Jornadas sobre Innovación y Calidad del Software

26/27 de junio de 2003

Fundació Politècnica
de Catalunya (Barcelona)

Organizadas por el Grupo
de Calidad de Software
de la Asociación de
Técnicos de Informática

Asociación de Técnicos de Informática

UPC Fundació Politècnica de Catalunya: la formació contínua

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Actividad vinculada al Master de Ingeniería del Software

Patrocinado por:

NTE SA

inQA.labs Software Testing

Telelogic

MÉTODOS Y TECNOLOGÍA

IBM

gesein

Programa de las VIII Jornadas sobre Innovación y Calidad del Software

Jueves 26 de junio

9:00 Entrega de documentación
9:30 Apertura a cargo del *Sr. Pere Lluís Barberà*, Presidente de ATI-Catalunya y la *Sra. Mercé Sala*, Presidenta de la Fundació Politécnica de Catalunya.
10:00 Medida del Coste de la Calidad y ROI de las actividades de Calidad y Pruebas, *José Javier Domingo*, Empresas de Calidad Software
10:30 Midiendo la complejidad de expresiones OCL: una aproximación basada en "tracing" , *Luis Reynoso, Mario Piattini, Marcela Genero*, Univ. Nac. de Comahue, Univ. de Castilla-La Mancha
11:00 Ponencia patrocinador : "ALM: Application Lifecycle Management. Busqueda de la calidad mediante la automatización y control de los procesos de desarrollo software", *Antonio Rodríguez Perales,* Telelogic
11:30 Café
12:00 Wireless Testing Framework, *Mamdouh El Cuera*, Métodos y Tecnología
12:30 ¿Podemos automatizar una inspección de código?, *Celestina Bianco*, NTE
13:00 Ponencia patrocinador: "Maximizar el retorno de las pruebas de rendimiento mediante simulaciones eficientes", *Raynald Korchia*, inQA.labs
13:30 Coloquio sobre "Pruebas de software y control de calidad"
14:00 Comida
15:30 El eslabón perdido en la Calidad del Software, *Raquel Navarro*, Universitat Oberta de Catalunya
16:00 Ponencia patrocinador : Un modelo organizativo para la mejora continuada de procesos de SW. Pautas para la puesta en marcha, *Eulalia Clos*, NTE.
16:30 Café
17:00 Un Modelo Tridimensional de Calidad de la Web, *Julián Ruiz, Coral Calero, Mario Piattni.* Universidad de Castilla-La Mancha
17:30 Ponencia patrocinador : "Herramientas y Outsourcing para la Estimación y Gestión de Proyectos. Claves del éxito en la Gestión de Proveedores", *Ramiro Carballo*, GESEIN
18:30 Coloquio: "¿Importa la calidad en el negocio del software? Empresas, profesionales, etc."
19:00 Final de jornada

Viernes 27 de junio

10:00 Creación de un entorno de test automático eficaz y eficiente. Medida del retorno de la inversión. *Raúl Rubio*. InQA.labs
10:30 Generación de casos de prueba a partir de especificaciones UML , *Luis Fernández, Pedro J. Lara.* Universidad Europea de Madrid, *Celia Gutiérrez*, Universidad Camilo José Cela
11:00 Ponencia patrocinador: "La plataforma de desarrollo de Software para un mundo on demand", *Ana Lopez-Mancisidor Rueda*, Rational IBM España
11:30 Café
12:00 Dos años del departamento de Aseguramiento de la Calidad de la Construcción de Software en NTE: *Joan Bosch, Sara Martín*, NTE.
12:30 Un nuevo método para la aplicación simultánea de ISO/IEC 15504 y ISO 9001:2000 en PYMES de desarrollo de software, *Antònia Mas, Esperança Amengual*, Universitat de les Illes Balears
13:00 Ponencia patrocinador :Outsourcing de Calidad de Software: Uso de "La Factoría", *José Manuel de Río,* Métodos y Tecnología
13:30 Coloquio sobre "Medición y procesos de software"
14:00 Cierre a cargo del *Sr. Josep Molas*, Presidente de ATI y el *Sr. Eduard Pallejà*, Director General de la Fundació Politècnica de Catalunya
14:15 Cocktail final

[redacted]

Una propuesta de métricas para expresiones OCL basada en "tracing"

Luis Reynoso
Universidad Nacional del Comahue
Facultad de Economía y Administración
8300 Neuquén, Argentina
lreynoso@uncoma.edu.ar

Marcela Genero, Mario Piattini
Universidad de Castilla La Mancha
Grupo de Investigación Alarcos
13071, Ciudad Real, España
{Marcela.Genero, Mario.Piattini}@uclm.es

RESUMEN

Ninguna de las métricas existentes propuestas considera la complejidad añadida a los diagramas de clases UML al incorporarles expresiones escritas en el "Object Constraint Language" (OCL). Por ello, proponemos un conjunto de métricas para las propiedades estructurales de expresiones OCL considerando aquellos conceptos del lenguaje que se relacionan con la técnica de "tracing". Consideramos que dicha técnica afecta la complejidad cognitiva y la entendibilidad de expresiones, lo que influenciará directamente al mantenimiento de diagramas de clases.
En este trabajo nos centraremos en la definición y la validación teórica de las métricas siguiendo los marcos formales de Briand et al.

Palabras clave

Expresiones OCL, propiedades estructurales, complejidad cognitiva, entendibilidad, métricas OO, validación teórica.

1. INTRODUCCIÓN

En ingeniería de software es ampliamente conocido que el uso de métricas en las fases iniciales de ciclo de vida del software orientado a objeto (OO) ayuda a los modeladores y diseñadores a construir software OO de mejor calidad, sin tener que realizar revisiones innecesarias en etapas posteriores de desarrollo cuando los cambios son más caros e incluso difíciles de llevar a cabo [1], [2], [3], [4], [5], [6], [7]. Además la disponibilidad de tales métricas permite predecir atributos de calidad externos [8], [9], tales como la entendibilidad y la facilidad de mantenimiento [10]. En estas etapas iniciales uno de los artefactos claves es el diagrama de clases, ya que en ellos se basa todo el trabajo de diseño e implementación posterior. Por ello es necesario prestar especial atención a la calidad de los diagramas de clases. En la literatura actual existen numerosas métricas que pueden aplicarse a diagramas de clases UML [11] en una etapa de alto nivel [12],[13],[14],[15],[16],[17],[18]. La mayoría de estos trabajos se centran en la medición de atributos internos de la calidad como la complejidad estructural, el acoplamiento, el tamaño, etc. Pero ninguna de las propuestas de métricas existentes considera la complejidad añadida a los diagramas de clases UML al incorporarles expresiones escritas en OCL. Sin lugar a dudas la aparición del lenguaje OCL, definido por OMG, ha resultado fundamental en el desarrollo de software OO utilizando UML, ya que permite obtener un modelamiento preciso con UML, proveyendo al modelador de una notación expresiva para capturar las propiedades esenciales del sistema [19]. OCL realmente enriquece los diagramas UML ya que los complementa con expresiones que especifican propiedades semánticas del modelo [20], mejoran la precisión del sistema, su documentación [21], y su comprensibilidad en etapas iniciales del desarrollo. La importancia de OCL nos llevó a pensar en la necesidad de contar con métricas para medir las propiedades estructurales de expresiones OCL.
Briand et al. [22] han definido una base teórica para el desarrollo de modelos cuantitativos que relacionan las propiedades estructurales y los

atributos de calidad externos. Asumimos en este trabajo una representación similar para las expresiones OCL. Implementamos la relación entre las propiedades estructurales por un lado, y la entendibilidad y mantenibilidad por el otro (ver figura 1). Nuestra hipótesis es que las propiedades estructurales de una expresión OCL tienen un impacto en su complejidad cognitiva. El concepto de complejidad cognitiva significa la carga mental de las personas que tienen que tratar con artefactos (por ej. modeladores, personas que realizan verificación o mantenimiento). Una alta complejidad cognitiva conduce a que una expresión reduzca su entendibilidad y esto conduce a cualidades externas indeseables, tal como una mantenibilidad menor.

Suponemos que las propiedades estructurales de expresiones OCL tienen un impacto en la complejidad cognitiva de los modeladores,

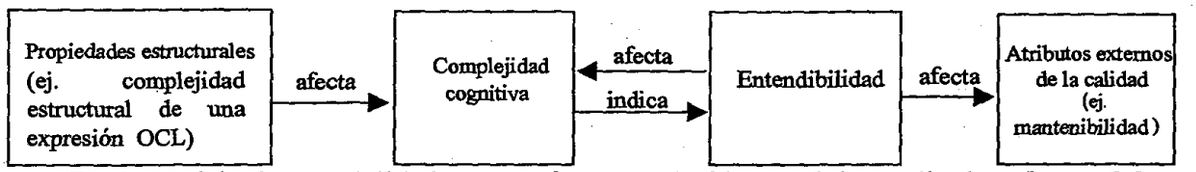


Fig 1: Modelo de complejidad para artefactos producidos en el desarrollo de software OO.

debido a que al querer comprender expresiones OCL se emplean técnicas cognitivas, tales como "chunking" y "tracing"[1] [23],[24],[25]. Por ello se requiere estudiar y definir métricas que refieran á las propiedades estructurales de expresiones OCL concernientes a esas técnicas cognitivas empleadas.

Por todo lo dicho, el siguiente artículo persigue dos objetivos:

1. Proponer de manera metodológica un conjunto de métricas para propiedades estructurales de expresiones OCL, considerando sólo aquellos conceptos de OCL especificados en su metamodelo [26], que afectan al "tracing".
2. Asegurar que las métricas propuestas realmente miden lo que pretenden medir, a través de su validación teórica, siguiendo los marcos formales de Briand et al. [13],[17],[28] creados para tal propósito.

Para cumplir nuestro primer objetivo es importante describir en primer lugar algunos conceptos del modelo cognitivo de Cant el al. [24] en el cual nos hemos basado para definir las métricas (ver sección 2) y luego mostrar cuáles conceptos de OCL involucran el empleo de esa técnica durante la entendibilidad de una expresión (ver sección 3). De este modo seremos capaces de presentar la definición de métricas y sus validaciones.

La definición propiamente dicha de las métricas y su ilustración mediante ejemplos es presentada en la sección 4. La validación teórica de las mismas es presentada en la sección 5. Y finalmente se presentan las conclusiones y algunas líneas que quedan pendientes para una futura investigación.

2. TÉCNICAS COGNITIVAS QUE TIENEN IMPLICACIONES EN LA CARGA COGNITIVA.

El Modelo de Complejidad Cognitiva (CCM) de Cant et al. [24] provee una teoría cognitiva general de complejidad de software y su incidencia en entendibilidad [25]. Aunque el estudio del CCM ha sido considerado un punto de partida razonable para la comprensión del impacto de las propiedades estructurales en la entendibilidad de código fuente por parte de programadores, el modelo puede ser aplicado a

[1] El nombre de ambas técnicas están expresadas en inglés, y debido a no contar con un término adecuado en español para las mismas se optó por mantener el nombre en su lenguaje original.
La técnica de "chunking", involucra el reconocimiento de grupos de declaraciones, mientras que la técnica de "tracing" se lleva a cabo en conjunto con la de "chunking" permitiendo resolver dependencias de ese grupo de declaración con otros conceptos. Ver sección 2 para mayor detalle.

modeladores de UML durante la comprensibilidad de especificaciones OCL. En CCM se argumenta que la comprensión consiste de dos técnicas: "chunking" y "tracing", que son concurrentemente y sinérgicamente aplicadas en la resolución de problemas [24], [25].

- Las técnica de "chunking", la cual es parte de la capacidad de memoria a corto plazo (STM), involucra el reconocimiento de grupos de declaraciones (no necesariamente secuenciales) y la extracción de información que es recordada en una única abstracción mental: un "chunk"[2].
- Las técnicas de "tracing" involucran el barrido de información, en diferentes direcciones, con el objetivo de identificar "chunks" relevantes [25] para resolver determinadas dependencias de conceptos.

La determinación de qué constituye un "chunk" es difícil según Cant [24], debido a que es producto del conocimiento semántico. Para nuestro propósito consideraremos a una expresión OCL como una unidad de "chunk".

Aunque las dos técnicas, como se expresó anteriormente, se aplican concurrentemente, nos enfocaremos en primer instancia a la técnica de "tracing".

Henderson-Sellers nota que las técnicas de "tracing" interrumpen el proceso de "chunking". Debido a que durante la entendibilidad de un "chunk" se requiere comprender el "chunk" en sí mismo, comprender los "chunks" en los cuales éste depende, y realizar "tracing" en esas dependencias. Como ejemplo podemos citar que los programadores deben efectuar "tracing" en la comprensión de código fuente 'cuando un método invoca a otro método de una clase diferente.' [25]. Este proceso o técnica cognitiva también es realizada por modeladores de UML durante la entendibilidad de especificaciones OCL. Para comprender una expresión OCL, como un "chunk", los desarrolladores deben llevar a cabo técnicas de "tracing" para entender algunas propiedades (atributos, operaciones, navegaciones) que envuelven a otras clases, entender el envío de un mensaje, etc. Son estos conceptos los explicados a continuación.

3. CONCEPTOS DE OCL QUE REQUIEREN PROCESOS DE "TRACING".

Aunque el propósito de OCL es que éste sea utilizado para definir expresiones sobre cualquier diagrama UML, nos hemos enfocado en aquellas expresiones que se definen en diagramas de clases UML. OCL es utilizado en este tipo de diagramas principalmente para: "especificar invariantes en las clases y tipos del modelo de clases, para describir pre- y post-condiciones en las operaciones, para especificar mensajes y acciones, para especificar restricciones en las operaciones, para especificar reglas de derivación para atributos" [26].

Debido a que OCL es un lenguaje tipado (del inglés 'typed language'), es importante señalar que 'cada expresión OCL está escrita en el contexto de una instancia de un tipo específico. El tipo, es referido en una expresión a través de la palabra reservada *self*, la cual representa la instancia contextual' [26],[21]. Muchas de las métricas que definiremos conciernen a aquellos conceptos de OCL que permiten en una expresión usar propiedades pertenecientes a otros tipos (clases del diagrama) distintos de la instancia contextual que representa la expresión. A continuación definiremos dichos conceptos, el lector puede recurrir a [26] para una definición más completa de los mismos.

Navegaciones. Comenzando desde un objeto específico, es posible navegar una asociación en el diagrama de clases para referirse a otros objetos y sus propiedades (atributos y operaciones). Una relación es navegada cuando se utiliza en una expresión el nombre de rol del extremo opuesto de una relación que vincula la clase donde se define la expresión con otra clase del diagrama (o eventualmente si el rol es omitido en el modelo, el nombre de la clase asociada al extremo opuesto, en minúscula). Es

[2] El término "chunk" es un término en inglés que, de igual manera a los términos "chunking" y "tracing", al no haber encontrado un término adecuado en español para traducirlo se optó por mantener su nombre en su lenguaje original.

posible navegar varias relaciones para acceder a aquellas propiedades de otras clases que se necesiten en una expresión.

Colecciones. La manipulación de colecciones es una característica importante en sistemas orientados a objetos. OCL provee tres tipos concretos de colecciones: Conjunto (o Set), Bag, y Secuencia (o Séquence). Una colección Conjunto es una colección de elementos sin duplicados. Un Bag es similar a un Conjunto pero permite elementos duplicados. Una secuencia es similar a un Bag pero sus elementos están ordenados.

Parámetros de entrada, de salida y de entrada/salida, Valor de Retorno. Una operación puede incluir en su definición parámetros [26] de entrada, de salida, o de entrada/salida. Si las operaciones tienen parámetros de salida o entrada/salida, el resultado de esta operación es una tupla conteniendo todos estos tipos de parámetros y el valor de retorno.

Operaciones de Colecciones. OCL define muchas operaciones para manipular los elementos en una colección. Las operaciones permiten al modelador proyectar nuevas colecciones de las existentes.

Las operaciones como *select*, *reject*, *iterate*, *forAll* and *exists*, toman cada elemento en una colección y evalúan una expresión en ellos. La expresión evaluada para cada colección puede ser definida en término de una nueva navegación. Tendremos en cuenta aquellas expresiones de operaciones sobre colecciones que están definidas en término de otras navegaciones, proponiendo algunas métricas para estos casos.

Mensajes. La expresión de un mensaje es usada para especificar el hecho que un objeto ha enviado o enviará algún mensaje a otro objeto en algún momento en el tiempo [26].

Finalmente para concluir la presentación de conceptos que requieren de técnicas de "tracing", definiremos el concepto de clase de utilidad, el cual es una facilidad de UML y no de OCL, pero debido a que pueden ser utilizadas en una expresión y su utilización se relaciona al "tracing" la presentamos en esta sección.

Clases de Utilidad. Una clase de utilidad es un tipo de valor (del inglés 'value type') definido como un nuevo tipo en un diagrama de clases UML y utilizado como si fuera un tipo básico [11],[21].

Ejemplo: Usaremos una pequeña parte del ejemplo de Royal and Loyal (R&L) de [21] para analizar algunas expresiones y calcular en la próxima sección sus métricas. La parte superior de la Fig. 2 muestra una parte pequeña del diagrama de clase de R&L. La figura 2 también incluye en su parte inferior cinco expresiones OCL definidas en los tipos de las clases mencionadas.

- La expresión 1, 2 y 5 son ejemplos de expresiones invariantes mientras que las expresiones 3 y 4 representan pre- y post-condiciones respectivamente.
- *self^invalidate()* del invariante 1, es un ejemplo de un mensaje enviado a un objeto que representa la instancia contextual.
- *membership.card* del invariante 2 es un ejemplo de navegación de LoyaltyProgram to CustomerCard. La navegación hace uso de dos relaciones: una desde LoyaltyProgram a Membership (una clase asociación) y otra desde Membership a CustomerClass. En la primer relación no hay un nombre de rol asociado en extremo opuesto donde la clase Membership es la clase fuente de la asociación, por tal razón el nombre de la clase es usada, comenzando con minúscula. Mientras que, en la última asociación la navegación es representada por el nombre de rol 'card'.
- *self.customer* de la expresión invariante 2 es otro ejemplo de navegación donde la operación de colección *forAll* está especificada para expresar que todas las edades de los clientes de LoyaltyProgram deben ser mayores a 30 años.
- Date es una clase de utilidad, y sus atributos y operaciones son utilizadas en la expresión 1.

4. DEFINICIÓN DE MÉTRICAS PARA LA PROPIEDADES ESTRUCTURALES DE EXPRESIONES OCL

Para definir las métricas hemos seguido un método basado en dos propuestas [29], [30], que consta de varias etapas y que su utilización pretende asegurar la validez y la confiabilidad de las métricas. Aunque, como mencionamos en la introducción, en este artículo solo nos ocuparemos de la definición y de la validación teórica de las métricas.

Así como Fenton en [10] sugiere que no es recomendable definir una única métrica para la complejidad estructural, debido a que la complejidad está formada por muchos atributos específicos los cuales deben ser medidos separadamente, hemos definido un conjunto de métricas que capturan los diferentes aspectos de propiedades estructurales de expresiones OCL, relacionadas con las técnicas de "tracing".

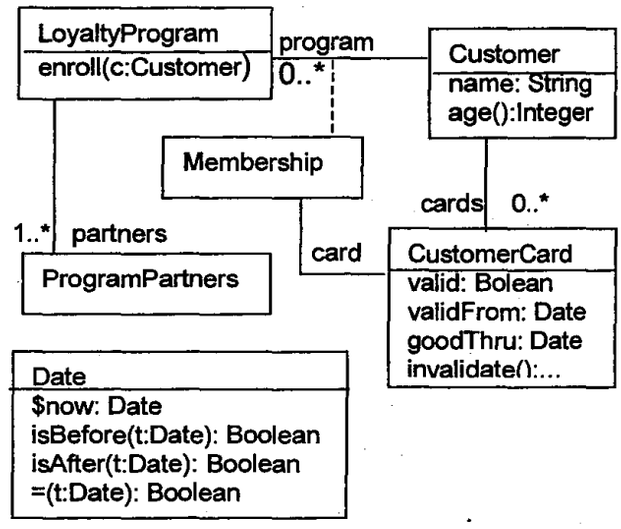
Utilizando la plantilla de GQM [31],[32] para la definición de objetivos, el objetivo perseguido para la definición de métricas para expresiones OCL es:

> Analizar *estructuras de expresiones OCL relacionadas con técnicas de "tracing"*
> Con el propósito de *Evaluar*
> Con respecto a su *Entendibilidad*
> Desde el punto de vista de *Modeladores y Diseñadores de software OO*
> En el contexto de *Organizaciones de desarrollo de software*

Así, nuestro objetivo es definir métricas para expresiones OCL midiendo las propiedades estructurales de expresiones OCL relacionadas con técnicas de "tracing", y luego determinar a través de la validación empírica si éstas métricas están relacionadas con su entendibilidad.

A continuación presentaremos cada una de las métricas que hemos definido y que pueden ser aplicadas a expresiones OCL. Para ilustrar su utilización para cada una de ellas presentaremos un ejemplo.

Métrica NNR: NÚMERO DE RELACIONES NAVEGADAS: Cuenta el número total de relaciones navegadas en una expresión. Una relación es navegada cuando se utiliza en una expresión el nombre de rol del extremo opuesto de una relación que vincula la clase donde se define la expresión con otra clase del diagrama (o eventualmente si el rol es omitido en el modelo, el nombre de la clase asociada al extremo opuesto). Si una relación se navega dos veces, por ejemplo para referenciar a dos diferentes atributos de una clase o interfaz, esta relación se cuenta una sola vez.




```
(expresión 1)
  context CustomerCard inv:
  validFrom.isBefore(goodThru) and
  goodThru.isAfter(Date.now)implies self^invalidate()

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

(expresiones 3 y 4 (pre- y post-cond. respectivamente))
  context LoyaltyProgram::enroll(c: Customer)
  pre: not customer->includes( c )
  post: customer = customer @ pre-> including ( c)

  (expresión 5 )
    context 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: Ejemplos de expr. OCL. Diagrama de R&L [33].

Ejemplo: En la expresión 1 de la Fig. 2, el valor de NNR= 0. En la expresión 2, NNR = 3.

Métrica NAN: NUMERO DE ATRIBUTOS REFERIDOS POR MEDIO DE NAVEGACIONES: Cuenta el número total de atributos referidos a partir de las navegaciones. Una navegación puede ser utilizada para referirnos a un atributo de otra clase, clase de asociación[3] o interfaz. Los atributos referidos más de una vez son contados una sola vez.

Ejemplo: En la expr. 2 de Fig. 2 sólo el atributo goodThru es utilizado, luego el valor de NAN =1.

Métrica WNO: NÚMERO PONDERADO DE OPERACIONES REFERIDAS POR MEDIO NAVEGACIONES: Se define como el número ponderando de operaciones que son referidas a partir de navegaciones. Las operaciones son ponderadas por el número de parámetros actuales de la operación (sólo es necesario especificar los parámetros de entrada y entrada/salida al invocar una operación [26]) y por la cantidad de valores resultantes accedidos (esto es, el resultado de la operación más la cantidad de parámetros de salida ó entrada/salida de la operación). La ponderación se efectúa de la siguiente forma:

$$\sum_{m' \text{ ref } N(\text{expresión})} (1 + |Par(m')|)(1 + R + |P_{out,\ in/out}\ (m')|)$$

donde:

N(expresión): Navegaciones de una expresión.

m' ref N(expresión): la operación m' es referida a partir de una navegación.

|Par(m')|: cantidad de parámetros actuales de la operación m'. R representa el resultado.

$|P_{out,\ in/out}\ (m')|$: Cantidad de parámetros de salida o entrada/salida de la operación m'. Debido a que en una navegación es posible acceder exclusivamente a un resultado por vez, esta ponderación da un mayor peso a aquellos operaciones referidas a partir de navegaciones que acceden a varios resultados.

Ejemplo:

Dada la siguiente definición de una operación llamada "income" y su post-condición OCL [26].

context Person::income(d: Date, bonus: Integer): Integer
post: result = type { bonus =...., result =....}

la operación tiene un tipo resultante de tipo Integer, un parámetro de entrada (d), y un parámetro de salida (bonus).

Ahora consideremos una expresión en la cual navegando a la clase Persona operamos con sus dos resultados, de la siguiente forma:

context Salary::calculate()
post: person.income(aDate).bonus + person.income(aDate).result

Si aplicamos la métrica WNO a la post-condición de la operación *calculate*, esta se obtiene de la siguiente forma:

WNO = (1 + 1) (1 + 1 + 1).

Métrica NNC: NÚMERO DE DIFERENTES CLASES, CLASES DE ASOCIACIÓN O INTERFACES A LAS CUALES SE HA NAVEGADO: Cuenta el número total de clases o interfaces a las cuales es posible navegar. Cuando es utilizada una navegación en la definición de una expresión, las clases e interfaces están involucradas, esta métrica tiene en cuenta sus cantidades. Si la clase que contiene la expresión tiene una relación reflexiva, y esta relación es navegada, se contará una sola vez a la clase. Además una clase puede ser alcanzable desde una clase inicial a partir de diferentes formas de navegación (es decir siguiendo diferentes relaciones), por ello si una clase es utilizada en más de una navegación esta clase será contada una sola vez.

Ejemplo: En el invariante 2 de la figura 2, el valor de NNC = 3.

Métrica WNM: NÚMERO PONDERADO DE MENSAJES: Cuenta el número de mensajes definidos en una expresión, donde los mensajes están ponderados en función de sus parámetros.

La ponderación se realiza de la siguiente forma.

$$\sum_{m' \in M(\text{expresión})} (1 + |Par(m')|)$$

donde:

M(expresión): Conjunto de operaciones correspondientes a los mensajes definidos en una expresión.

[3] del Inglés "association class".

|Par(m')|: cantidad de parámetros actuales de la operación m'.
Ejemplo: En el invariante 1 de Fig. 2, el valor de WNM es 1 para el único mensaje cuya operación es *invalidate* y no contiene parámetros.

Métrica NPT: NÚMERO DE PARÁMETROS CUYOS TIPOS SON CLASES O INTERFACES: Esta métrica es utilizada especialmente en pre- y post-condiciones para una operación, y toma en cuenta los parámetros formales de la operación que son utilizados en la definición de la expresión, cuyos tipos representan clases o interfaces definidas en el diagrama de clases.
Ejemplo: En la expresión 3 de la Fig. 2, el valor de NPT =1. Idem para la expresión 4.

Métrica NUCA: NÚMERO DE ATRIBUTOS CORRESPONDIENTES A UNA CLASE DE UTILIDAD: En una expresión OCL es posible utilizar una clase de utilidad. Esta métrica cuenta el número de atributos correspondientes a una clase de utilidad que es utilizada en una expresión (independientemente de la clase de utilidad a la que pertenezcan). Los atributos son contadas una sola vez, si ellos son utilizadas más de una vez.
Ejemplo: En el invariante 1 de Fig. 2, el valor NUCA=1.

Métrica NUCO: NÚMERO DE OPERACIONES CORRESPONDIENTES A UNA CLASE DE UTILIDAD: En una expresión es posible utilizar una clase de utilidad. Esta métrica cuenta el número de operaciones correspondientes a una clase de utilidad que es utilizada en una expresión (independientemente de la clase de utilidad a la que pertenezcan). Las operaciones son contadas una sola vez, si ellas son utilizadas más de una vez.
Ejemplo: En el invariante 1 de Fig. 2, el valor NUCO=2.

Métrica WNN: CANTIDAD PONDERADA DE NAVEGACIONES: Como se explicó en la sección previa, una operación de colección está compuesta de una expresión que es evaluada para cada elemento de la colección. Si la expresión evaluada involucra una nueva navegación (o muchas) daremos un mayor peso a las nuevas navegaciones usadas en la definición de la navegación mas externa.
Ejemplo: En la expresión 3 de la Fig 2 el valor de WNN es 1, existe solo una navegación. Ahora, mostraremos como se obtiene WNN en la expresión 5 de la Fig. 2. Dos subexpresiones están conectadas por un operador 'and'. Cada subexpresión envuelve navegaciones. Mientras que la primer navegación no incluye en su evaluación una nueva navegación, la segunda utiliza una operación de colección definida en término de otra (aunque ellas representan la misma navegación *self.customer*) el valor de WNN se obtiene en la siguiente forma:
*1** 2 + *2** 1 = 4
El número mostrado en itálica representa el *peso* aplicado, y el número mostrado en fuente normal indica el número de navegaciones. Como las operaciones de colección pueden ser definidas recursivamente, llamaremos "nivel" a diferentes composiciones de navegaciones, en el caso de que una navegación B es utilizada en la definición de una operación de colección para una navegación A, diremos que B está en el nivel 2 y A en el nivel 1.
El peso asociado con cada nivel es igual al nivel. La definición de WN es:
WNN = Σ (peso del nivel)* (número de navegaciones en el nivel).

Métrica DN: PROFUNDIDAD DE LA NAVEGACIONES: Dada una expresión OCL pueden ser utilizadas muchas navegaciones en su definición, construiremos un árbol de navegación usando las clases que son navegadas. La raíz del árbol es el nombre de la clase (o interfaz) que representa *self*. Self es siempre el punto inicial de cualquier expresión (self puede estar implícito en una expresión). Construiremos una rama para cada navegación, donde cada clase 'a la cual navegamos' es un nodo en la rama. Los nodos son conectados por 'arcos de navegación'. DN es la profundidad máxima del árbol.
Ejemplo: En la expresión invariante 2 de la Fig. 2 definida para LoyaltyProgram hay dos navegaciones. El árbol construído usando el

método descripto antes es mostrado en la figura 3 (a). La profundidad DN es 2

Cuando una navegación incluye una expresión de una operación colección definida en término de una (o más) navegación, construiremos un nuevo árbol para las navegaciones utilizadas en la expresión de la operación de colección, usando el mismo método descrito anteriormente, luego los conectaremos con un arco de *conexión de definición.*

De acuerdo a la expresión 5 de Fig. 2, el árbol construído usando el método es el mostrado en fig 3 (b). El arco punteado representa una conexión de definición. Para computar la profundidad del árbol en la métrica DN aplicaremos la siguiente regla:

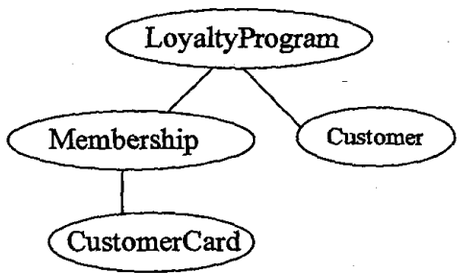


Fig 3 (a)

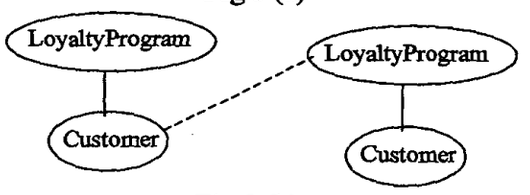


Fig 3 (b)

"Una conección de navegación es contada una sola vez, y la conexión de definición es contada dos veces". El DN para el invariante 5 de la Fig. 2 es igual a 4.

Métrica WCO. CANTIDAD PONDERADA DE OPERACIONES COLECCIÓN.
Definición: Como explicamos en la sección previa, una operación de colección está compuesta de una expresión que es evaluada para cada elemento de la colección, si la expresión evaluada envuelve una nueva navegación (o más de una) con una nueva operación de colección daremos un mayor peso a las operaciones de colección utilizadas en la definición de la operaciones colección más externas.
WCO= $\sum$ peso del nivel * cantidad de operaciones colección del nivel.

5. VALIDACIÓN TEÓRICA DE LAS MÉTRICAS PROPUESTAS.

Como ya mencionamos 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 [29].

Para realizar la validación téorica de las métricas propuestas utilizaremos dos aproximaciones basadas en propiedades:

- el marco formal de Briand et al. [27], [28] 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. [13] en el cual se definen propiedades de métricas basadas en la interacción para acoplamiento y cohesión de un diseño de alto nivel.

Debido a limitaciones de espacio mostraremos un ejemplo de validación de una métrica de cada tipo y el resto las presentaremos de forma resumida en la Tabla 1.

Propiedades de NAN como Métricas de Acoplamiento Basado en la Interacción (según Briand et. al. [13]).

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 NAN (Número de Atributos referidos por medio de navegaciones) es una métrica de acoplamiento basado en la interacción [13] tendremos en cuenta que: una

expresión OCL será considerada una parte individual de software, y un conjunto de *atributos a los cuales se navega en dicha expresión* (posiblemente vacío) será considerado el sistema de software asociado a dicha parte. La navegación 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*.

Solo hemos definido métricas relacionadas con acoplamiento importado. Nuestra hipótesis es similar a la hipótesis ISP de [13]: "a *mayor número de partes de software usadas, mayor será el contexto que requiere ser entendido, y mayor probabilidad de ocurrencia a fallas*"

Las propiedades de acoplamiento basado en interacción son instanciaciones, para nuestro contexto de expresiones OCL, de las propiedades definidas en [27],[28] para acoplamiento.

- **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 NAN, podemos decir que si una expresión *sp* (la parte del software que consideramos) no contiene navegaciones que refieran a un atributo entonces NAN(*sp*) = 0. Por otro lado, la métrica NAN 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 navegación que refiere a un atributo, y obtenemos una nueva expresión modificada, digamos *sp'*, pueden suceder dos casos: (1) el atributo al cual nos referimos por medio de la nueva navegación agregada 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 por medio de la nueva navegación agregada 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 NAN 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 NNR, NAN, WNO, NNC, WNM, NPT, NUCA, NUCO son métricas de acoplamiento basadas en interacción.

Propiedades de WNM como Métrica de Tamaño (Marco de Briand et al. [27],[28]).

- **No-negatividad.** se prueba directamente, es imposible obtener un valor negativo.
- **Valor nulo.** Una expresión OCL *e* sin mensajes tiene un WNM(*e*) = 0
- **Aditividad de Módulo:** Cuando dos expresiones son unidas y estas expresiones no cuentan en su definición con mensajes en común, la nueva expresión (producto de la unión) tiene tantos mensajes como cada una de las expresiones unidas. Pero si las expresiones originales tienen algún mensaje en común el WNM de la expresión resultante debería ser menor que el WNM de las expresiones originales.

Otras métricas que cumplen las propiedades de tamaño son: WNN, WCO.

Propiedades DN como Métrica de Longitud (Marco de Briand et al. [27],[28]).

Las propiedades de **No-negatividad** y **Valor nulo** se satisfacen inmediatamente, la

profundidad del árbol no puede ser nunca negativa, y una expresión sin navegación tiene un árbol vacío (DN es igual a 0).

- **Monotonicidad no incremental para componentes conectados:** Si agregamos relaciones entre elementos de un árbol (clases o interfaces) la profundidad no varía.
- **Monotonicidad no decremental para componentes no-conectados:** Agregar una relación a dos componentes no conectados (dos árboles) hace que ellos se conecten y su longitud no es menor que la longitud de los dos componentes conectados.
- **Módulos Disjuntos:** La profundidad de un árbol está dada por el componente que tiene menos niveles desde la raíz a las hojas.

Clasificación según Briand et al. [13],[27],[28].	**Métricas para expresiones OCL**		
	NNR, NAN, WNO, NNC, WNM, NPT, NUCA, NUCO.	DN	WNN, WCO, WNM
Acoplamiento basado en la interacción	X		
Tamaño			X
Longitud		X	

Tabla 1: Validación según el marco de Briand et al. [13], [27], [28].

CONCLUSIONES Y TRABAJO FUTURO.

En este artículo hemos presentado un conjunto de métricas y las hemos validado teóricamente siguiendo los marcos formales de Briand et al. [13], [27], [28]. Ellas pueden ser aplicadas a expresiones OCL especificadas en diagramas de clases UML y refieren a las propiedades estructurales de expresiones OCL, considerando aquellos conceptos de OCL que requieren técnicas de "tracing" durante la entendibilidad de expresiones OCL. Utilizando tales métricas podremos:

- Realizar comparaciones cuantitativas de alternativas de diseño y sus especificaciones, y por consiguiente una selección objetiva entre diversas alternativas de modelos conceptuales (en nuestro caso, diferentes especificaciones en OCL de un diagrama de clase) con contenido semántico equivalente.
- Estimar la carga cognitiva aplicada por los modeladores.
- Predecir características de calidad externas, como entendibilidad y mantenibilidad en fases iniciales del ciclo de vida, y una mejor asignación de recursos en base a estas predicciones.

Somos concientes que esta es una primera propuesta de métricas, y que para poder proveer un conocimiento acabado sobre la utilidad de las mismas es necesario someterlas a un proceso de validación empírica. Cómo mencionan varios autores, la validación empírica a través de experimentos o casos de estudios es fundamental para asegurar que las métricas son realmente útiles en la práctica. Actualmente estamos planificando un experimento controlado para mostrar evidencia de que las métricas que proponemos realmente pueden ser indicadores de la entendibilidad de una expresión OCL.

Como trabajo futuro, trabajaremos en la generalización de los beneficios de este conjunto de métricas, intentando obtener la complejidad "global" de una clase debido al uso de expresiones OCL. Además es necesario ampliar este conjunto de métricas con la definición de otras que estén relacionadas a otros aspectos de la propiedades intrínsecas a una expresión, por ejemplo el uso de iteradores en operaciones colección, expresiones 'let' para definir variables reutilizables, expresiones 'if' etc.

AGRADECIMIENTOS

Esta investigación es parte de los siguientes proyectos: 1) DOLMEN financiado por la

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), y 3) UNComa 04/E048 (Modelado de Componentes Distribuídos Orientados a Objetos).

BIBLIOGRAFÍA Y 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] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

[8] 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.

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

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

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

[12] 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.

[13] L. C. Briand, S. Morasca y V. Basili. "Defining —and validating measures for object-based high level design", *IEEE Transactions on Software Engineering*. 25 (5), setiembre, 1999, pp. 722-743.

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

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

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

[17] M. Genero, M. Piattini y C. Calero, "Early Measures For UML class diagrams",

L'Objet, 6(4), Hermes Science Publications, enero, 2000, pp. 489-515.

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

[19] Object Modeling with the OCL. The Rationale behind the Object Constraint Language. Lecture Notes in Computer Science 2263. Springer, 2001.

[20] 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.

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

[22] L. C. Briand, J. Wüst, S. Ikonomovski y H. Lounis, "A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: an Industrial Case Study". *International Software Engineering Research Network TR ISERN-98-29,* 1998.

[23] 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.

[24] 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.

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

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

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