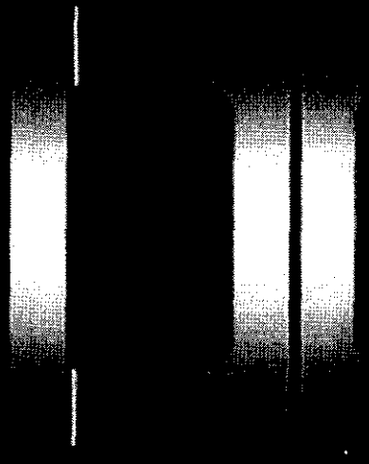
I+D Computación

Una publicación de la Academia de Posgrado de Ciencias Computacionales



Noviembre de 2002, Vol. 1 No. 2

ISSN 1665-238X

Calidad de Modelos Conceptuales

Marcela Genero, Mario Piattini y Coral Calero
Grupo de Investigación Alarcos, E.S. Informática,
Paseo de la Universidad, 4, 13071 Ciudad Real (España)
{Marcela.Genero, Mario.Piattini, Coral.Calero}@uclm.es

Abstract.– In this paper, the second of a series dedicated to analyze different aspects of database and information systems quality, we present the main proposals in the area of metrics for object oriented conceptual models.

Keywords: database quality, conceptual modelling, databases.

1. INTRODUCCIÓN

Actualmente, el desarrollo de software es una tarea cada vez más compleja debido a que los proyectos son más grandes, más costosos, llevan más tiempo y son más difíciles de administrar. El paradigma de la orientación a objetos (OO) ha surgido como una solución para el diseño y desarrollo de software, y su adopción se ha ido imponiendo en la industria. Este paradigma introduce nuevos conceptos y abstracciones como clases, métodos, mensajes, herencia, polimorfismo, agregación y encapsulación, los cuales requieren que los diseñadores de software OO deban estructurar cuidadosamente el diseño, que luego será implementado.

En los últimos años ha habido una explosión de propuestas de metodologías de análisis y diseño, lenguajes y sistemas de gestión de bases de datos basados en el paradigma OO. El Lenguaje de Modelado Unificado (UML) propuesto por Booch et al. [4] ha surgido como un lenguaje estándar para expresar los distintos modelos necesarios para desarrollar software OO. El uso de UML se ha extendido ampliamente, ha sido aplicado exitosamente para construir sistemas para desarrollar una amplia gama de tareas como, por ejemplo: comercio electrónico, comando y control, juegos de computadoras, dispositivos médicos electrónicos, telefonía, robótica, aviación, etc. [5] y también se utiliza cada vez más para el modelado conceptual de bases de datos [3, 27, 28]. Sin embargo, el hecho de que exista un nuevo estándar de modelado, no asegura la calidad de los modelos producidos a largo del proceso de desarrollo de software, por lo tanto es necesario evaluar tal calidad.

La medición del software es ampliamente reconocida como un medio efectivo para entender y mejorar el desarrollo de software y proyectos de mantenimiento [6], para mantener la calidad de los sistemas [22], para focalizar las áreas problemáticas [13], y para determinar la mejor manera de ayudar a los profesionales e investigadores en su trabajo [29].

Si bien en la literatura existente podemos encontrar métricas para software OO algunas pueden sólo aplicarse una vez que el producto está terminado, o casi completo, ya que se basan en información extraída de la implementación del producto (código) o del diseño avanzado. Esto proporciona la información demasiado tarde para ayudar a mejorar las características internas del producto antes de su

Artículo recibido el 3 de julio de 2002.

ISSN 1665-238X

finalización. Por eso hay una gran necesidad de contar con métricas y modelos que puedan aplicarse en las etapas tempranas del desarrollo, especialmente durante el modelado conceptual de la base de datos. Este enfoque de la medición puede brindarles a los desarrolladores una oportunidad para solucionar problemas, eliminar atributos de diseño no correctos, y eliminar la complejidad no deseada desde etapas tempranas del ciclo de desarrollo. Esto llevaría a una reducción considerable del trabajo extra necesario durante y después de la implementación.

En este artículo resumiremos un conjunto de métricas OO extraídas de la literatura existente. Solamente se tendrán en cuenta aquellas métricas que se pueden aplicar a los modelos conceptuales OO (diagramas de clases) en una etapa de diseño de alto nivel. Consideramos que en este tipo de etapa, un diagrama de clases consta de los siguientes elementos:

- Paquetes
- Clases
- Cada clase tiene atributos y operaciones
- Las operaciones sólo tienen su signatura (el nombre de las operaciones y sus parámetros)
- Relaciones: asociaciones, agregaciones, generalizaciones y dependencias.

2. MÉTRICAS DE CHIDAMBER Y KEMERER

En [13] propusieron seis métricas para medir la complejidad del diseño OO, llamadas MOOSE, de las cuales solamente tres se pueden aplicar a nivel conceptual (ver tabla 1). Quizá sean las métricas más conocidas en este ámbito, aunque su definición ha sido criticada en varias oportunidades por ser imprecisas y ambiguas.

Tabla 1. Métricas MOOSE aplicadas en una etapa de diseño de alto nivel.

Nombre	Definición
WMC	La métrica WMC (*Weighted Methods per Class*) se define como: $$WMC = \sum_{i=1}^{n} c_i$$ Donde c_1, ..., c_n es la complejidad de los métodos de una clases con métodos M_1, ...,M_n. Si consideramos la complejidad de los métodos como la unidad, el WMC=n, número de métodos.
DIT	La métrica DIT (*Depth of Inheritance Tree*) es la profundidad del árbol de herencia de una clase. En los casos en los de herencia múltiple, DIT será la máxima longitud desde el nodo, hasta la raíz del árbol. DIT es una medida de cómo varias clases ancestro pueden afectar potencialmente a una clase. Esta métrica fue propuesta como una medida de complejidad de una clase, complejidad de diseño y potencial reusabilidad. Se basa en la idea de que cuanto más profunda esté una clase en la jerarquía, mayor será el número de operaciones que tienda a heredar.
NOC	La métrica NOC (*Number of Children*) es el número de subclases inmediatas subordinadas a una clase en la jerarquía de clases. Esta es una medida de cuántas subclases van a heredar las operaciones de la clase padre.

En [1] han realizado la validación empírica de la métrica DIT, concluyendo que cuanto mayor es el valor de DIT, mayor es la probabilidad de encontrar un fallo. También observaron que cuanto mayor es NOC, más pequeña es la probabilidad de encontrar un fallo. En [12] encontraron una

correlación positiva entre la métrica DIT y el número problemas encontrados por el usuario. En [25] mostraron a través de un estudio empírico que las métricas propuestas por Chidamber y Kemerer parecían ser adecuadas para la predicción de la frecuencia de cambios en las clases durante la fase de mantenimiento.

En relación con la validación teórica, en [13] se corroboró que tanto DIT como NOC satisfacen los axiomas de Weyuker [33]. En [6] clasificaron la métrica DIT como una medida de longitud, y la métrica NOC como una medida de tamaño. En [34] se demostró que tanto DIT como NOC se encuentran por encima de la escala ordinal, mientras que en [30] demostraron mediante el marco formal DISTANCE que podían ser caracterizadas en una escala ratio.

3. MÉTRICAS DE BRITO E ABREU Y CARAPUÇA

En [7] propusieron las métricas MOOD (ver tabla 2), para medir los principales mecanismos del paradigma OO, tales como, encapsulamiento, herencia, polimorfismo y paso de mensajes, y polimorfismo y su consecuente influencia sobre la calidad del software y la productividad en el desarrollo. Las métricas MOOD se pueden utilizar en las fases de diseño y se definieron para ser aplicadas a nivel de diagrama de clases.

Tabla 2. Métricas MOOD que pueden aplicarse en las etapa de diseño de alto nivel.

Nombre	Definición
MHF	El Factor de Ocultamiento de los Métodos (*Method Hiding Factor*) se define como el cociente entre la suma de las invisibilidades de todos los métodos definidos en todas las clases y el número total de métodos definidos en el sistema considerado. La invisibilidad de un método es el porcentaje del total de clases desde las cuales el método es invisible. $$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)} \quad (1)$$ $$V(M_{mi}) = \frac{\sum_{j=1}^{TC} es\_visible(M_{mi}, C_j)}{TC - 1} \quad (2)$$ $$es\_visible(M_{mi}, C_j) = \begin{cases} 1 \Leftrightarrow J \neq i \wedge C_j \text{ puede llamar a } M_{mi} \\ 0 \quad \text{caso contrario} \end{cases} \quad (3)$$ Donde: TC = número total de clases en el sistema considerado; $M_d(C_i)$ = $M_v(C_i)+M_h(C_i)$ = métodos definidos en C_i, $M_v(C_i)$ = métodos visibles en la clase C_i (métodos públicos), $M_h(C_i)$ = métodos ocultos en la clase C_i (métodos protegidos y privados). En otras palabras, MHF es la proporción entre los métodos definidos como protegidos o privados y el número total de métodos. MHF se propone como una medida de encapsulación, cantidad relativa de información oculta.
AHF	El Factor de Ocultamiento de los Atributos (*Attribute Hiding Factor*) se define como el cociente entre la suma de las invisibilidades de todos los atributos definidos en todas las clases y el número total de atributos definidos en el sistema considerado. La invisibilidad de un atributo es el porcentaje del total de clases desde las cuales los atributos son invisibles.

	$$AHF = \frac{\sum_{i=1}^{TC}\sum_{m=1}^{A_d(Ci)}(1-V(A_{mi}))}{\sum_{i=1}^{TC}A_d(C_i)} \quad (4)$$ $$V(A_{mi}) = \frac{\sum_{j=1}^{TC} es\_visible(A_{mi}, C_j)}{TC-1} \quad (5)$$ $$es\_visible(A_{mi}, C_j) = \begin{cases} 1 \Leftrightarrow J \neq i \wedge C_j \text{ puede llamar a } A_{mi} \\ 0 \qquad \text{caso contrario} \end{cases} \quad (6)$$ Donde: $A_d(C_i)$ = $A_v(C_i)$+(C_i) = atributos definidos en C_i, $A_v(C_i)$ = atributos visibles en la clase C_i, $A_h(C_i)$ = atributos ocultos en la clase C_i (atributos públicos), $M_h(C_i)$ = atributos privados en la clase C_i (atributos privados y protegidos). AHF se definió como una medida de encapsulación.
MIF	El Factor de Herencia de los Métodos (*Method Inheritance Factor*) se define como el cociente entre la suma de los métodos heredados en todas las clases del sistema considerado y el número total de métodos existentes (tanto los definidos localmente como los heredados) en todas las clases. $$MIF = \frac{\sum_{i=1}^{TC}\sum_{m=1}^{A_d(Ci)}(1-V(A_{mi}))}{\sum_{i=1}^{TC}A_d(C_i)} \quad (7)$$ Donde: $M_a(C_i) = M_d(C_i)+M_i(C_i)$ = métodos existentes en C_i (aquellos que pueden ser invocados en asociación con C_i), $M_d(C_i)+M_o(C_i)$ = métodos definidos en la clase C_i (aquellos declarados en C_i), $M_n(C_i)$ = métodos nuevos en la clase C_i (aquellos declarados dentro de C_i y que no invalidan a los otros heredados), $M_o(C_i)$ = Métodos redefinidos en la clase C_i (aquellos declarados dentro de C_i que redefinen a los heredados), $M_i(C_i)$ = métodos heredados en la clase C_i (aquellos heredados (y no redefinidos) en C_i). MIF se define como una medida de herencia, y por lo tanto como una medida del nivel de reusabilidad.
AIF	El Factor de Herencia de los Atributos (*Attribute Inheritance Factor*) se define como el cociente entre la suma de los atributos heredados en todas las clases del sistema considerado y el número total de atributos existentes (tanto los definidos localmente como los heredados) en todas las clases. $$AIF = \frac{\sum_{i=1}^{TC}A_i(C_i)}{\sum_{i=1}^{TC}A_a(C_i)} \quad (8)$$ Donde: $A_a(C_i) = A_d(C_i)+A_i(C_i)$ = atributos existentes en C_i (aquellos que pueden ser manipulados en asociación con C_i), $A_d(C_i) = A_n(C_i)+A_o(C_i)$ = atributos definidos en la clase C_i (aquellos declarados en C_i), $A_n(C_i)$ = nuevos atributos en la clase C_i (aquellos declarados dentro de C_i que no anulan a los atributos heredados), $A_o(C_i)$ = atributos redefinidos en la clase C_i (aquellos declarados dentro de C_i que redefinen a los atributos heredados), $A_i(C_i)$ = atributos heredados en la clase C_i (aquellos heredados y no redefinidos en C_i). Al igual que MIF, AIF se considera como un medio para expresar la capacidad de reutilización en un sistema.

PF	El Factor de Polimorfismo (*Polymorphism Factor*) se define como el cociente entre el número actual de posibles diferentes situaciones de polimorfismo, y el número máximo de posibles situaciones distintas de polimorfismo para la clase C_i. $$PF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} [M_n(C_i \times DC(C_i)]} \quad (9)$$ Donde: $M_o(C_i)$ = métodos redefinidos en la clase C_i, $M_n(C_i)$ = métodos nuevos en la clase C_i, $DC(C_i)$ = número de descendientes de la clase C_i. PF es una medida del polimorfismo y una medida indirecta de la asociación dinámica en un sistema.

Después de llevar a cabo un estudio empírico, en [8] sugirieron que:

- Cuando el valor de MHF aumenta, la densidad de defectos y el esfuerzo requerido para corregirlos, tendría que disminuir.
- Cuando el valor de MIF aumenta, la densidad de defectos y el esfuerzo requerido para corregirlos, tendría que disminuir.
- Idealmente, el valor de la métrica AHF sería 100%, por ejemplo todos los atributos estarían ocultos y solo podrían ser accedidos desde los métodos de las clases correspondientes.
- A primera vista podría resultar tentador pensar que la herencia debería ser usada extensivamente. Sin embargo, la excesiva reusabilidad a través de la herencia hace a los sistemas más difíciles de comprender y mantener.
- En relación con la métrica PF, en algunos casos los métodos redefinidos pueden contribuir a reducir la complejidad e incluso a hacer el sistema más comprensible y fácil de mantener.

En [23] han usado el marco de Kitchenham et al. [24] para validar estas métricas. Todas las métricas resultaron válidas menos PF, ya que presenta una discontinuidad en aquellos sistemas sin herencia.

4. MÉTRICAS DE LORENZ Y KIDD

En [26] propusieron un conjunto de métricas llamadas "métricas de diseño", que se refieren a características estáticas del diseño de un producto software. Estos autores clasificaron las métricas de la siguiente forma:

- Métricas de tamaño
- Métricas de herencia
- Métricas de características internas de las clases

En la tabla 3 se muestran sólo aquellas métricas que pueden aplicarse a un diseño de alto nivel.

Tabla 3. Métricas de Lorenz y Kidd.

	Nombre	**Definición**
Métricas de tamaño	PIM	El Número de Métodos de Instancia Públicos de una clase es el número total de métodos públicos de instancias. Los métodos públicos son aquellos que están disponibles como servicios para otras clases. Lorenz y Kidd [26] sugirieron que ésta métrica es una buena medida de la cantidad de responsabilidad que tiene una clase.
	NIM	El Número de Métodos de Instancia es la suma de todos los métodos de una clase, considerando tanto los métodos públicos como los protegidos y privados. Según Lorenz y Kidd [26] esta métrica es una buena medida de la cantidad de colaboración utilizada.
	NIV	El Número de Variables de Instancia es el número total de variables a nivel de instancia que tiene una clase, considerando las variables privadas y protegidas disponibles en una clase.
	NCM	El Número de Métodos de Clase es el número total de métodos a nivel de clase, i.e. aquellos métodos que son globales a todas las instancias que tiene una clase.
	NVV	El Número de variables de Clase es el número total de variables a nivel de clase que tiene una clase.
Métricas de herencia	NMO	El Número de Métodos Sobreescritos es el número total de métodos sobreescritos por una subclase. Esta métrica se definió para medir la calidad del uso de la herencia, ya que examina la relación de herencia entre subclases y superclases.
	NMI	El Número de Métodos Heredados es el número total de métodos que una clase hereda. Esta métrica se definió para medir la calidad del uso de la herencia, ya que examina la relación de herencia entre subclases y superclases.
	NMA	El Número de Métodos Añadidos es el número total de métodos que se definen en una subclase. Esta métrica se definió para medir la calidad del uso de la herencia, ya que examina la relación de herencia entre subclases y superclases.
	SIX	El Indice de Especialización para cada clase se define así: $\frac{\textit{Número de Métodos Sobreescritos} * \textit{Nivel de Anidamiento en la Jerarquía}}{\textit{Número Total de Métodos}}$ Está métrica mide hasta qué punto una subclase redefine el comportamiento de una superclase.
Métricas de características internas de las clases	APPM	El Promedio de Parámetros por Método se define así: $\frac{\textit{Número Total de Parámetros por Método}}{\textit{Número Total de Métodos}}$

De acuerdo a nuestro conocimiento, no se han publicados trabajos sobre la validación teórica de estas métricas.

Después de aplicar estas métricas a cinco proyectos reales (escritos en Smalltalk y C++), Lorenz and Kidd [26] propusieron ciertas recomendaciones, alguna de las cuales citamos a continuación:

- Una jerarquía poco profunda o demasiado profunda tiene repercusión directa sobre la calidad del diseño.
- Ningún método de instancias o muchos métodos de instancia puede indicar una distribución de responsabilidades poco óptima (en relación con la métrica NIM).
- Un gran número de variables de instancia puede indicar demasiado acoplamiento con otras clases, lo que reduce el reuso (en relación con la métrica NIV).

- El promedio de variables de clases debería ser bajo. En general debe haber menos variables de clase que variables de instancia.
- Demasiados métodos a nivel de clase puede indicar el uso inapropiado de las clases para hacer el trabajo en lugar de instancias (en relación con la métrica NCM).
- Métodos sobreescritos, especialmente en nivele muy profundos de la jerarquía de herencia pueden indicar un diseño pobre de la jerarquía (en relación con la métrica NMO).
- El Índice de Especialización para cada puede ser muy útil para identificar clases teniendo en cuenta su ubicación dentro de la jerarquía y aquellas que puedan traer problemas en el diseño.
- Ellos también sugieren un valor límite de 0,7 para el número de métodos promedio por clase (en relación con la métrica APPM).

5. MÉTRICAS DE GENERO ET AL.

En [17, 21] propusieron un conjunto de métricas para medir la complejidad estructural de los diagramas de clases debido al uso de relaciones UML, como la agregaciones, asociaciones y dependencias. Estas métricas se agrupan en métricas a nivel de diagrama de clases (ver tabla 4) y métricas a nivel de clase (ver tabla 5).

Tabla 4. Métricas a nivel de diagrama de clases .

Nombre	Definición
NAssoc	La métrica Número de Asociaciones se define como el número total de asociaciones dentro de un diagrama de clases.
NAgg	La métrica Número de Agregaciones se define como el número total de relaciones de agregación dentro de un diagrama de clases (cada par todo-parte en una relación de agregación).
NDep	La métrica Número de Dependencias se define como el número total de relaciones de dependencia dentro de un diagrama de clases.
NGen	La métrica Número de Generalizaciones se define como el número total de relaciones de generalización dentro de un diagrama de clases (cada par padre-hijo en una relación de generalización).
NGenH	La métrica Número de Jerarquías de Generalización se define como el número total de jerarquías de generalización dentro de un diagrama de clases.
NAggH	La métrica Número de Jerarquías de Agregación se define como el número total de jerarquías de agregación dentro de un diagrama de clases.
MaxDIT	La métrica Máximo DIT se define como el máximo entre los valores DIT obtenidos de cada clase del diagrama de clases. El valor de DIT para una clase dentro de una jerarquía de generalización es la longitud de la ruta más larga partiendo de la clase a la clase raíz de la jerarquía.
MaxHAgg	La métrica Máximo HAgg se define como el máximo de los valores HAgg obtenidos de cada clase del diagrama de clases. El valor HAgg para una clase dentro de la jerarquía de agregación es la longitud de la ruta más larga desde la clase hasta las hojas.

Tabla 5. Métricas a nivel de clases.

Nombre	Definición
NAssosC	La métrica Número de Asociaciones por Clase se define como el número total de asociaciones que una clase tiene con otras clases o con ella misma.
HAgg	La Altura de una clase dentro de una jerarquía de agregación se define como la longitud de la ruta más larga desde la clase a las hojas.
NODP	La métrica Número de Partes Directas de una clase se define como el número total de

	"Partes Directas" que contiene una clase que pertenece a una jerarquía de agregación.
NP	La métrica "Número de Partes" se define como el número de clases "Partes" (directas o indirectas) de una clase "Todo".
NW	La métrica "Número de Todos" se define como el número de clases "Todos" (directas o indirectas) de una clase "Parte".
MAgg	La métrica "Agregación Múltiple" se define como el número de clases "Todos" directas que tiene una clase en una jerarquía de agregación.
NDepIn	La métrica "Número de Dependencias In" se define como el número de clases que dependen de una clase dada.
NDepOut	La métrica "Numero de Dependencias Out" se define como el número de clases de las que la clase dada depende.

Con respecto a la validación teórica, podemos decir que:

En [16] fueron validadas usando el marco de Briand et al. [6], concluyendo que las métricas NP, NW, NOPD y MAgg son métricas de tamaño; utilizando el marco de Zuse [34] concluyendo que la mayoría de las métricas están por encima de la escala ordinal, y utilizando el marco DISTANCE [30, 31] concluyendo que las métricas miden los atributos que pretenden medir y que además están en la escala de ratio.

Se realizaron varios experimentos controlados para validar empíricamente las métricas a nivel de diagramas de clases [18-21] concluyendo que la mayor parte de las mismas puede considerarse como indicadores válidos de la calidad de los modelos de clases.

6. CONCLUSIONES

La tabla 6 resume las características más importantes de las principales propuestas sobre métricas para modelo de clases existentes en la literatura. La primera columna hace referencia a la fuente de las métricas. En la segunda, se presenta el enfoque de las métricas (normalmente se refieren a la complejidad). La tercera columna se refiere al alcance de las métricas: el modelo conceptual completo o un elemento simple del modelo. La cuarta columna muestra si las métricas son objetivas o subjetivas[1], por ejemplo, si se calculan mediante un método objetivo o mediante uno subjetivo (normalmente puntuaciones dadas por los usuarios o participantes). La quinta y sexta columnas reflejan si existen estudios publicados en los que se haya realizado la validación teórica o empírica de las métricas. La última columna refleja si existe una herramienta automática para el cálculo de las métricas.

Podemos concluir que el trabajo sobre métricas modelos conceptuales OO es escaso y aún no está consolidado. Incluso, aunque algunas de las métricas hayan sido validadas teóricamente, cada autor siguió diferentes marcos de trabajo, considerando diferentes propiedades o axiomas. Algunos usan aproximaciones basadas en propiedades como las de Briand et al. [6] o en los axiomas de Weyuker [33], mientras que otros aproximaciones basadas en a teoría de la medida como el marco formal de Zuse [34] o el marco DISTANCE [30, 31]. Esto ocurre debido a que aún no existe un estándar de cómo validar teóricamente una medida.

[1] Cuando se miden atributos de entidades, nos esforzamos en mantener nuestras medidas objetivas. Aun así, es importante reconocer que las medidas subjetivas pueden ser útiles, siempre y cuando entendamos la imprecisión.

Tabla 6. Resumen de las métricas definidas para diagramas de clases.

Autores	Enfoque	Alcance	Objetivas/ Subjetivas	Validación Teórica	Validación Empírica	Herramienta
Chidamber y Kemerer [13]	Complejidad	Clase	Objetivas	SI	Parcialmente	SI
Brito e Abreu y Carapuça [7]	Características OO	Diagrama de Clases	Objetivas	SI	Parcialmente	SI
Lorenz y Kidd [26]	Características estáticas del diseño OO	Clase/ Diagrama de Clases	Objetivas	NO	Parcialmente	SI
Genero et al. [17, 21]	Complejidad estructural debido al uso de relaciones	Clase/ Diagrama de Clases	Objetivas	SI	Parcialmente	SI

Los experimentos son útiles para probar la validación empírica de las métricas, pero su replicación[2] interna y externa es necesaria [2, 11], para obtener resultados consistentes. Como Brooks et al. [11] señalan, el uso de experimentos precisos y repetibles es el "sello de calidad" de una disciplina científica o de ingeniería madura.

En relación con la validación teórica, pensamos que el primer paso es llegar a un acuerdo común en la comunidad de medición del software sobre la forma correcta de validar las métricas teóricamente. Solo entonces habrá un criterio consolidado y será posible demostrar la validación teórica de las métricas.

Respecto a la validación empírica, queremos apuntar que aunque los experimentos son útiles para obtener una primera aproximación para la obtención de resultados, es también necesario contar con datos sobre "proyectos reales", con la finalidad de obtener conclusiones reales y mejorar nuestro entendimiento sobre la calidad en el modelado conceptual. Sin embargo la escasez de tales datos continua siendo un grave problema, que debemos resolver cuando intentamos validar las métricas. Brito e Abreu et al. [9] sugirieron la necesidad de contar con un repositorio público sobre experiencias en medición, lo cual pensamos que podría ser un paso importante para lograr el éxito en la realización de todos los trabajos relacionados con la medición del software. Además se necesitan gran cantidad de datos empíricos, para definir valores deseables de cada medida. Sin embargo como señala De Champeaux [14], debemos de ser conscientes de que "asociar las calificaciones de buenos y malos a determinados rangos numéricos es la parte dura". La definición de estos valores deseables, o valores "umbrales" puede contribuir a la definición de métricas válidas, que puedan resultar útiles para los diseñadores de bases de datos, para tomar mejores decisiones en sus tareas de modelado, que es la meta más importante que cualquier propuesta de medición debe perseguir si pretende ser útil [15].

Además, debe ponerse especial énfasis en la definición formal de las métricas [10], ya que sin definiciones claras y precisas es imposible construir herramientas adecuadas para la extracción de las métricas, se dificulta la replicación de los experimentos y la interpretación de resultados no será correcta.

[2] Las replicaciones internas son aquellas que se llevan a cabo por los experimentadores originales y pueden ser publicadas en artículos de congresos o revistas, mientras que las externas son realizadas por investigadores independientes que quieren corroborar y si es posible mejorar los resultados de otros investigadores [11].

Coincidimos con [9, 32] en que es necesario seguir trabajando en el área de las métricas para modelos conceptuales OO, para obtener métricas que también cubran los modelos dinámicos y funcionales.

7. REFERENCIAS

[1] V. Basili, L. Briand y W. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators". *IEEE Transactions of Software Engineering*, vol. 22, No. 10, pp. 751-761, 1996.

[2] V. Basili, F. Shull y F. Lanubile, "Building Knowledge Through Families of Experiments". *IEEE Transactions on Software Engineering*, vol. 25, No. 4, pp. 435-437, 1999.

[3] E. Bertino y E. Marcos, "Object Oriented Database Systems". En *Advanced Databases: Technology and Design*, O. Díaz y M. Piattini (Eds.), Artech House, 2000.

[4] G. Booch, J. Rumbaugh y I. Jacobson, *"The Unified Modeling Language User Guide"*, Addison-Wesley, 1998.

[5] G. Booch, "UML in Action", *Communications of the ACM*, vol. 2, No. 10, Oct. 1999.

[6] L. Briand, S. Morasca y V. Basili, "Property-Based Software Engineering Measurement", *IEEE Transactions on Software Engineering*, vol. 22, No. 1, pp. 68-86, 1996.

[7] F. Brito e Abreu y R. Carapuça, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process", *4th Int Conference on Software Quality*, McLean, VA, USA, 1994.

[8] F. Brito e Abreu y W. Melo, "Evaluating the Impact of Object-Oriented Design on Software Quality", *Proc. of 3rd International Metric Symposium*, 1996.

[9] F. Brito e Abreu, H. Zuse, H. Sahraoui y W. Melo, "Quantitative Approaches in Object-Oriented Software Engineering". *Object-Oriented Technology: ECOOP'99 Workshop Reader*, Lecture Notes in Computer Science 1743, Springer-Verlag, pp. 326-337, 1999.

[10] F. Brito e Abreu, "Using OCL to Formalize Object Oriented Metrics Definitions". *Technical Report ES007/2001*. FCT/UNL y INESC, 2001.

[11] A. Brooks, J. Daly, J. Miller, M. Roper y M. Wood, "Replication of Experimental Results in Software Engineering", *Technical Report ISERN-96-10*, International Software Engineering Research Network, 1996.

[12] M. Cartwright y Shepperd, "An Empirical Investigation of Object-Oriented Software in Industry", *Technical Report TR 96/01*, Dept. of Computing, Talbot Campus, Bournemouth University, 1996.

[13] S. Chidamber y C. Kemerer, "A Metrics Suite for Object Oriented Design". *IEEE Transactions on Software Engineering*, vol. 20, No. 6, pp. 476-493, 1994.

[14] D. De Champeaux, *"Object-Oriented Development Process and Metrics"*, Upper Saddle River, Prentice-Hall, 1997.

[15] N. Fenton y M. Neil, "Software Metrics: a Roadmap". *Future of Software Engineering*. Ed: Anthony Finkelstein, ACM, pp. 359-370, 2000.

[16] M. Genero, M. Piattini y C. Calero, "Métricas para Jerarquías de Agregación en Diagramas de Clases UML", *Memorias del Jornadas Iberoamericanas de Ingeniería de Requisitos y Ambientes de Software, IDEAS'2000*, Cancún, México, pp. 373-384, 2000.

[17] M. Genero, M. Piattini y C. Calero, "Early Measures For UML Class Diagrams". *L'Objet*, vol. 6, No. 4, Hermes Science Publications, pp. 489-515, 2000.

[18] M. Genero, J. Olivas, M. Piattini y F. Romero, "Using Metrics to Predict OO Information Systems Maintainability". *CAISE 2001*, Interlaken, Suiza, Lecture Notes in Computer Science

(LCNS) LNCS 2068, Dittrich, K., Geppert, A. y Norrie, M.C. (eds.) Springer-Verlag, pp. 388-401, 2001.

[19] M. Genero, L. Jiménez y M. Piattini, "A Prediction Model for OO Information Systems Quality Based on Early Indicators". *5th East-European Conference on Advances in Databases and Information Systems, ADBIS 2001*, Vilnius, Lituania, pp. 211-224, 2001.

[20] M. Genero, L. Jiménez y M. Piattini, "Empirical Validation of Class Diagram Complexity Metrics", *SCCC 2001*, Chile, IEEE Computer Society, pp. 95-104, Nov. 2001.

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

[22] I. Graham, *"Migrating to Object Technology"*. Wokingham, Addison-Wesley, 1995.

[23] R. Harrison, S. Counsell y R. Nithi, "An Evaluation of the MOOD Set of Object-Oriented Software Metrics". *IEEE Transactions on Software Engineering*, vol. 24, No. 6, pp. 491-496, 1999.

[24] B. Kitchenham, S. Pflegger y N. Fenton, "Towards a Framework for Software Measurement Validation". *IEEE Transactions of Software Engineering*, vol. 21, No. 12, pp. 929-943, 1995.

[25] W. Li y S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, vol. 23, No. 2, pp. 111-122, 1993.

[26] M. Lorenz y J. Kidd, *"Object-Oriented Software Metrics: A Practical Guide"*. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[27] E. Marcos, B. Vela, y J.M. Cavero, "Extending UML for Object-Relational Database Design", *Springer Verlag. Lectures Notes in Computer Science, LNCS 2185*, pp. 225-239, 2001.

[28] R. Muller, *"Database Design for Smarties Using UML for Data Modelling"*. San Francisco, Morgan Kaufmann, 1999.

[29] S. Pfleeger, "Assessing Software Measurement". *IEEE Software*. March/April, pp. 25-26, 1997.

[30] G. Poels y G. Dedene, DISTANCE: A Framework for Software Measure Construction, Research, *Technical Report DTEW9937*, Dept. Applied Economics, Katholieke Universiteit Leuven, Belgium, 1999.

[31] G. Poels y G. Dedene, "Distance-Based Software Measurement: Necessary and Sufficient Properties for Software Measures", *Information and Software Technology*, vol. 42, No. 1, pp. 35-46, 2000.

[32] G. Poels y G. Dedene, "Measures for Assessing Dynamic Complexity Aspects of Object-Oriented Conceptual Schemes", *Proceedings of the 19th International Conference on Conceptual Modeling* (ER 2000), Salt Lake City, USA, pp. 499-512, 2000.

[33] E. Weyuker, "Evaluating Software Complexity Measures", *IEEE Transactions Software Eng.*, vol. 14, No. 9, pp. 1357-1365, 1988.

[34] H. Zuse, *"A Framework of Software Measurement"*, Berlin, Walter de Gruyter, 1998.