# Métricas para propiedades estructurales de expresiones OCL relacionadas con la técnica de "chunking".

**Luis Reynoso**
Universidad Nacional del Comahue, Departmento de Ciencias de Computación, Neuquén, Argentina, 8300.
lreynoso@uncoma.edu.ar

**Marcela Genero, Mario Piattini**
Departmento de Ciencias de Computación, Universidad de Castilla La Mancha, Ciudad Real, España, 13071.
{Marcela.Genero, Mario.Piattini}@uclm.es

**Resumen**— La literatura sobre métricas incluye numerosas métricas orientadas a objetos (OO), algunas de las cuales se pueden aplicar a atributos internos de calidad de los diagramas de clases. Esta métricas se definieron con el objeto de poder estimar atributos de calidad externos, poder tomar mejores decisiones de diseño, y evitar, en etapas tardías, que el mantenimiento sea más caro en términos de costo y esfuerzo. OCL (Object Constraint Language) permite expresar, de forma más precisa, el conocimiento acerca del sistema a modelar. Sus expresiones declaran la semántica de diferentes atributos de los diagramas de clase, asociando expresiones del lenguaje a los elementos básicos del diagrama (esto es, clases y operaciones). Sin embargo, no existen en la literatura, métricas que tengan en cuenta la complejidad añadida por el uso de expresiones OCL. Por este motivo, presentamos en este artículo un conjunto de métricas para expresiones OCL que están relacionadas con el proceso cognitivo de "chunking".
Nuestra hipótesis es que las propiedades estructurales de expresiones OCL inciden en su complejidad cognitiva, y esta última, incide en atributos de calidad externos como la entendibilidad, la mantenibilidad, etc. En este trabajo mostramos la definición del conjunto de métricas (junto con los conceptos de OCL involucrados en tales métricas), y su validación teórica de acuerdo al marco basado en propiedades de Briand et al.

**Palabras Reservadas**— métricas orientadas a objetos, OCL, propiedades estructurales, complejidad cognitiva, entendibilidad, validación teórica.

## 1. Introducción.

Asegurar la calidad de los artefactos más significativos en las etapas inciales del desarrollo de software OO, evita la toma de decisiones en etapas tardías, que involucran cambios más caros e incluso más difíciles de llevar a cabo, que si las mismas decisiones se tomaran en etapas iniciales [1], [4], [9], [11], [17], [23], [34]. El diagrama de clases es el artefacto clave en la etapa inicial de desarrollo OO, por ello es fundamental evaluar su calidad

mediante métricas objetivas en pro de construir software de mayor calidad. De las métricas propuestas en la literatura que han sido aplicadas a diagramas de clase UML [36], la mayor parte se centraron en la medición de atributos internos de la calidad [5],[6],[18],[19],[24],[25],[27] como lo son: la complejidad estructural, el acoplamiento, el tamaño, etc. Pero las mediciones obtenidas no consideran todas las características semánticas del sistema modelado en un diagrama de clases, debido a que como argumentan varios autores "hay restricciones del sistema que no pueden expresarse utilizando solamente las notaciones gráficas que proporciona UML" [20],[33],[36]. Estas restricciones son escritas comúnmente en el lenguaje OCL (definido por OMG [32]) y nos permiten describir durante las etapas iniciales del desarrollo y de una forma más precisa, el conocimiento del sistema que se modela, asociando expresiones del lenguaje a los elementos básicos de los diagramas de clases (esto es, a clases y operaciones).

Otras características importantes de OCL es que facilita la documentación del sistema [38], mejora la comunicación entre desarrolladores (evitando errores producidos por malas interpretaciones) [38] y hace que la expresividad del sistema en etapas iniciales sea mayor.

Definiendo metricas válidas para expresiones escritas con OCL podremos considerar un espectro más amplio de la calidad del producto de software que se está modelando.

Debido a su importancia, y por ende a su contribución a la calidad del software, hemos comenzado a definir y validar métricas para expresiones OCL. Hemos abordado la definición de las métricas de acuerdo al marco metodológico basado en las propuestas de Calero et al. [13] y Cantone y Donzelli [16], el cual incluye un conjunto de etapas que permiten obtener métricas válidas. En en este trabajo nos ocuparemos solo de dos etapas, la definición y de la validación teórica de las métricas. Tanto la validación teórica como la empírica son igualmente importantes, pero es aconsejable realizar la validación teórica en primer lugar. Para la definición de las métricas nos basamos en:

1. El modelo teórico de Briand et al. [10] para el desarrollo de modelos cuantitativos relacionando métricas de productos y atributos de calidad externos [28], cuya hipótesis es que "las propiedades estructurales de un componente de software tiene un impacto en su complejidad cognitiva" [21], y que una alta complejidad cognitiva conduce a que un componente exhiba propiedades externas indeseables [12].
2. Y el Modelo de Complejidad Cognitiva para la entendibilidad de software, definido por Cant et al. [14],[15], donde se estudia la complejidad de software y su incidencia en la entendibilidad. Dicha entendibilidad puede pensarse en función de un "chunk" (una unidad de abstracción mental, producto del conocimiento semántico de los individuos) y de dos de sus técnicas cognitivas: "chunking" (técnica que involucra la entendibilidad de un chunk en sí mismo) y "tracing" (técnica que implica el barrido de información en diferentes direcciones con el objeto de entender otros chunks relevantes y relacionados a un chunk específico) [30].

Luego, la aplicación de ambos modelos a nuestro caso particular de expresiones OCL es mostrada en el esquema de la figura 1.

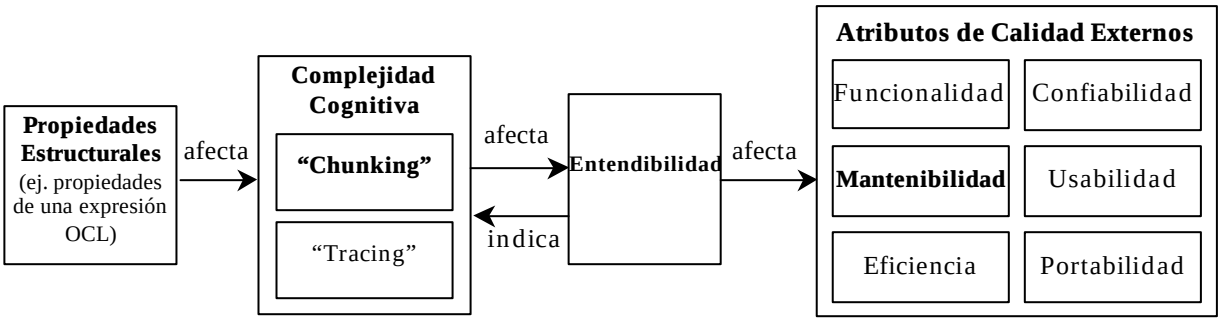


**Fig. 1:** Relación entre las propiedades estructurales, la complejidad cognitiva, la entendibilidad, y atributos de calidad externos (basados en [10] ,[28], [14] y [15]).

Consideramos que:

- Los modeladores en el proceso de entendimiento de las expresiones (un tipo especial de "chunk") emplearán tecnicas de "tracing" y "chunking", y,
- Del conjunto de propiedades estructurales de las expresiones OCL (nuestros componentes en la base teórica de Briand et al. citada en el punto 1) que inciden en las complejidad cognitiva de los modeladores, es posible separar dos subconjuntos primarios, aquellas propiedades estructurales que se relacionan con "tracing" y aquellas que se relacionan con "chunking".

Sin embargo como las técnicas cognitivas según Cant et al. [14] se emplean, sinérgica y concurrentemente, la separación de los conceptos de OCL y sus propiedades estructurales no es del todo exacta. Por ello hemos decidido dividir nuevamente el conjunto de propiedades estructurales que refieren a "chunking" en dos grupos. La Tabla 1 muestra un esquema sobre los conceptos relacionados con cada grupo y los documentos que hemos publicado sobre las métricas definidas para ellos.

Como se puede observar en la tabla 1, en [30] y [31] hemos definido y validado teóricamente un conjunto de métricas que refieren a "tracing" y al grupo 1 de "chunking" respectivamente. El presente documento pretende definir un conjunto de métricas para el grupo 2 de "chunking". El documento se estructura de la siguiente forma, el siguiente capítulo describe los conceptos de OCL involucrados en dicho grupo, el capítulo 3 presenta la definición de las métricas que se aplican a esos conceptos, y el capítulo 4 muestra su validación teórica. Finalmente se presentan las conclusiones y se describe el trabajo futuro.

## 2. Conceptos de OCL relacionados con "chunking".

La instancia contextual, la cual sirve como punto de referencia para la interpretación de una expresión OCL, está representada por la palabra reservada *self*. *Self* es fundamental en la definición del conjunto de métricas mostrado en este documento. A continuación mostraremos los conceptos principales de OCL especificados en su metamodelo [32] que nos servirán como base para interpretar correctamente como se obtiene el valor de las métricas.

**Tabla 1:** Esquema sobre técnicas cognitivas, conceptos de OCL relacionados y documentos de definición y validación de métricas para los mismos.

| **Técnica cognitiva** | **Conceptos de OCL relacionados con la técnica.** | **Documentos relacionados.** |
|---|---|---|
| "tracing" | Aquellos que permiten a una instancia contextual referirse a propiedades de objetos definidos en otras clases, por ejemplo: navegación, mensajes, parámetros cuyos tipos son clases en un diagrama de clases, clases de utilidad, etc. | En [30] se define un conjunto de métricas y se valida teóricamente con el marco de Briand et al.[6],[7],[8]. |
| Grupo 1 de "chunking" | Características propias del lenguaje OCL, esto es, iteradores, variables definidas por *let* y que sólo son posibles reusar en el ámbito de una expresión, subexpresiones del tipo *if*, etc. | En [31] se define un conjunto de métricas y se valida teóricamente con el marco de Briand et al. [7],[8]. |
| Grupo 2 de "chunking" | Refieren a la instancia contextual de una expresión OCL: mecanismos de OCL que permiten referirnos a atributos y operaciones pertenecientes al tipo que representa *self*, variables que pueden ser reutilizables en el ámbito del tipo que representa *self*, operaciones definidas en el lenguaje que comparan el tipo que representa *self* con otros tipos (relacionados por medio de herencia), etc. | La definición y validez teórica de métricas para este grupo es el objetivo principal de este documento. |

- **Referencias a Atributos u Operaciones que pertenecen al mismo tipo que representa *self***

  Las expresiones OCL pueden referirse a Classifiers [32] (tipos, clases, interfaces, tipos de datos, etc.) y a sus propiedades. Se considera en este documento al igual que en [32] que una *propiedad* es:
  - un atributo,
  - un extremo final de una asociación,
  - operaciones y métodos libres de efectos laterales.

  Mientras que el acceso a una propiedad del tipo "extremo final de una asociación" nos permite navegar una relación (característica relacionada con el "tracing"), referirnos a una propiedad atributo u operación a continuación de la instancia contextual (y en conjunto con "la notación punto") nos permite acceder a estos tipos de propiedades del Classifier que representa *self*.

  **Ejemplo 1**: En la siguiente expresión invariante:

  *context Persona invariant:*
  *self.nivel = "Senior" implies self.edad() = 21*

  se hace referencia a dos propiedades:
  - la primera, el atributo *nivel* perteneciente al tipo representado por la instancia contextual *self* (esto es, Persona).
  - la segunda, la operación *edad()*, también perteneciente al mismo tipo.

- **Restricciones de Definición**[1]

  Es posible reutilizar variables en el contexto de una expresion OCL o en el contexto de múltiples expresiones OCL. Las variables reutilizadas pueden ser definidas a partir de

[1] Del inglés "<<definition>> constraints"

expresiones *let* (permitiendo su reutilización en el ámbito de una expresión), o pueden ser definidas a partir de una restricción de definición (lo cual permite su reutilización en el ámbito de cualquier expresión asociada a una clase) [32].

**Ejemplo 2** En la siguiente restricción de definición se define una variable, *sueldo*, para que la misma sea reutilizada en cualquier expresión asociada a la clase Persona:

*context Person def:*
*attr sueldo: Integer = self.job.salary->sum()*

De esta forma podríamos escribir una expresión para la clase Persona haciendo uso de la variable *sueldo.*

*context Person invariant:*
*if self.esDesempleado then*
*sueldo < 100 else*
*sueldo >= 100 endif*

- **Operaciones predefinidas**

Hay diversas propiedades que se aplican a todos los objetos, y están predefinidas en OCL, ellas son:

- **oclIsTypeOf (t :OclType):Boolean**

The operación oclIsTypeOf retorna verdadero si el tipo de *self* es idéntico al tipo de su argumento (t).

- **oclIsKindOf (t :OclType):Boolean**

La propiedad oclIsKindOf determina si el tipo de *self* es idéntico al tipo de su argumento o a cualquiera de los supertipos de un objeto.

**Ejemplo 3:** Dada la jerarquía de clases de la figura 2, en el contexto de la clase Diseñador, el valor de verdad de cada una de las siguientes operaciones predefinidas serían:

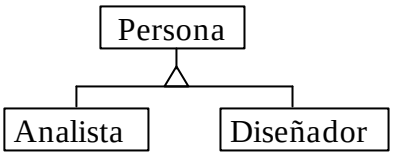


**Fig. 2.**

*self.oclIsTypeOf(Persona) = false self.oclIsKindOf(Persona) = true*
*self.oclIsTypeOf(Diseñador) = true*
*self.oclIsKindOf(Diseñador) = true*

- **oclAsType (t :OclType): instancia de OclType**

Siempre que una propiedad es redefinida dentro de un tipo, la propiedad de los supertipos pueden ser accedidas usando la operación oclAsType().

**Ejemplo 4**: Si B es un subtipo de la clase A entonces es posible escribir:

*context B invariant:*
*self.oclAsType(A).p1*

para referirnos a la propiedad p1 definida en A.

De acuerdo a su funcionalidad las operaciones oclIsTypeOf, oclIsKindOf y oclAsType son utilizadas frecuentemente con conceptos involucrados con herencia.

- **Valores previos en postcondiciones:** Por lo general el valor de una propiedad en una postcondición es el valor de la misma una vez que la operación se ha completado. Para referirnos al valor de una propiedad al comienzo de la operación, es necesario utilizar el postfijo "@pre" junto con el nombre de la propiedad [32].

  **Ejemplo 5** En la siguiente postcondición de un ejemplo obtenido de [20] la propiedad *usage* refiere a la propiedad de la instancia *Bathroom* que ejecuta la operación. La propiedad *usage@pre* refiere al valor de propiedad *usage* de Bathroom que ejecuta la operación, al comienzo de la operación [32].

  *context Bathroom::uses (g: Guest)*
  *pre: .....*
  *post: usage = usage@pre + 1*

## 3. Definición de Métricas

Mientras que la medición es considerada cada vez más como una base esencial para la toma de decisiones, es necesario prestar atención a la calidad de los procesos de obtención de métricas. Por ello, aplicaremos un marco metodológico que permite asegurar la validez y la confiabilidad en la obtención de métricas correctas. Como mencionamos anteriormene la definición de métricas se basa en el marco metodológico de Calero et al. [13], y Cantone y Donzelli [16], y del conjunto de etapas que incluye el marco sólo nos ocuparemos de la definición y de la validación teórica de las métricas.

Por otro lado, para definir claramente el objetivo a alcanzar mediante la definición de métricas, tendremos en cuenta un componente importante de la medición orientada a metas la cual se basa en el paradigma GQM (Goal/Question/Metric), utilizaremos la plantilla GQM [2],[37]. La tabla 2 muestra la aplicación de la plantilla a nuestro caso.

**Tabla 2:** Plantilla GQM aplicada a la definición de metricas para expresiones OCL relacionadas con la técnica de "chunking".

| | |
|---|---|
| Analizar | *Estructuras de expresiones OCL relacionadas con técnicas de "chunking"* |
| con el propósito de | *Evaluar* |
| con respecto a su | *Entendibilidad* |
| desde el punto de vista de | *Modeladores de software OO* |
| en el contexto de | *Organizaciones de desarrollo de software* |

La tabla 3 contiene un resumen del conjunto de métricas definidas, las cuales involucran los conceptos definidos en la sección anterior.

**Tabla 3:** Definición de Métricas para la técnica de "chunking"

| Siglas de la Métrica | Descripción de la Métrica. |
|---|---|
| NAS | Número de atributos referidos a partir de *self* en forma inmediata (atributo perteneciente al Tipo que representa *self*). |
| NOS | Número de operaciones referidos a partir de *self* en forma inmediata (operacion perteneciente al Tipo que representa *self*). |
| NVD | Número de variables utilizadas en una expresión que han sido definidas en restricciones "definición". |
| N@P | Número de distintas propiedades postfijadas con @pre. |
| NIO | Número de veces que se utiliza una operación oclIsTypeOf, oclIsKindOf ó oclAsType en una expresión. |

A continuación definiremos cada una de ellas en forma más precisa y para ilustrar su utilización mostraremos un ejemplo.

- **NAS:** Esta métrica representa el número de atributos referidos en una expresión, donde esos atributos pertenecen al Classifier que representa *self*.

  **Ejemplo**:
  *context Persona invariant:*
  *ref = (if self.esHombre = true then*
  *'Mr.' else 'Ms.' endif)*
  En este ejemplo se utilizan dos atributos de la clase Persona, *ref* y *esHombre*, en el primer caso la instancia contextual *self* está implícita y en el segundo explícita, por lo tanto el valor de NAS es igual a 2.

- **NOS:** Esta métrica representa el número de operaciones referidas en una expresión, donde esas operaciones pertenecen al Classifier que representa *self*.

  **Ejemplo**: El valor de NOS en la expresión invariante del ejemplo 1 es igual a 1, debido a que solo se utiliza la operación *edad()* de la clase Persona.

- **NVD:** Esta métrica toma en cuenta el número de variables que han sido definidas a partir de una restricción definición. La métrica NVD no toma en cuenta las veces que una variable es utilizada en una expresión sino la cantidad de variables definidas.

  **Ejemplo:** La métrica NVD en la expresión invariante del Ejemplo 2 toma valor 1 debido a que se utiliza la variable *sueldo*, la cual ha sido definida en una restricción definición.

- **NIO**: Esta métrica representa el número de veces que se utiliza alguna de las siguientes propiedades predefinidas: oclIsTypeOf, oclIsKindOf ú oclAsType.

  **Ejemplo:** Utilizaremos una figura y una expresión OCL definida en [38], para ejemplificar la aplicación de la métrica NIO.
  Dada la siguiente expresión correspondiente a la clase *ApplePie* de la figura 2,

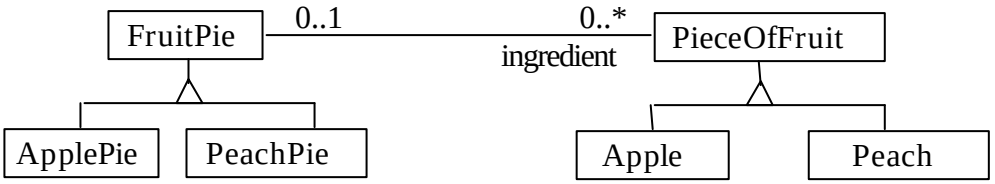


**Fig. 2**. Ejemplo para ilustrar la utilización de *oclIsKindOf()*.

*context ApplePie invariant:*
*self.ingredient->forAll(oclIsKindOf(Apple))*

El valor de la métrica NIO aplicado a esta expresión es igual a 1 debido a que hemos utilizado la operación oclIsKindOf() una sola vez.

Somos conscientes que la métrica NIO no sólo hace uso del tipo representado por *self*, sino que hace uso de otros tipos conectados con el tipo que representa *self* a partir de mecanismos de herencia, lo cual puede ser interpretado como propio de la técnica de "tracing", pero hemos preferido asociar estas operaciones al conjunto de propiedades del grupo de "chunking". Este es un claro ejemplo, en el cual ambas técnicas se emplean concurrentemente.

- **N@P**: Esta métrica (aplicada exclusivamente a postcondiciones) representa el número de diferentes propiedades postfijadas con "@pre", es decir representa la cantidad de propiedades a las cuales se hace referencia a sus valores previos en postcondiciones.

  **Ejemplo:** Considerando el ejemplo 5 el valor de N@P es 1, debido a que el postfijo "@pre" es utilizado solamente con la propiedad *usage*.

## 4. Validación Teórica.

La validación teórica de las métricas, se realiza para asegurar que las métricas miden el atributo para el cual fueron definidas. Esto permite asegurar la validez de constructo, aspecto sumamente importante a la hora de utilizar estas métricas en estudios empíricos [13]. Además nos permite conocer la escala a la que pertenece una métrica y así determinar que operaciones estadísticas se pueden realizar con las valores de las métricas. Hay dos tendencias principales para realizar validación teórica de las métricas: una basada propiedades y otra basada en la teoría de la medida.

Para validar teóricamente las métricas definidas en la sección anterior utilizaremos dos aproximaciones basadas en propiedades:

- el marco formal de Briand et al. [7], [8] que permite clasificar las métricas de acuerdo al atributo que pretenden medir, esto es métricas de: acoplamiento, tamaño, complejidad, longitud y cohesión.
- el marco formal de Briand et al. [6] en el cual se definen propiedades de métricas basadas en la interacción para acoplamiento y cohesión de un diseño de alto nivel.

### 4.1. Propiedades de NVD como Métrica de Tamaño (Marco de Briand et al. [7],[8]).

Para nuestro propósito y de acuerdo al marco formal de Briand [7], [8], consideramos que una expresión OCL es un sistema compuesto de variables definidas por medio de restricciones de "definición" (elementos) y las relaciones están representadas por la

relación "pertenece a", la cual refleja que una variable de ese tipo pertenece a una expresión OCL. Una sub-expresión será considerada un módulo. Demostraremos que NVD satisface todos los axiomas que caracterizan a las métricas de tamaño, como se muestra a continuación:

- **No-negatividad:** Se prueba directamente, es imposible obtener un valor negativo.
- **Valor nulo:** Una expresión OCL *e* que no reutiliza variables definidas por medio de restricciones definición tiene un NVD(*e*) = 0
- **Aditividad de Módulos:** Cuando dos expresiones son unidas y estas expresiones no cuentan en su definición con variables (definidas por medio de restricciones definición) en común, la nueva expresión (producto de la unión) tiene tantas variables como cada una de las expresiones unidas. Pero si las expresiones originales tienen alguna variable en común el NVD de la expresión resultante debería ser menor que el NVD de las expresiones originales.

### 4.2. Propiedades de NAS como Métrica de Acoplamiento Basado en la Interacción (según Briand et. al. [6]).

En el contexto de [27] el concepto de acoplamiento está definido como una relación entre una parte de software individual, y su sistema de software asociado, en lugar de una relación entre dos partes de software. Para demostrar que la métrica NAS (Número de atributos referidos a partir de *self* en forma inmediata, donde el atributo pertenece al Tipo que representa *self*) es una métrica de acoplamiento basado en la interacción [6] tendremos en cuenta que: una expresión OCL será considerada una parte individual de software, y un conjunto de atributos a los cuales se hace referencia en dicha expresión (posiblemente vacío) será considerado el sistema de software asociado a dicha parte. La referencia representará la interacción entre la parte de software y el sistema de software asociado.
Además tendremos en cuenta las siguientes definiciones:

- Una interacción DU es la interacción entre la declaración de un dato y su utilización en una expresión OCL.
- Acoplamiento importado: Dado una parte de software *sp* (una expresión OCL), el acoplamiento importado de *sp* es el número de interacciones DU entre la declaración de dato externa a *sp* y la utilización del dato (atributos referidos a partir de navegación) dentro de *sp*.

La hipótesis de [6] es: "a mayor número de partes de software referenciadas, mayor será el contexto que requiere ser entendido, y mayor probabilidad de ocurrencia a fallos"

Las propiedades de acoplamiento basado en interacción son las siguientes:

- **No-negatividad:** Dada una parte de software sp, la métrica de acoplamiento aplicada a *sp* es mayor ó igual a 0.

  Aplicado a la métrica NAS, podemos decir que si una expresión *sp* (la parte del software que consideramos) no refiere a atributos pertenecientes al mismo tipo que representa *self* entonces NAN(*sp*) = 0. Por otro lado, la métrica NAS nunca puede tomar valores negativos. Luego la propiedad se verifica.

- **Monotonicidad:** El hecho de incrementar las interacciones importadas no puede decrementar su acoplamiento importado.

  Si agregamos a una expresión *sp* una referencia a un atributo de ese tipo obtenemos una nueva expresión modificada, digamos *sp'*; luego, pueden suceder dos casos: (1) el atributo al cual nos referimos es un atributo al cual ya se hacia referencia en alguna parte de la expresión *sp*, por lo tanto NAN(*sp*) = NAN(*sp'*). (2) el atributo al cual no referimos es un nuevo atributo, por lo tanto NAN(*sp*) < NAN(*sp'*). Por lo tanto la propiedad se verifica.

- **Unión de Módulos:** La suma de los acoplamientos importados de dos módulos no es menor que el acoplamiento de la unión de los dos módulos.

  El valor de la métrica NAS para una expresión que consiste de la unión de otras dos expresiones originales sólo puede mantenerse constante cuando los conjuntos de atributos referidos por cada expresión original son disjuntos, y es menor cuando dichos conjuntos no son disjuntos. Por lo tanto, la propiedad se verifica.

De igual forma es posible demostrar que NOS, N@P y NIO son métricas de acoplamiento basadas en interacción.

## 5. Conclusiones.

Hemos presentado y validado teóricamente en este documento un conjunto de métricas para expresiones OCL seleccionando un conjunto de propiedades estructurales de OCL que están relacionadas con la técnica de "chunking", más precisamente hemos seleccionado aquellos conceptos de OCL que refieren directamente al tipo que representa la instancia contextual de una expresión. La instancia contextual, *self*, puede ser considerada un punto de referencia para la interpretación de la expresión, por ello hemos considerado que estas características relacionadas con la técnica cognitiva de "chunking". Midiendo los conceptos que demandan técnicas de "chunking" contribuiremos a estimar la carga cognitiva aplicada a los modeladores.

Las métricas definidas pueden ser aplicadas a expresiones OCL que complementan a los diagramas de clases UML, y tratan de revelar como las propiedades estructurales de una expresión OCL tienen impacto en un atributo de calidad externo: la entendibilidad. Las métricas definidas en este documento fueron validadas teóricamente como métricas de tamaño (NVD) y métricas de acoplamiento basado en la interacción (NAS, NOS, N@P y NIO ) de acuerdo al marco formal de Briand et al. [6], [7], [8].

Somos conscientes que para poder proveer un conocimiento acabado sobre la utilidad de las métricas definidas en este trabajo y en [30] y [31] es necesario realizar su validación empírica, a través de experimentos y casos reales. La validación empírica es fundamental para asegurar que las métricas son realmente útiles en la práctica [3],[22],[29],[35]. Por ello, actualmente estamos planificando un experimento controlado que realizaremos en una primera instancia con alumnos y luego lo aplicaremos con profesionales.

## Agradecimientos

La investigación es parte de los siguientes proyectos: 1) CALDEA TIC 2000-0024-P4-02,

financiado por la Subdirección General de Proyectos de Investigación, Ministerio de Ciencia y Tecnología 2) VII-J-RITOS2 (Red Iberoamericana de Tecnologías de Software para la Década del 2000), y 3) UNComa 04/E048 (Modelado de Componentes Distribuídos Orientados a Objetos).

## Referencias

[1] J. Bansiya y 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. Basili y H. Rombach, "The TAME project: towards improvement-oriented software environments", *IEEE Transactions on Software Engineering* 14(6), junio, 1998, pp. 758-773.

[3] V. Basili, F. Shull. y F. Lanubile F. "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 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.

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

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

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

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

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

[10] L. C. Briand, J. Wüst., S. Ikonomovski y H. Lounis, "Investigating Quality Factors in Object-oriented Designs: An Industrial Case Study", En *Proceedings of the 21st IEEE International Conference on Software Engineering ICSE'99*, mayo, 1999, pp. 345-354.

[11] L. C. Briand, J. Wüst, 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 y W. Melo, "Evaluating the impact of object-oriented design on software quality", En *Proceedings of 3rd International Metric Symposium*, marzo, 1996, pp. 90-99.

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

[14] S. N. Cant, B. Henderson-Sellers y D. R. Jeffery, "Application of Cognitive Complexity Metrics to Object-Oriented Programs". *Journal of Object-Oriented Programming*, 7 (4), Julio-Agosto, 1994, pp. 52-63.

[15] S. N. Cant., D. R. Jeffery y B. Henderson-Seller, "A Conceptual Model of Cognitive Complexity of Elements of the Programming Process". *Information and Software Technology*, 37 (7), julio, 1995, pp. 351-362.

[16] G. Cantone y P. Donzelli, "Production and maintenance of software measurement models". *Journal of Software Engineering and Knowledge Engineering*, 5, 10(5), octubre, 2000, pp. 605-626.

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

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

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

[20] S. Cook, A. Kleepe, R. Mitchell, B. Rumpe, J. Warmer, y A. Wills. "The Amsterdam Manifiesto on OCL. Object Modeling with the OCL", LNCS 2263, pp. 115-149, 2002.

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

[22] N. Fenton y S. L. Pfleeger, "Software Metrics: A Rigorous and Practical Approach", PWS Publishing Co, 2nd Edition, 1997.

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

[24] M. Genero, "Defining and Validating Metrics for Conceptual Models", Tesis de Doctorado, Universidad de Castilla-La Mancha, 2002.

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

[26] M. Gogolla y M. Richters, "Expressing UML Class Diagrams Properties with OCL", En Tony Clark and Jos Warmer editors, Object Modeling with the OCL, Springer, Berlin, LNCS 2263, 2001, pp. 86-115.

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

[28] ISO, Software Product Evaluation-Quality Characteristics and Guidelines for their Use, ISO/IEC Standard 9126, Geneva, 1999.

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

[30] L. Reynoso, M. Genero y M. Piattini, "Measuring OCL expressions: a "tracing"-based approach", Workshop on Quantitative Approaches in Object-Oriented Software Engineering. QAOOSE' 20003.

[31] L. Reynoso, M. Genero y M. Piattini, “Métricas para expresiones OCL relacionadas con la técnica cognitiva de “chunking””, XXIX Congreso Latinoamericano de Informática, CLEI 2003, Bolivia. En proceso de revisión.

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

[33] B. Rumpe, “<<Java>>OCL Based on New Presentation of the OCL-Syntax”, Object Modeling with the OCL, LNCS 2263, pp. 189-212, 2002.

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

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

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

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

[38] J. Warmer y A. Kleppe, “The Object Constraint Language. Precise Modeling with UML”, Object Technology Series. Addison-Wesley, 1999.