CAPÍTULO 6

Métricas para Sistemas Orientados a Objetos

El Software Orientado a Objetos (OO) es fundamentalmente distinto del software que se desarrolla utilizando métodos convencionales. Las métricas para sistemas OO deben de ajustarse a las características que distinguen el software OO del software convencional. Estas métricas hacen hincapié en el encapsulamiento, la herencia, complejidad de clases y polimorfismo. Por lo tanto las métricas OO se centran en métricas que se pueden aplicar a las características de encapsulamiento, ocultamiento de información, herencia y técnicas de abstracción de objetos que hagan única a esa clase. Como en todas las métricas los objetivos principales de las métricas OO se derivan del software convencional: comprender mejor la calidad del producto, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado a nivel del proyecto.

Se conoce que las medidas y las métricas son componentes clave de cualquier disciplina de la ingeniería; la ingeniería de software orientada a objetos no es una excepción. Lamentablemente, la utilización de métricas para sistemas orientados a objetos ha progresado con mucha más lentitud que la utilización de los demás métodos OO [Luis A. Laranjeira '90]. Sin embargo, a medida que los sistemas OO van siendo más habituales, resulta fundamental que los ingenieros del software dispongan de mecanismos cuantitativos para estimar la calidad de los diseños y la efectividad de los programas 00.

6.1 Objetivo de las métricas Orientados a Objetos

Los objetivos principales de las métricas orientadas a objetos son los mismos que los existentes para las métricas surgidas para el software estrucutrado:

- Comprender mejor la calidad del producto
- Estimar la efectividad del proceso
- .Mejorar la calidad del trabajo realizado en el nivel del proyecto.

Cada uno de estos objetivos es importante en sí, pero para el ingeniero de software, la calidad del producto debe de ser lo esencial. ¿Cómo se puede medir la calidad de un sistema 0.0? ¿ Que características del modelo de diseño se pueden estimar para decretar si el sistema será o no fácil de implementar, se podrá probar, que será fácil de modificar, y lo que es más importante, resultará tolerable para los usuarios finales? [Laranjeira 1990]. Estos argumentos se tratarán de resolver a lo largo de este capítulo

6.2 Características del software Orientado a Objetos

El software orientado a objetos es esencialmente distinto del software que se desarrolla utilizando métodos convencionales. Por esta razón, las métricas para sistemas 00 deben de concordarse a las características que distinguen el software 00 del software convencional.

Berard [Laranjeira ‘90] define cinco características que dan lugar a unas métricas especializadas:

- Localización,
- Encapsulamiento,
- Ocultamiento de información,
- Herencia y
- Técnicas de abstracción de objetos.

6.2.1 Localización

La localización es una característica del software que indica la forma que se concentra la información dentro de un programa. En el contexto OO, la información se concentra mediante el encapsulamiento tanto de datos como de procesos dentro de los límites de una clase u objeto.

Dado que el software convencional hace hincapié en las funciones como mecanismos de localización, las métricas de software se han centrado en la estructura interna o complejidad de las funciones (p. ej.: longitud del módulo, cohesión, o complejidad ciclomática) o bien en la forma en que las funciones se conectan entre sí (p. ej.: acoplamiento de módulos).

Dado que las clases constituyen la unidad básica de los sistemas 00, la localización está basada en los objetos. Por tanto, las métricas deberían de ser aplicables a la clase (objeto) como si se tratara de una entidad completa. Además, la relación entre operaciones (funciones) y clases no es precisamente uno-a-uno.

Por tanto, las métricas que reflejan la forma en que colaboran las clases deben de ser capaces de adaptarse a las relaciones uno-a-muchos y muchos-a-uno.

6.2.2 Encapsulamiento

Berard [Pressman '98] define el encapsulamiento como "el empaquetamiento (o enlazado) de una colección de elementos. Entre los ejemplos de encapsulamiento de bajo nivel (software convencional) se cuentan los registros y matrices, y los subprogramas (por ejemplo, procedimientos, funciones, subrutinas y párrafos) son mecanismos de nivel medio para el encapsulamiento".

Para los sistemas 00, el encapsulamiento comprende las responsabilidades de una clase, incluyendo sus atributos (y otras clases para objetos agregados) y operaciones, y los estados de la clase, según se definen mediante valores específicos de atributos.

El encapsulamiento influye en las métricas cambiando el objetivo de la medida, que pasa de ser un único módulo a ser un paquete de datos (atributos) y de módulos de procesamiento (operaciones). Además, el encapsulamiento impulsa a la medida hasta un nivel de abstracción más elevado.

6.2.3 Ocultamiento de información

El ocultamiento de información suprime los detalles operativos de un componente de un programa. Tan sólo se proporciona la información necesaria

para acceder a ese componente o a aquellos otros componentes que deseen acceder a él.

Un sistema 00 bien diseñado debería de impulsar al ocultamiento de información. Por tanto, aquellas métricas que proporcionen una indicación del grado en que se ha logrado el ocultamiento proporcionarán una indicación de la calidad del diseño 00.

6.2.4 Herencia

La herencia es un mecanismo que hace posible que los compromisos de un objeto se difundan a otros objetos. La herencia se produce a lo largo de todos los niveles de la jerarquía de clases, bien es sabido que en general, el software convencional, no admite esta característica.

Dado que la herencia es una característica fundamental de muchos sistemas 00, hay muchas métricas 00 que se centran en ella. Esto se conocerá entre los ejemplos que se tratarán más adelante: se cuentan el número de descendientes (número de instancias inmediatas de una clase), número de predecesores (número de generalizaciones inmediatas), y grado de anidamiento de la jerarquía de clases (profundidad de una clase dentro de una jerarquía de herencia) y otros.

6.2.5 Abstracción

La abstracción es un mecanismo que permite al diseñador centrarse en los detalles esenciales de algún componente de un programa (tanto si es un dato

como si es un proceso) sin preocuparse por los detalles de nivel inferior. Cuando los niveles de abstracción van elevándose, se ignoran más y más detalles, por lo tanto, se proporciona una visión más general de un concepto u objeto. A medida que pasamos a niveles mas reducidos de abstracción, se muestran más detalles, esto es, se proporciona una visión más específica de un concepto u objeto.

Dado que una clase es una abstracción que se puede visualizar con muchos niveles distintos de detalles, y de muchas maneras diferentes, las métricas OO representan la abstracción en términos de medidas de una clase (p.ej.: número de instancias por clase por aplicación).

6.3 Métricas para el modelo de diseño Orientado a Objetos

Gran parte del diseño orientado a objetos es subjetivo un diseñador experimentado "sabe" como puede caracterizar un sistema 00 para que se implemente de forma efectiva los requisitos del cliente. Pero a medida que los modelos de diseño 00 van creciendo de tamaño y complejidad, puede resultar beneficiosa una visión más objetiva de las características del diseño, tanto para el diseñador experimentado como para el menos experimentado.

Una visión objetiva del diseño debería de tener un componente cuantitativo y esto nos lleva a las métricas 00, en donde se pueden aplicar no solo al modelo de diseño, sino también al modelo de análisis [Laranjeira.'90].

6.4 Métricas orientadas a Clases

Se sabe que la clase es la unidad principal de todo sistema 00. Por consiguiente, las medidas y métricas para una clase individual, la jerarquía de clases, y las colaboraciones de clases resultarán sumamente valiosas para un ingeniero de software que tenga que estimar la calidad de un diseño. Se ha visto que la clase encapsula a las operaciones (procesamiento) y a los atributos (datos). La clase suele ser el "predecesor" de las subclases (que a veces se denominan "descendientes") que heredan sus atributos de operaciones. La clase suele colaborar con otras clases. Todas estas características se pueden utilizar como bases de las métricas explicadas , enseguida:

6.4.1 El conjunto de métricas CK

Uno de los conjuntos de métricas de software 00 a los que se hace más ampliamente referencia es el propuesto por Chidamber y Kemener [Laranjeira '90]. Estas métricas propuestas de diseño basadas en clases, a las cuales suele aludirse con el nombre de conjunto de métricas CK para sistemas 00.

Los métodos ponderados por clase (MPC), son aquellos en donde se definen n métodos de complejidad C_1, $C_{2...,}$ C_n, para una clase C. La métrica de complejidad específica que se selecciona debe de normalizarse de tal modo que la complejidad nominal para un método tome el valor 1.0.

$$MPC = Ó\ C_i \quad \text{para } i = 1 \text{ hasta } n \qquad (6.1)$$

El número de métodos y su complejidad es un indicador razonable de la cantidad de esfuerzo necesaria para implementar y comprobar una clase. Además, cuanto mayor sea el número de métodos, más complejo será el árbol de herencia, (todas las subclases heredan el método de sus predecesores). Finalmente, a medida que el número de métodos crece para una clase dada; es más probable que se vuelva cada vez más específico de la aplicación, imitando por tanto su potencial de reutilización. Por todas estas razones, MPC debería de mantener un valor tan bajo como sea razonable.

Aun cuando podría parecer relativamente sencillo desarrollar un contador del número de métodos de una clase, el problema es en realidad más complejo de lo que parece. Churcher y Shepperd [Laranjeira '90] examinan este problema precisamente cuando escriben:

> Para contar métodos, es preciso responder a una pregunta fundamental: "¿Pertenece un método solamente a la clase que lo define, 0 bien pertenece también a todas aquellas clases que lo heredan de forma directa o indirecta?". Las preguntas como estas pueden parecer superficiales, por cuanto el sistema de ejecución acabará últimamente por resolverlas. Sin embargo, las implicaciones para las métricas pueden ser significativas.

Una posibilidad es restringir el recuento a la clase en curso, ignorando los miembros heredados. La motivación para esto sería que los miembros heredados ya habrán sido contados en la clases en que fueran definidos, así que el incremento de clase es la mejor medida de su funcionalidad. Con objeto de

entenderlo que hace una clase, la fuente de información más importante son sus propias operaciones.

Por otro lado, el recuento podría envolver a todos aquellos métodos definidos en la clase en curso, junto con todos los métodos heredados. Este enfoque hace hincapié en la importancia del espacio de estados, en lugar de hacer hincapié en el incremento de la clase, para comprender la clase.

Al igual que la mayoría de las convenciones de recuento en las métricas de software, cualquiera de los enfoques mencionados anteriormente es aceptable, siempre y cuando este enfoque de recuento se aplique de forma consistente siempre que se recoja una métrica.

Árbol de Profundidad de Herencia (APH). Esta métrica se define como la longitud máxima desde el nodo hasta la raíz del árbol en la Figura 6.1 [Pressman‘98], el valor de APH para la jerarquía de clases mostrada el valor resultante es 4.

A medida que crece el APH, es más probable que las clases de niveles inferiores hereden muchos métodos. Esto da lugar a posibles dificultades cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (con un valor grande de APH) lleva también a una mayor complejidad de diseño. Por el lado positivo, los valores grandes de APH implican que se pueden reutilizar muchos métodos.

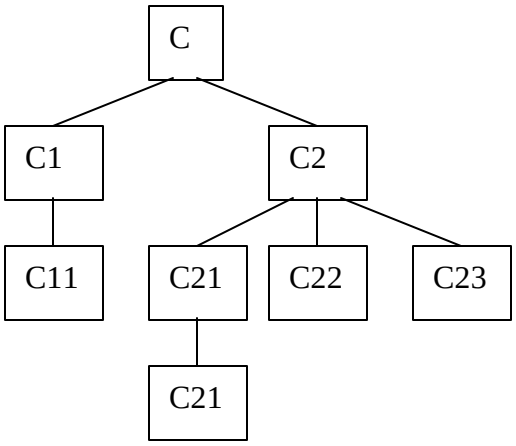

Figura 6.1 Una Jerarquía de Clases [Pressma '98]

Número de Descendientes (NDD). Las subclases que son inmediatamente subordinadas a una clase son denominan descendientes. En la Figura 6.1 la clase C2 tiene tres descendientes las subclases C21, C22 y C23.

A medida que crece el número de descendientes, se incrementa la reutilización, pero también es cierto, que a medida que crece NDD, la abstracción representada por la clase predecesora puede verse diluida. Esto es, existe la posibilidad de que algunos de los descendientes no sean realmente miembros propios de la clase predecesora. A medida que NDD va creciendo, la cantidad de pruebas crecerá también.

Respuesta Para una Clase (RPC). El conjunto de respuestas de una clase es un conjunto de métodos que pueden ser ejecutados potencialmente en respuesta a un mensaje recibido por un objeto de esa clase [Pressman'98]. RPC se define como el número de métodos existentes en el conjunto de respuestas.

A medida que crece RPC, el esfuerzo necesario para la comprobación crece, porque la sucesión de comprobación va creciendo y también la complejidad global de diseño de la clase crece.

Carencia de Cohesión en los Métodos (CCM). Todo método situado dentro de una clase C, accesa a uno o más , y en donde CCM es el número de métodos que acceden a uno o más de los mismos atributos. Si ningún método accede a los mismos atributos, entonces CCM será 0.

Para enseñar el caso en que CCM es distinto de 0, supóngase una clase con 6 métodos, en donde cuatro de los métodos tienen en común uno o más atributos, por consiguiente, CCM = 4.

Si CCM es elevado, los métodos pueden estar acoplados entre sí a través de atributos. Esto incrementará la complejidad del diseño de clases. En general, unos valores elevados para CCM implican que la clase podría diseñarse mejor descomponiéndola en dos o más clases distintas. Aun cuando existen casos en que es justificable un valor elevado de CCM, en donde es deseable mantener elevado un grado de cohesión, esto es mantener un valor bajo para CCM.

6.4.2 Métricas propuestas por Lorenz y Kidd

En el libro de métricas realizado por Lorenz y Kidd 0.0 [Laranjeira '90], dividen las métricas basadas en clases en cuatro categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas a tamaños para una clase 00 se centran en cálculos de atributos y de operaciones para una clase individual, y promedian los valores para el sistema 00 en su totalidad. Las métricas basadas en herencia se centran en la forma en que se reutilizan las operaciones a

lo largo y ancho de la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión y asuntos relacionados con el código, y las métricas orientadas a valores externos examinan el acoplamiento y la reutilización.

Tamaño de Clase (TC). El tamaño general de una clase se puede determinar empleando las medidas siguientes:

- El número total de operaciones (tanto operaciones heredadas como privadas de la instancia) que están encapsuladas dentro de la clase.
- El número de atributos (tanto atributos heredados como atributos privados de la Instancia) que están encapsulados en la clase.

Si existen valores grandes de TC éstos mostrarán que una clase puede tener demasiada responsabilidad, lo cual reducirá la reutilizabilidad de la clase y complicará la implementación y la comprobación, por otra parte cuanto menor sea el valor medio para el tamaño, más probable es que las clases existentes dentro del sistema se puedan reutilizar ampliamente.

Número de Operaciones Invalidadas por una subclases (NOI). Existen casos en que una subclase sustituye una operación heredada de su superclase por una versión especializada para su propio uso, ya esto se le denomina invalidación. Los grandes valores de NOI suelen indicar un problema de diseño ya que si NOI es elevado, entonces el diseñador ha violado la abstracción implicada por la superclase. Esto da lugar a una jerarquía de clases débil, y a un software 00 que pueda resultar difícil de comprobar y modificar.

Índice de Especialización (IE). El índice de especialización proporciona una indicación aproximada del grado de especialización de cada una de las subclases existentes en un sistema orientado a objetos.

La especialización se puede alcanzar añadiendo o borrando operaciones, o bien por invalidación.

$$IE = [NOI \times nivel] M_{total} \quad (6.2)$$

en donde *nivel* es el nivel de la jerarquía de clases en que reside la clase, y M_{total} es el número total de métodos para la clase. Cuanto más elevado sea el valor de IE es más probable que la jerarquía de clases tenga clases que no se ajustan a la abstracción de la superclase.

6.5 Métricas Orientadas a Operaciones

Dado que la clase es la unidad dominante en los sistemas 00, se han planteado menos métricas para las operaciones de clases. Churcher y Shepperd [Presmman'98] describen esto cuando afirman:

> Los resultados de los últimos estudios indican que los métodos tienden a ser pequeños, tanto en términos del número de sentencias como en términos de su complejidad lógica [WIL92], lo cual sugiere que la estructura de conectividad de un sistema pueda resultar más importante que el contenido de los módulos individuales.

Sin embargo, existen algunas ideas que pueden llegar a estimarse, se indican a continuación tres métricas sencillas propuestas por Lorenz y Kidd [Pressman '98]:

- **Tamaño medio de operación (TOavg).** Aunque se lograría utilizar las líneas de código como indicador para el tamaño de operación, la medida LOC padece de considerables problemas. Por esta razón, el número de mensajes enviados por la operación proporciona una alternativa para el tamaño de la operación. A medida que asciende el número de mensajes enviados por una única operación, es posible que las responsabilidades no hayan sido bien estipuladas dentro de la clase.
- **Complejidad de operación (CO).** La complejidad de una operación se consigue calcular empleando cualquiera de las métricas de complejidad propuestas para el software convencional Sabiendo que las operaciones convendrían limitarlas a una responsabilidad específica, en donde el diseñador debería esforzarse por mantener el valor de CO tan bajo como sea posible.

Número Medio de Parámetros por operación (NPavg). En cuanto sea más grande el número de parámetros de la operación, será más compleja la colaboración entre objetos. En general, NPavg debería de mantenerse tan bajo como sea posible.

6.6 Métricas para Pruebas Orientadas a Objetos

Las métricas de diseño proporcionarán una predicción de la calidad del diseño, también proporcionan una indicación general de la cantidad de esfuerzo de pruebas necesario para aplicarlo en un sistema 00.

Binder [Pressman ‘98] sugiere una amplia gamma de métricas de diseño que tienen influencia directa en la “comprobabilidad” de un sistema 00. Las métricas se crean en categorías que reflejan características de diseño importantes:

- **Encapsulamiento:**

  Carencia de Cohesión en Métodos (CCM). Cuanto más alto sea el valor de CCM, más estados tendrán que ser probados para asegurar que los métodos no den lugar a efectos secundarios.

  Porcentaje Público y Protegido (PPP). Los atributos públicos se heredan de otras clases, y por tanto son visibles para esas clases. Los atributos protegidos son una especialización y son privados de alguna subclase específica. Esta métrica indica el porcentaje de atributos de clase que son públicos. Unos valores altos de PPP incrementan la probabilidad de efectos colaterales entre clases, y por lo tanto es preciso diseñar comprobaciones que aseguren que se descubran estos efectos colaterales.

  Acceso Público a Datos miembros (APD). Esta métrica muestra el número de clases (o métodos) que pueden acceder a los atributos de otra clase, violando así el encapsulamiento. Unos valores altos de APD dan lugar a un potencial de efectos colaterales entre clases. Es preciso diseñar comprobaciones para asegurar que se descubran estos efectos colaterales.

- **Herencia**

  Número de Clases Raíz (NCR). Esta métrica es un recuento de las jerarquías de clases distintas que se describen en el modelo de diseño., en donde será preciso desarrollar conjuntos de pruebas para cada una de las clases raíz, y para la correspondiente jerarquía de clases. A medida que crece NCR crece también el esfuerzo de comprobación.

  ADMisión (ADM). Cuando se utiliza en el contexto 00, la admisión es una indicación de herencia múltiple. Cuando la ADM sea mayor a 1, indica que una clase hereda sus atributos y operaciones de más de una clase raíz. Siempre que sea posible, es preciso evitar un valor de ADM mayor a 1.

6.7 Métricas para proyectos Orientados a Objetos

Se sabe que el trabajo del administrador de un proyecto es planificar, coordinar, seguir y controlar un proyecto de software. ¿Pero qué sucede con las medidas? ¿Existen métricas 00 especializadas que pueda utilizar el administrador del proyecto para disponer de una mejor visión de su progreso? La respuesta es por supuesto que “sí”.

La primera actividad que desarrolla el administrador del proyecto es la planificación y una de las primeras tareas de la planificación es la estimación. Por tanto, el plan y sus estimaciones de proyecto se visitan después de cada iteración

de Análisis Orientado a Objetos(A00), Diseño Orientados a Objetos (D00), e incluso la Programación Orientada a Objetos (POO).

Uno de los problemas fundamentales a los que se enfrenta un administrador de proyectos durante la planificación es, la estimación del tamaño implementado del software, ya que se conoce que el tamaño es directamente proporcional al esfuerzo y la duración. Las métricas 00 siguientes [Pressman'98] pueden aportar ideas acerca del tamaño del software:

Número de Clases Clave (NCC). Las clases clave se centran directamente en el dominio del negocio para el problema que se estará analizando, y tendrán menor probabilidad de ser implementadas mediante reutilización. Por esta razón, unos valores elevados NCC indican que en nuestro camino encontraremos una cantidad notable de trabajo de desarrollo. Lorenz y Kidd [Pressman'98] sugieren que entre el 20 y el 40 por ciento de todas las clases de un sistema 00 típico son clases clave. El resto sirve como infraestructura de apoyo (IGU, comunicaciones, bases de datos, etc.).

Número de SUBsistemas (NSUB). El número de subsistemas proporciona una idea general de la asignación de recursos, de la planificación (con especial hincapié en el desarrollo en paralelo), y del esfuerzo global de integración.

Las métricas NCC y NSUB se pueden escoger para otros proyectos OO anteriores, y se pueden relacionar con el esfuerzo invertido en el proyecto y también con las actividades de proceso individuales. Estos datos se pueden utilizar también junto con métricas de diseño, con el objeto de calcular "métricas de productividad" tales como el número medio de clases por desarrollador o el

promedio de métodos por persona y mes. En su conjunto, estas métricas se pueden utilizar para estimar el esfuerzo, la duración, el personal y otras informaciones acerca del proyecto para el proyecto actual.