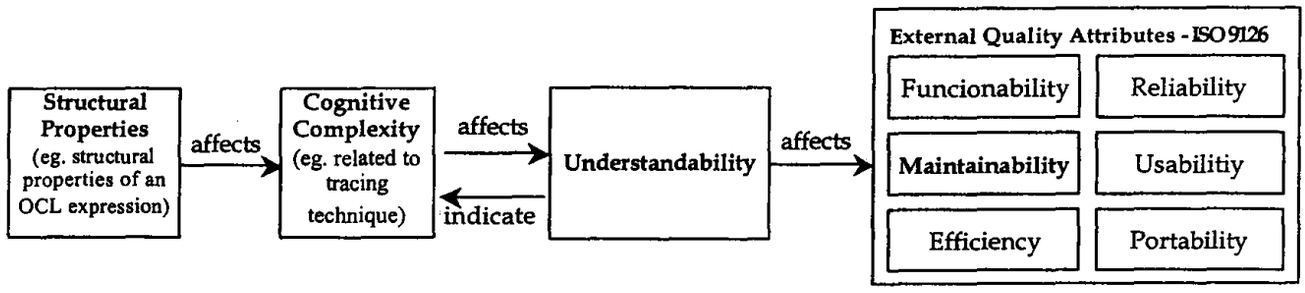


Fig 1: Relationship between structural properties, cognitive complexity, understandability and external quality attributes (based on [9] and [27]).

concepts specified in its metamodel [31] related to "tracing" technique.

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].

To obtain the first goal we start in the following section to describe some concepts of the cognitive model of Cant et al. [15], on which we are based, to define the metrics. Section 3 presents those OCL concepts related to "tracing". The proper definition of the metrics is presented 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 [15]

The Cognitive Complexity Model (CMM) defined by Cant et al. [15] gives a general cognitive theory of software complexity that elaborates on the impact of structure on understandability [20]. Although the study of Cant et al. 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 [15],[20]:

- "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" [15].
- "Tracing" technique: involves scanning, either forward or backwards, in order to identify relevant "chunks" [20], resolving some dependencies.

Cant et al. [15] argue that it is difficult to determine what constitutes a "chunk" since it is a product of semantic knowledge. For our purposes we will consider an OCL expression as a "chunk" unit, whilst also the comprehension of an operation, an attribute or a relationship with their associated OCL expression are considered to be a "chunk". Henderson-Sellers 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" [20]. When a method calls for another method to be used in a different class, or when an inherited property needs to be understood [20], 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 and its concepts related to Tracing"

OCL is mainly used to add precision to UML models beyond the capabilities of the graphical diagrams [30] on any UML model [36]. However, we will focus on UML class diagrams where the metrics will be applied. OCL can be used for different purposes: to specify invariants on classes and types in the class diagram, to describe pre- and post conditions on operations, to specify target (sets) for messages, to specify constraints on operations, etc. [31].

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 [31],[36]. The metrics we will define are mainly related to those OCL concepts related to "tracing" techniques which allows an expression to use properties belonging to other classes or interfaces, different to the type of the contextual instance, such as:

- Navigations: Starting from a specific object it is possible to navigate an association on the class diagram, to refer to other objects and their properties. 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). It is possible to navigate many

relationships to access as many properties as needed in an expression.

- Collections: Manipulating and dealing with collections is an important characteristic in OO software. OCL provides three concrete types of collections: Set, Bag, and Sequence [31].
- In, Out and In/Out Parameters, and Return Values: Operations may have *in, out, in/out* parameters. If the operation has *out* or *in/out* parameters, the result of this operation is a tuple containing all *out, in/out* parameters and the return value [31].
- Collection Operations: OCL defines many operations to handle the elements in a collection. The operations allow the modeller to project new collections from the existing one. Operations like select, reject, iterate, forAll and exists, take each element in a collection and evaluate an expression on them. The expression evaluated for each collection can be defined in terms of new navigations. We will take into account those expressions of collection operations which are defined in terms of other navigations.
- Messages: "OCL Message expressions are used to specify the fact that an object has, or will sent some message to another object at some moment in time" [29], [31].

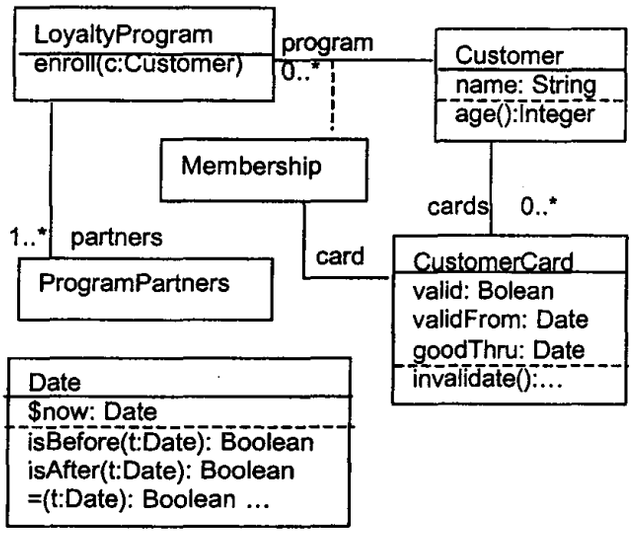
Finally, we will define an utility class, which is a UML concept, but it can be used inside an expression definition:

- Utility class: A utility class is a value type defined as a new type in a class diagram [34],[36].

For illustrating some of the concepts defined in this section, and the metric calculation, we use the example shown in Fig. 2. This example is a small part of the Royal and Loyal (R&L) example proposed in [36].

Example. R&L company handles loyalty programs for its Programs Partners that offer their customers various kinds of facilities. The LoyaltyProgram class administers a single loyalty program[36]. The objects of class Customer represent the people who have entered the program. A membershipCard or CustomerCard is issued to one customer. The Fig. 2 also includes in its bottom part, five OCL expressions. For example:

- Self^invalidate() of expression 1, is an example of a message sent to an object representing the contextual instance.
- Membership.card of expression 2 is an example of a navigation from LoyaltyProgram to CustomerCard. It navigates two relationships: one from LoyaltyProgram to Membership (an association class), and another from Membership to CustomerClass. In the former relationship there is no rolename attached to the association-end where Membership is the sink class, and for that reason the name of the class is used, starting with a lowercase letter. Meanwhile, in the latter relationship, the navigation is represented by its rolename "card".
- Self.customer of expression 2 is another example of a navigation where a forAll collection operation is specified to express that all the ages of the customers of a LoyaltyProgram should be more than 30 years old.
- Date is an example of a utility class, their attributes and operations are used in expression 1.




```
(expression 1)
  CustomerCard inv:
  validFrom.isBefore(goodThru) or
  goodThru.isAfter(Date.now)implies self^invalidate()

(expression 2)
  LoyaltyProgram inv:
  membership.card -> forAll (
              goodThru = Date.fromYMD  (2007,1, 1)
  and self.customer->forAll (age()>30)

(expressions 3 and 4 (pre- and post-cond. respectively))
  LoyaltyProgram::enroll(c: Customer)
  pre: not customer->includes( c )
  post: customer = customer @ pre-> including ( c)

(expression 5 )
  LoyaltyProgram inv:
  self.customer ->forAll( age() <= 30) and
  self.customer ->forAll (c1 | self.customer ->
    forAll (c2 | c1 <> c2 implies c1.name <> c2.name )
```

Fig. 2: Examples of OCL expressions [36].

4 METRICS DEFINITION FOR OCL EXPRESSIONS

In order to define valid and reliable metrics we have applied a method based on [13], [16], 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 an empirical study, the theoretical validation of metrics demostrate their construct

validity, i.e. it "proves" that they are valid measures to be used as variables in the empirical study. The rest of the steps will be tackled in future works.

Fenton [21] 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 a "tracing" technique.

Using the GQM ([2],[35]) template for goal definition, the goal pursued for the definition of the metrics for OCL expression is:

> Analyse *OCL expression structures related to a "tracing" technique*
> for the purpose of *Evaluating*
> with respect to *their Understandability*
> from the point of view of *the OO Software modellers*
> in the context of *OO software organizations.*

Thus, our goal is to begin defining a set of metrics for OCL expressions measuring their structural properties related to a "tracing" technique, and afterwards ascertaining through experimentation if these metrics could be used as early understandability indicators. Next, we will present each of the metrics we have defined to be applied at OCL expression level.

- **NNR Metric** (Number of Navigated Relationships): This metric counts the total number of navigated relationships in an expression. If a relationship is navigated twice, for example using different properties of a class or interface, this relationship is counted only once.
  Whenever we navigate to an association class we will consider the association to which the association class is attached.

  Example. In the expression 1 of Fig. 2., the value of NNR = 0 because there is no navigation in the expression definition, however in the expression 2, NNR = 2 because two relationship where navigated: the relationship between LoyaltyProgram and Customer (it was navigated from LoyaltyProgram to Membership, and from LoyaltyProgram to Customer), the relationship between Membership and CustomerCard.

- **NAN Metric** (Number of Attributes referred through Navigations): The number of attributes referred through navigations in an expression.

  Example. In the expression 2 of Fig. 2, only the goodThru attribute is used, then NAN = 1.

- **WNO Metric** (Weighted Number of referred Operations through navigations): The metric is defined as the sum of weighted operations (operations which are referred through navigations). For that reason we must consider the operation calls. The operations are weighted by the number of actual parameters (only the values of all *in* or *in/out* parameters are necessary to specify in the operation call [31]) and the number of *out* parameters acceded (also considering the operation return type or result). The definition of the WNO metric is defined as it is shown in Table 1:

Table 1: WNO metric definition.

$\sum_{m \in N(expression)} (1 + \lvert Par(m) \rvert)(1 + R + \lvert P_{out,\ in/out}(m) \rvert)$
N(expression): Set of different[1] operations referred through navigations.
$\lvert Par(m) \rvert$: quantity of actual parameter of *m*
R stand for the *m* operation result and represents the value of 1.
$\lvert P_{out,\ in/out}(m) \rvert$: quantity of *out* and *in/out* parameter of the *m* operation. Through a navigation it is possible to exclusively access one result as a time [31], this weighted formula giving a higher weight to those operations used many times to access different results.

  Example. According to the operation called "income" (of [31]) and its post-condition OCL.
  context Person::income(d: Date, bonus: Integer): Integer
  post: result = type { bonus =...., result =....}
  The operation has a result of type Integer, an in parameter (d) and an out parameter (bonus). Now, consider an expression in which we navigate to the Person class, and we operates with the two returned values of income:
  context Salary::calculate()
  post: person.income(aDate).bonus + person.income(aDate).result
  Applying the WNO to the postcondition expression of the calculate operations we obtain: WNO = (1 + 1) (1 + 1 + 1).

- **NNC Metric** (Number of Navigated Classes): This metric counts the total number of classes, association classes or interfaces to which an expression navigates to. If a class contains a reflexive relation and an expression navigates it, the class will be considered only one time in the metric. Also, as a class might be reachable from a starting class/interface from different forms of navigations (ie. following different relationships) we must consider this situation as a special case: If a class is used in two (or more) different navigations the class is counted only one time.

  Example. In the expression 2 of Fig. 2., the value of NNC = 3, because the classes Membership, Customer and CustomerCard are used.

- **WNM Metric** (Weighted Number of Messages): The number of messages defined in an expression weighted by its actual parameters. The weighted operation is carried out according to Table 2.

  Example. If we apply WNM metric to the expression 1 of Fig. 2, WNM = 1, because there is only one

[1] We will consider the actual parameter of the operation call [31] in order to analyze if two operations (referred through navigations) are different elements in the N(expression)-set. Having two operation calls they will be different if they have different operation name or different actual parameters (their parameter names or/and the quantity of actual parameter are not equal).

message (without parameters) in the expression.

Table 2: WNM metric definition.

$\sum_{m \in M(expression)} (1 + \lvert Par(m) \rvert)$
M(expression): Set of different operations[2] used through messaging in an expression.
$\lvert Par(m) \rvert$: quantity of actual parameter of the m operation.

- **NPT Metric** (Number of Parameters whose Types are classes defined in the class diagram): This metric is specially used in pre- and post-condition expressions, and it counts the method parameters, and the return type (also called result) used in an expression, having each parameter/result a type representing a class or interface defined in the class diagram.

  Example. In the expression 3 and 4 of Fig. 2., the value of NPT = 1 because only one parameter (c), whose type is a class in the class diagram (Customer), is used in the expression.

- **NUCA Metric** (Number of Utility Class Attributes used). The number of attributes belonging to a utility class used in an expression. Attributes are counted once if they belong to the same utility class and are also used more than once.

  Example. In the expression 1 of Fig. 2, the value of NUCA = 1 because only one attribute (now) of a utility class (Date) is used.

- **NUCO Metric** (Number of Utility Class Operations used). The definition of this metric is analogous to the NUCA metric, but considering operations instead of attributes.

  Example. In the expression 1 of Fig. 2, NUCO = 2 because the isBefore and isAfter operations (belonging to the utility class Date) are used.

- **WNN Metric** (Weighted Number of Navigations): As we explain in the previous section, an operation collection is composed of an expression which is evaluated for each collection element, and if the evaluated expression involves a new navigation (or many) we will give a higher weight to the new navigation used inside the definition of the outermost expression.

  Example. In the expression 3 of Fig. 2, the value of WNN is 1, and there is only one navigation (self.customer). Now, we will show how the WNN is obtained in the expression 5 of Fig. 2. Two subexpressions are connected by an "and" operator. Each subexpression involves navigations. Meanwhile the navigation of the first subexpression does not include a new navigation in its evaluation, the second one (of the last subexpression) uses a collection operation defined in terms of another, and the value of WNN is obtained in the following way: $1 * 2 + 2 * 1 = 4$.

  The number shown in italics represents the applied weight, and the number shown in normal font indicates the number of navigations. As the collection operation can be defined in terms of a new navigation and its collection operations, ie. in a recursive way, we will call "level" to the different composition of navigation. In the case that navigation B is used in the immediate definition of an operation collection for an navigation A, we said that B is in level 2 and A in level 1.
  The weight associated with each level is equal to the level number. Then the definition of the WNN metric is:
  WNN = $\sum$ weight of the level* number of navigations of the level.

- **DN Metric** (Depth of Navigations): Given that in an OCL expression there can be many navigations regarding its definition, we build a tree of navigation using the class name to which we navigates to. We will only consider navigations starting from the contextual instance (from *self*). The root of the tree is the class name which *self* represents. *Self* is always the starting point of any expression (sometimes *self* can be left implicit in an expression), then we build a branch for each navigation, where each class we navigate to is a node in the branch. Nodes are connected by "navigation relations". DN is the maximum depth of the tree.
  When navigation includes a collection operation expression defined in terms of new navigation(s), we will build a new tree for the navigation used in the collection operation expression, using the same method, then we will connect both trees using a "definition connection".

  Example. In the expression 2 of Fig. 2 defined for LoyaltyProgram there are two navigations. The tree built using the method described above is shown in Fig. 3 (a). Then, the value of DN is 2.

  Example. According to expression 5 of Fig. 2, the built tree using this method is shown in Fig. 3 (b). A dashed line represents a definition connection. When we obtain the depth of the tree, we will apply the following rule: "Navigation connection is counted one time, and definition connection twice". The DN value for the expression of Fig. 3 (b) is equal to 4.

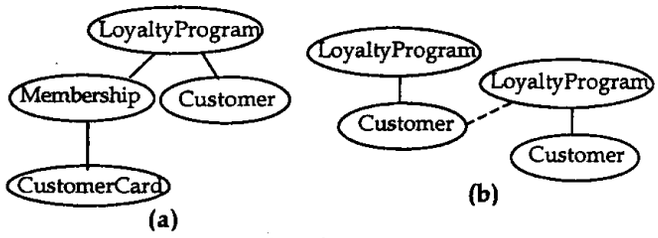


Fig. 3: Examples of navigation trees

[2] In order to analyze if two messages are different we will consider the object to which the message is sent and its operation call as we describe in WNO metric.

- **WCO Metric** (Weighted Number of Collection Operations): The collection operations used in the expression definition are weighted according to the level in which they are defined, so the metric is defined thus:
  WCO = Σ weight of the level * number of collection operations of the level.

5 THEORETICAL VALIDATION OF THE PROPOSED METRICS

To develop the theoretical validation of metrics we have used the property-based framework of 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.

5.1 NAN properties as coupling (interaction-based) metric [6]

In the context of [6] the coupling concept is defined as a relation between an individual software part, and its associated software system, rather than as a relation between two software parts. First at all, it is important to make some definitions prior to the application of properties of interaction-based metrics for coupling to the NAN metric.

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

We only have defined metrics related to import coupling. Our hypothesis is similar to the ISP-hypothesis of [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 [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 added navigation is an attribute already used by interaction. Then the metric NAN applied to the expression obtained, is equal to NAN(sp). (2) navigation added refers to a new attribute, then applied to the new expression is greater than NAN
- Merging Modules: This property can be expressed f context in the following way: "the sum of the ir coupling of two modules is no less than the coupling module which is composed of the data used of th modules". The value of NAN for an expression consists of the union of two original expressions, is ec the NAN of each expression merged when the s attributes referred to in each original expressic disjointed, otherwise is less than NAN of each expression merged.

In a similar way, it is possible to show that NNR, NNC, WNM, NPT, NUCA, NUCO are interaction-measures for coupling.

5.2 WNM as a size metric

For our purpose and in accordance with framework of et al. [7],[8], we consider that an OCL expression is a s composed of OCL messages (elements) and relationsh represented by the relation "belong to", which reflec message belongs to an OCL expression. A sub-expres be considered a module. We will demonstrate that fulfils all of the axioms that characterise size meti follows:

- Nonnegativity: Is directly proven and it is impos obtain a negative value.
- Null value: An expression e without a message WNM(e) = 0
- Module Additivity: If we consider that an OCL exp is composed of modules with no message in comm number of messages of an OCL expression it always the sum of the number of messages of its m In other words, when two modules (sub-expr without messages in common are merged then t expression has as many messages as each of the expressions. But if the original merged module expressions) have some message in common, th WNM of the resulting expression should be le adding the WNM of the original expressions. In a way, it is possible to show that WNN, WCO a metrics.

In a similar way, it is possible to show that WNN ar are size metrics.

5.3 DN as a length metric

For our purpose and in accordance with framev Briand et al. [7],[8], we consider that an OCL expres system. The elements are the classes (to wh expression navigates to) and the relationships navigations of UML relationship. We will demonstr DN fulfils all of the axioms that characterise length as follows:

- Nonnegativity and Null Value are straightfo satisfied, the depth of a tree can never be negativ

expression without navigation has a empty tree, and DN is 0.

- Nonincreasing monotonocity for connected components: If we add relationships between elements of a tree (classes or interfaces) the depth 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 which has more levels from the root to the leaves.

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

Metric Classification	OCL expression metrics		
	NNR, NAN, WNO, NNC, WNM, NPT,NUCA, NUCO.	DN	WNN,WCO, WNM
Size [7],[8]			X
Length [7],[8]		X	

6 Conclusions and Future Work

The main contribution of this paper is the definition of a set of metrics for OCL expressions in a rigorous and disciplined manner. The metrics were defined to measure structural properties of OCL expressions, considering only the OCL concepts that involve "tracing" technique.

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], we can conclude that NNR (Number of Navigated Relationships), NAN (Number of Attributes referred through Navigations), WNO (Weighted Number of referred Operations through navigations), NNC (Number of Navigated Classes), WNM (Weighted Number of Messages), NPT (Number of Parameters whose Type are classes defined in the class diagram), NUCA (Number of Utility Class Attributes used), NUCO (Number of Utility Class Operations used) are interaction-based coupling metrics; DN (Depth of Navigations) is a length metric and WNN (Weighted Number of Navigations), WCO (Weighted Number of Collection Operations), WNM (Weighted Number of Messages) are size metrics.

We are aware that to provide a complete knowledge about this set of metrics is necessary to perform the empirical validation of them. As many authors mentioned [3],[21],[28],[33], 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 a expression level, we will be able to extend them at a class level.

We believe that is necessary to define other metrics for OCL expressions which capture the intrinsic complexity of an OCL expression, for example the complexity related to the use of tuples, nested "let" expressions, the reuse of variables over multiple OCL expressions, etc.

Acknowledgments

This research is part of the following projects: 1) DOLMEN supported by the Subdirección General de Proyectos de Investigación - Ministerio de Ciencia y Tecnología (TIC 2000-1673-C06-06), 2) VII-J-RITOS2 (Red Iberoamericana de Tecnologías de Software para la Década del 2000), and 3) UNComa 04/E048 (Modelado de Componentes Distribuídos OO).

References

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

[2] V. Basili, and H. Rombach, "The TAME project: towards improvement-oriented software environments", In IEEE Transactions on Software Engineering 14(6), pp. 758-773, 1998.

[3] V. Basili, F. Shull and F. Lanubile,"Building knowledge through families of experiments", IEEE Transactions on Software Engineering, 25(4), pp. 435-437, 1999.

[4] L. C.Briand, S. Arisholm, F. Counsell, F. Houdek and P. Thévenod-Fosse, "Empirical Studies of Object-Oriented Artefacts, Methods, and Processes: State of the Art and Future Directions", Empirical Software Engineering, 4(4), pp. 387-404, 2000.

[5] L. C. Briand, W. Devanbu, and W. Melo, "An investigation into coupling measures for C++", 19th International Conference on Software Engineering (ICSE 97), Boston, USA, 412-421, 1997.

[6] L. C. Briand, S. Morasca, and V. Basili, "Defining and validating measures for object-based high level design", IEEE Transactions on Software Engineering, 25(5), pp. 722-743.

[7] L. C. Briand, S. Morasca, and V. Basili, "Property-based software engineering measurement", IEEE Transactions on Software Engineering, 22(1), pp. 68-85, 1996.

[8] L. C. Briand, S. Morasca and V. Basili, "Response to: comments 'Property-Based Software Engineering Measurement': Refining the additivity properties", IEEE Transactions on Software Engineering, 22(3), pp. 196-197, 1997.

[9] L. C. Briand, J. Wüst, "Modeling Development Effort in Object-Oriented Systems Using Design Properties", IEEE Transactions on Software Engineering, 27(11), pp. 963-986, 2001.

[10] L. C. Briand, J. Wüst, H. Lounis, "A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: an Industrial Case Study", 21st Int'l Conf. Software Engineering, Los Angeles, 345-354. 1999.

[11] L. C. Briand, J. Wüst, and H. Lounis, "Investigating Quality Factors in Object-oriented Designs: An Industrial Case Study", Technical report ISERN 98-29 (version 2), 1998.

[12] F. Brito e Abreu, and W. Melo, "Evaluating the impact of object-oriented design on software quality", Proceedings of 3rd International Metric Symposium, pp. 90-99, 1996.

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

[14] S. N. Cant, B. Henderson-Sellers, and D. R. Jeffery, "Application of

Cognitive Complexity Metrics to Object-Oriented Programs", Journal of Object-Oriented Programming, 7(4), pp. 52-63, 1994.

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

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

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

[18] M. Cartwright, "An Empirical view of inheritance. Information and Software Technology", 40(14), pp. 795-799, 1998.

[19] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, 20(6), pp. 476-493. 1994.

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

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

[22] F. Fioravanti, and P. Nesi, "Estimation and Prediction Metrics for Adaptive Maintenance Effort of Object-Oriented Systems", IEEE Transactions on Software Engineering, 27(12), pp. 1062-1083, 2001.

[23] M. Genero, "Defining and Validating Metrics for Conceptual Models", Ph.D. thesis, University of Castilla-La Mancha, 2002.

[24] M. Genero, M. Piattini, and C. Calero, "Early Measures For UML class diagrams", L'Objet, 6(4), Hermes Science Publications, pp. 489-515, 2000.

[25] M. Gogolla and M. Richters, "Expressing UML Class Diagrams properties with OCL", In Tony Clark and Jos Warmer, editors, Advances in Object Modelling with the OCL, pp. 86-115. Springer, Berlin, LNCS 2263, 2001.

[26] R. Harrison, S. Counsell, R. Nithi, "Coupling Metrics for Object-Oriented Design", Metrics 1998, Metrics 1998, pp. 150-156, 1998.

[27] ISO, "Software Product Evaluation-Quality Characteristics and Guidelines for their Use", ISO/IEC Standard 9126, Geneva, 2001.

[28] B. Kitchenham, S. Pfleger, and N. Fenton, "Towards a Framework for Software Measurement Validation", IEEE Transactions of Software Engineering, 21(12), pp. 929-943, 1995.

[29] A. Kleppe and J. Warmer, "Extending OCL to include Actions", UML 2000, York, UK, October 2000, Proceedings, Springer, LNCS, pp. 440-450, 2000.

[30] Object Modeling with the OCL. The Rationale behind the Object Constraint Language. Lecture Notes in Computer Science 2263. Springer.

[31] Response to the UML 2.0 OCL RfP (ad/2000-09-03), OMG Document ad/2002-05-09. Revised Submission, Version 1.5, June 3, 2002.

[32] N. Schneidewind, "Body of Knowledge for Software Quality Measurement", IEEE Computer 35(2), pp. 77-83, 2002.

[33] N. Schneidewind, "Methodology For Validating Software Metrics", IEEE Transactions of Software Engineering, 18(5), pp. 410-422, 1992.

[34] UML Specification Version 1.5, formal/03-03-01, OMG, Disponible en http://www.omg.org.

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

[36] J. Warmer and A. Kleppe, "The Object Constraint Language. Precise Modeling with UML", Object Technology Series. Addison-Wesley.

Luis Reynoso PhD student in Computer Science at the University of Castilla-La Mancha, Ciudad Real, Spain. MSc in Computer Science by National University of South, Bahía Blanca, Argentine (1993). Teacher Assistant at the National University of Comahue, Neuquén, Argentine. His research interests are focused on software specification, software metrics, object oriented metrics, conceptual data models quality, empirical software engineering.

Marcela Genero PhD in Computer Science by the University of Castilla-La Mancha, Ciudad Real Spain (2002). MSc in Computer Science by National University of South, Bahía Blanca, Argentine (1989). Assistant Professor at the Escuela Superior de Informática of the University of Castilla-La Mancha. She is a member of the ALARCOS research group, in the same University, specialized in Information Systems, Databases and Software Engineering. Author of several book chapters and papers on metrics for measuring the quality of conceptual models. Her research interests are: advanced databases design, software metrics, object oriented metrics, conceptual data models quality, database quality, empirical software engineering.

Mario Piattini MSc and PhD in Computer Science by the Politechnical University of Madrid. Certified Information System Auditor by ISACA (Information System Audit and Control Association). Full Professor at the Escuela Superior de Informática of the Castilla-La Mancha University. Author of several chapter books, and papers on Metrics for Conceptual Models, Databases, Software Engineering and Information Systems. He leads the ALARCOS research group of the Department of Computer Science at the University of Castilla-La Mancha, in Ciudad Real, Spain. His research interests are: advanced database design, database quality, software metrics, object oriented metrics, software maintenance.