La Orientación a Objetos hoy

Métricas para diagramas de clases expresados en UML

Marcela Genero, Mario Piattini, Coral Calero
Grupo ALARCOS, Departamento de Informática
Universidad de Castilla-La Mancha

<{mgenero, mpiattin, ccalero}@inf-cr.uclm.es>

Resumen: *la tecnología orientada a objetos ha surgido como una nuevo paradigma para el diseño y desarrollo de software, y su adopción ha ido creciendo en la industria. Para que realmente el desarrollo de software orientado a objetos sea un éxito es necesario poder evaluar su calidad desde las etapas tempranas del ciclo de vida. La idea de este trabajo es proponer métricas para evaluar la complejidad introducida por las jerarquías de generalización en los diagramas de clases obtenidos en las etapas de análisis y diseño, expresados utilizando, el Lenguaje de Modelado Unificado (UML).*

La principal ventaja de las métricas propuestas es que pueden aplicarse en las etapas tempranas del ciclo de vida del sistema que se está desarrollando, pudiendo de esta forma analizar el mismo a medida que se va desarrollando, permitiendo así realizar mejoras antes de llegar a la implementación.

1.Introducción

La tecnología de objetos (**OOT**) ha surgido como una nuevo paradigma para el diseño y desarrollo de software, y su adopción ha ido imponiendo en la industria. La incorporación de aspectos como la herencia, la encapsulación y el polimorfismo, hace que el software desarrollado con OOT sea muy diferente al desarrollado utilizando los enfoques tradicionales y procedurales.

Dentro del desarrollo del software **OO** se invierte la mayor parte del tiempo y esfuerzo en las etapas tempranas del ciclo de vida, el análisis y el diseño. Esto hace que los diseñadores de software OO deban estructurar «más» cuidadosamente el diseño, para luego obtener un producto implementado de mayor calidad.

Para hacer que el desarrollo de software OO sea un proceso de ingeniería, es necesario poder medir el proceso y productos en diferentes etapas del ciclo de vida.

La medición del software es ampliamente reconocida como un medio efectivo para entender y mejorar el desarrollo de software y proyectos de mantenimiento (Briand et al., 1996), para mantener la calidad de los sistemas (Graham, 1995), para focalizar las áreas problemáticas (De Champeaux, 1997), y para determinar la mejor manera de ayudar a los profesionales e investigadores en su trabajo (Pflegeer, 1997).

Si bien en la literatura existente podemos encontrar métricas para software OO (Brito e Abreu y Carapuça, 1994; Chidamber y Kemerer, 1994; Fenton y Pfleeger, 1997; Henderson-Sellers, 1996), 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).

Esto proporciona la información demasiado tarde para ayudar a mejorar las características internas del producto antes de su finalización. Por eso hay una gran necesidad de contar con métricas y modelos que puedan aplicarse en las etapas tempranas del desarrollo, particularmente durante la especificación de requisitos y el diseño, para asegurar que el análisis y diseño tienen propiedades internas favorables que pueden conducir a desarrollar productos de software de calidad.

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.

Como UML (Booch, 1998) es un estándar de modelado que no lleva demasiado tiempo en el mercado del software OO, no existen muchas propuestas de métricas que permitan evaluar su calidad. Recientemente han surgido algunos trabajos sobre métricas aplicadas a diagramas de clases en OMT (Genero et al., 1999a) y en **UML** (Marchesi, 1998) aunque éstos no han tratado en profundidad el tema de las jerarquías de generalización.

A continuación se proponen dos métricas para evaluar la complejidad introducida por las jerarquías de generalización en los diagramas de clases expresados utilizando UML considerando que tal complejidad puede afectar a la mantenibilidad del producto de software que se esté desarrollando. Para quien esté interesado, la validación formal de dichas métricas siguiendo el marco propuesto por Briand et al. (1996) puede encontrarse en Genero et al. (1999b).

2. Métricas para diagramas de clases UML

A continuación se presentan dos métricas que permiten

medir la complejidad introducida por las jerarquías de generalización.

2.1. Métrica M_{JG}

El objetivo de la métrica Jerarquía de Generalización (M_{JG}) es evaluar la complejidad introducida por las jerarquías de generalización en un diagrama de clases, teniendo en cuenta algunos factores que influyen en el «tamaño de una jerarquía» (cantidad de subclases, cantidad niveles en la jerarquía y sobretodo la herencia múltiple). M_{JG} se define de la siguiente manera:

- Si el diagrama de clases *no tiene* jerarquías de generalización:

$$M_{JG} = 0$$

- Si el diagrama de clases *tiene* jerarquías de generalización M_{JG} se define de la siguiente manera:

$$M_{JG} = \sum_{i=1}^{n} CJ_i ,$$

siendo CJ_i la complejidad de la i-ésima jerarquía de generalización y n el número total de jerarquías de generalización en un diagrama de clases.

Para calcular CJ_i se combinan dos factores. El primer factor es la fracción de entidades en la jerarquía de generalización, que son hojas. Esta medida, llamada $FHOJA_i$ se calcula así:

$$FHOJA_i = \frac{N_i^{HOJA}}{N_i^C} ,$$

siendo

$$N_i^{HOJA}$$

el número de clases que son hojas en la i-ésima jerarquía de generalización, y

$$N_i^C$$

el número total de clases en la i-ésima jerarquía de generalización.

La **figura 1** muestra diferentes jerarquías de generalización con sus respectivos valores de $FHOJA_i$. $FHOJA_i$ tiende al valor 0,5 cuando el número de hojas es la mitad del número de entidades (**figura 1**, ej. **c** y **d**). Tiende a cero en el caso de un árbol unario (**figura 1**, ej. **b**). Y tiende a 1 cuando cada clase es una subclase de la superclase de la jerarquía (**figura 1**, ej. **a**). El factor $FHOJA_i$ tiene una propiedad poco deseable: para jerarquías poco profundas (dos o tres niveles) con una gran cantidad de hojas nos da un valor demasiado alto (**figura 1**, ej. **a**). Para corregir este problema, utilizamos un factor adicional para calcular el valor de la métrica CJ_i, el promedio de las subclases directas o indirectas para cada subclase que no sea la superclase de la jerarquía. A este factor lo denominamos $ALLSUP_i$ (la superclase de la jerarquía no la tenemos en cuenta ya que no tiene padres). Por lo tanto podemos definir CJ_i de la siguiente manera:

$$CJ_i = FHOJA_i - \frac{FHOJA_i}{ALLSUP_i}$$

2.2. Métrica M_{HMJG}

La métrica Herencia Múltiple para la Jerarquías de Generalización (M_{HMJG}) evalúa la complejidad introducida por la herencia múltiple en las jerarquías de generalización en un diagrama de clases.

La experiencia ha demostrado que el uso de herencia múltiple puede llevar a complicaciones imprevistas, y por eso es conveniente evitarla siempre que sea posible. M_{HMJG} se define de la siguiente manera:

- Si el diagrama de clases *no tiene* herencia múltiple

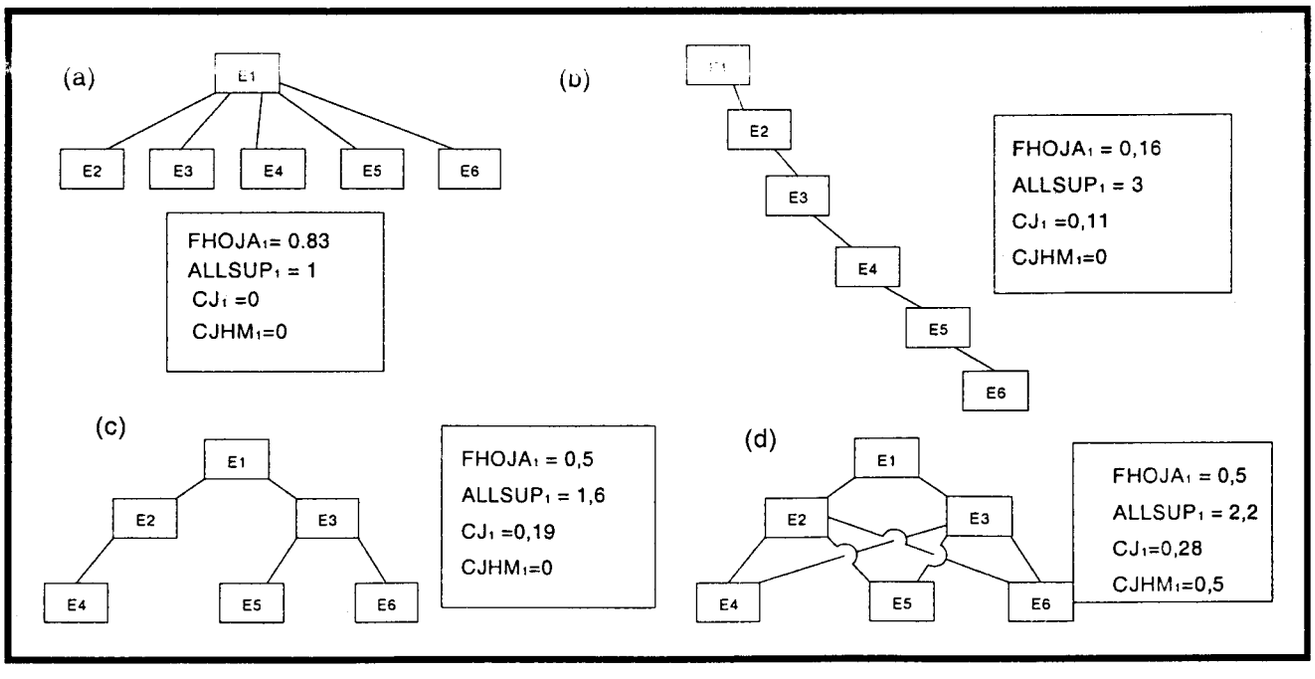

Figura 1. Ejemplos de jerarquías de generalización

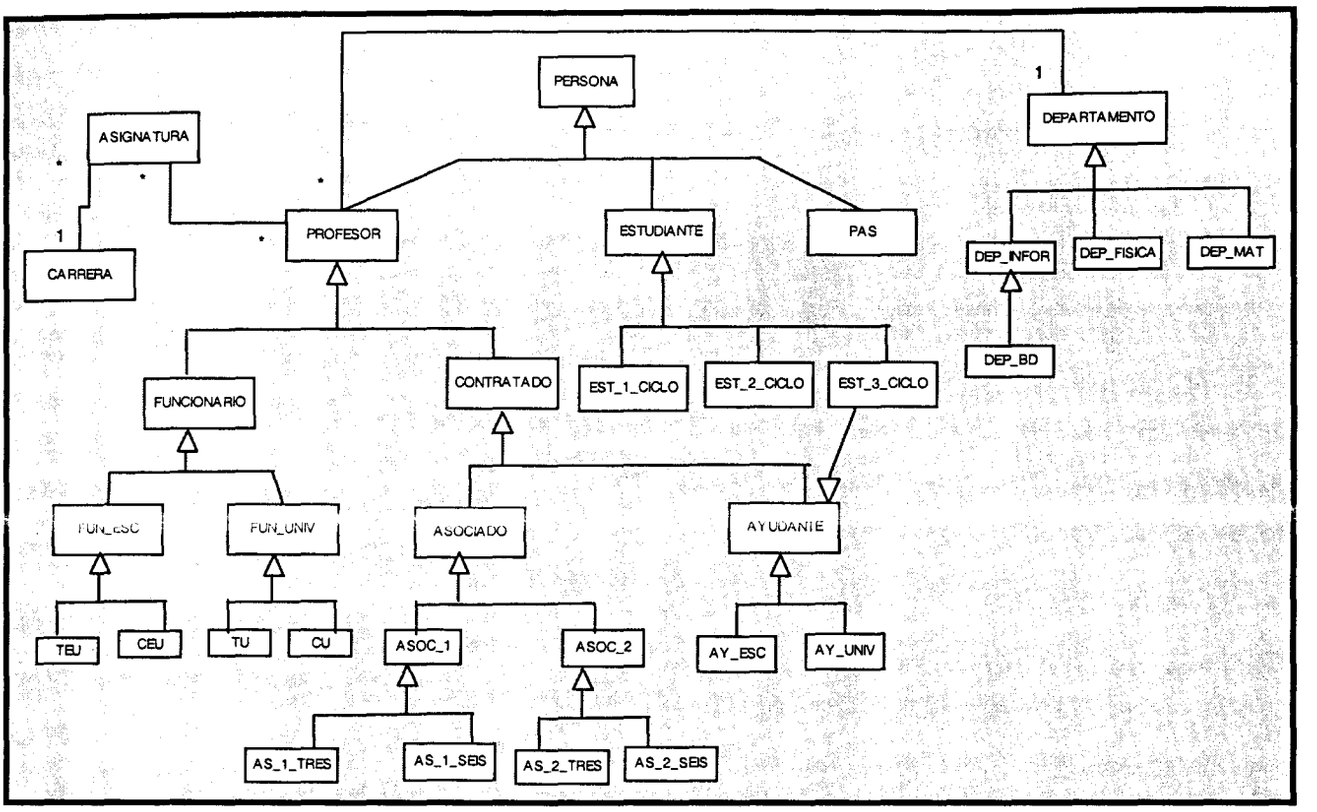

Figura 2. Ejemplo de un diagrama de clases utilizandoUML

$$M_{HMJG} = 0$$

Si el diagrama de clases *tiene* jerarquías de generalización MHMJG se define de la siguiente manera:

$$M_{HMJG} = \sum_{i=1}^{n} CJHM_i ,$$

siendo $CJHM_i$ la complejidad de herencia múltiple de la i-ésima jerarquía de generalización, y n es el número total de jerarquías de generalización en un diagrama de clases. $CJHM_i$ mide el ratio de padres extras (más de una superclase) que tiene cada clase y se calcula de la siguiente manera:

$$CJHM_i = \frac{N_i^{EX}}{N_i^{C}} \text{ , siendo } N_i^{EX}$$

el número de superclases (padres) extras que tiene la clase i-ésima, y

$$N_i^{C}$$

es el número total de clases en i-ésima jerarquía de generalización.

2. 3. Aplicación de las métricas a un ejemplo

A continuación se muestra la aplicación de las métricas propuestas al ejemplo mostrado en la **figura 2**. Para simplificar el diagrama y destacar el uso de jerarquías de generalización y la herencia múltiple, sólo se especifica en cada clase su nombre.

Cálculo de la métrica M_{JG}

El diagrama de clases de la **figura 2**, tiene dos jerarquías de generalización, una cuya clase raíz es la clase PERSONA (J1) y otra cuya clase raíz es la clase DEPARTAMENTO (J2).

Cantidad de clases J1 = 25, Cantidad de hojas J1= 13,

$$FHOJA_1 = \frac{13}{25} = 0{,}52$$

Cantidad de clases J2 = 5, Cantidad de hojas J2= 3,

$$FHOJA_2 = \frac{3}{5} = 0{,}6$$

Para calcular ALLSUP se debe sumar la cantidad de padres directos e indirectos que tiene cada clase de la jerarquía y luego dividirlo por la cantidad de clases en la jerarquía sin tener en cuenta la raíz, ya que no tiene padre.

$$ALLSUP_1 = \frac{79}{24} = 3{,}29$$

y entonces

$$CJ_1 = 0{,}52 - \frac{0{,}52}{3{,}29} = 0{,}52 - 0{,}16 = 0{,}36$$

$$ALLSUP_2 = \frac{5}{4} = 1{,}25$$

y entonces

$$CJ_2 = 0{,}6 - \frac{0.6}{1{,}25} = 0{,}6 - 0{,}48 = 0{,}12$$

Por lo tanto el valor de

$$M_{JG} = \sum_{i=1}^{n} CJ_i = 0{,}36 + 0{,}12 = 0{,}48$$

Si bien al mirar el diagrama de clases de la **figura 2**, uno

puede pensar que la jerarquía es un tanto compleja, el valor obtenido en la métrica no indica lo mismo, ya que esta métrica está influenciada sobretodo por la herencia múltiple, y en este diagrama existe una única clase EST_3_CICLO que hereda de las clases AYUDANTE Y ESTUDIANTE.

Cálculo de la métrica M_{HMJG}

Para calcular esta métrica hay que tener en cuenta el número de padres extra que tiene una clase, es decir la presencia de múltiple herencia. Como se ve en el diagrama de la figura 2 existe múltiple herencia en una única clase, la clase EST_3_CICLO que hereda de las clases AYUDANTE Y ESTUDIANTE. Por lo tanto

$$CJHM_1 = \frac{1}{25} = 0{,}04 \text{ y } CJHM_2 = 0$$

y

$$M_{MHJG} = \sum_{i=1}^{n} CJHM_i = 0{,}04 + 0 = 0{,}04$$

3. Conclusiones

En este trabajo se han propuesto dos métricas M_{JG} y M_{HMJG} que miden la complejidad introducida por las jerarquías de generalización y particularmente por la herencia múltiple en los diagramas de clase expresados en UML.

La principal ventaja de estas métricas con respecto a la mayoría de las existentes es que pueden aplicarse al diseño de un producto de software OO, es decir en las etapas tempranas del ciclo de vida, proporcionándole a los diseñadores información útil para mejorar las características internas del producto antes de su implementación.

Nuestro objetivo inmediato es seguir investigando en el tema de las métricas OO para proponer otras que consideren otros aspectos que influyen en la calidad del software OO, como la cohesión, el acoplamiento, reutilización, etc.

Si bien hemos realizado la validación formal de las métricas propuestas (Genero et al., 1999b), es necesario también validarlas empíricamente para poder corroborar que la «complejidad» de las jerarquías de generalización afecta efectivamente a algunos de los factores que hacen a la mantenibilidad. Como UML ha surgido hace relativamente poco tiempo, será posible realizar la validación empírica cuando haya disponible más datos sobre proyectos desarrollados utilizando este estándar.

Igualmente es objetivo de nuestro grupo de investigación el desarrollo de una herramienta que permita calcular y analizar los valores de las métricas de forma automática.

4. Referencias

Booch, G., Rumbaugh J. y Jacobson, I. (1998). *The Unified Modeling Language User Guide*. Addison-Wesley.

Briand, L., Morasca, S. and Basili, V. (1996). Property-Based Software Engineering Measurement. *IEEE Transactions on Software Engineering*, 22 (6), 68-86.

Brito e Abreu, F. y Carapuça, R. (1994). Candidate Metrics for Object Oriented Software within a Taxonomy Framework. *Journal of Systems and Software*, 26(1), North-Holland, Elsevier Science.

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

De Champeaux, D. (1997). Object-oriented development process and metrics. Upper Saddle River, Prentice-Hall.

Fenton, N. (1994). Software Measurement: A Necessary Scientific Basis. *IEEE Transactions on Software Engineering*, 20(3). 199-206.

Fenton E. y Pfleeger, S. (1997). Software Metrics. A Rigurous and Practical Approach. *International Thompson Computer Press.*

Genero, M., Manso, Mª E., Piattini, M. y García, F. (1999a). Assessing the Quality and the Complexity of OMT Models. *2nd European Software Measurement Conference - FESMA 99*, Amsterdam, The Netherlands. 99-109.

Genero, M., Piattini, M. y Calero, C. (1999b). Métricas para diagramas de clases UML. Actas V Congreso OO, Sevilla, octubre de 1999.

Graham, I. (1995). *Migrating to object technology. Wokingham*, Addison-Wesley.

Henderson-Sellers, B. (1996). *Object-Oriented Metrics - Measures of complexity*. Prentice-Hall, Upper Saddle River, New Jersey.

Marchesi, M. (1998). OOA metrics for the Unified Modeling Language. *Proceedings of the Second Euromicro Conference on Software Maintenance and Reenginerring*. Florencia. Italia, 67-73.

Pfleeger, S. (1997). Assessing Software Measurement. *IEEE Software*. March/April, 25-26.