CAPÍTULO

24 MÉTRICAS TÉCNICAS PARA SISTEMAS ORIENTADOS A OBJETOS

EN secciones anteriores de este libro, se mencionó que las métricas y mediciones son componentes clave para cualquier disciplina de ingeniería —y la ingeniería de software orientada a objetos no es una excepción—. Desgraciadamente, el uso de métricas para sistemas OO ha progresado mucho más despacio que'el uso de otros métodos OO. Ed Berard [BER95] mencionó la ironía de las mediciones, cuando declaraba que:

> La gente involucrada en el software parece tener una relación amor-odio con las métricas. Por una parte, menosprecian y desconfían de cualquier cosa que parezca o suene a medición. Son rápidos, bastante rápidos para señalar las «imperfecciones», en los argumentos de cualquiera que hable acerca de las mediciones a los productos de software, procesos de software, y (especialmente) personas involucradas en software. Por otra parte, esta misma gente parece no tener problemas al identificar qué lenguaje de programación es el mejor, las estupideces que los administradores hacen para «arruinar» proyectos, y la metodología de trabajo en qué situaciones.

La «relación amor-odio» que Berard declara es real. Más aun, como los sistemas OO se encuentran cada vez más implantados, es esencial que los ingenieros en software posean mediciones cuantitativas, para la evaluación de calidad de diseños, a niveles arquitectónicos y de componentes.

Estas mediciones permiten al ingeniero evaluar el software al inicio del proceso, haciendo cambios que reducirán la complejidad y mejorarán la viabilidad, a largo plazo, del producto final.

VISTAZO RÁPIDO

¿Qué es? Construir software OO ha sido una actividad de ingeniería, que confía más en el juicio, folklore y referencias cualitativas, que en la evaluación cuantitativa. Las métricas OO se han introducido para ayudar a un ingeniero del software a usar el análisis cuantitativo, para evaluar la calidad en el diseño antes de que un sistema se construya. El enfoque de métricas OO está en la clase —la piedra fundamental de la arquitectura OO—.

¿Quién lo hace? Los ingenieros del software se ayudan de las métricas para construir software de mayor calidad.

¿Por qué es importante? Como se expuso en el vistazo rápido del Capítulo 19, la evaluación cualitativa de software debe complementarse con el análisis cuantitativo. Un ingeniero del software necesita un criterio objetivo para ayudarse a conducir el diseño de la arquitectura OO, las clases y subsistemas que conforman la arquitectura, las operaciones y atributos que constituyen una clase. El comprobador necesita las referencias cuantitativas, que le ayudarán en la selección de casos de prueba y sus objetivos. Las métricas técnicas proporcionan una base desde la cual el análisis, el diseño y la verificación pueden conducirse de manera más objetiva, y ser evaluados más cuantitativamente.

¿Cuáles son los pasos? El primer paso en el proceso de medición es deducir las mediciones de software y métricas que pudieran ser apropiadas para la representación del software en consideración. Después, se recolectan los datos requeridos para aplicar las métricas formuladas. Una vez computados, se analizan basándose en orientaciones preestablecidas y en datos anteriores. Los resultados del análisis son interpretados para obtener una visión inherente a la calidad del software, y los resultados de la interpretación conducen a la modificación de resultados de trabajo deducidos del análisis, diseño, codificación o prueba.

¿Cuál es el producto obtenido? Las métricas de software que se calculan mediante los datos recolectados de los modelos de análisis y diseño, código fuente y casos de prueba.

¿Cómo puedo estar seguro de que lo he hecho correctamente? Debe establecer los objetivos de las mediciones antes de que la recolección de datos comience, definiendo cada métrica OO de manera concreta. Definir solamente algunas métricas y después utilizarlas, para obtener una vista inherente a la calidad de un producto de ingeniería de software.

24.1 EL PROPÓSITO DE LAS MÉTRICAS ORIENTADAS A OBJETOS

Los objetivos primarios para las métricas orientadas a objetos no son diferentes de aquellos de las métricas desarrolladas para el software convencional:

- para entender mejor la calidad del producto.
- para evaluar la efectividad del proceso.
- para mejorar la calidad del trabajo llevado a cabo al nivel del proyecto.

Cada uno de estos objetivos es importante, pero para el ingeniero de software la calidad del producto debe ser lo más importante. Pero ¿cómo medir la calidad de un sistema OO? ¿Qué característicasdel modelo de diseño pueden y deben evaluarse para determinar si el sistema será fácil de implementar, fácil de verificar, simple de modificar **y,** lo más importante, aceptable para los usuarios finales? Estas preguntas serán contestadas en la parte restante del capítulo.

24.2 CARACTERÍSTICAS DISTINTIVAS DE LAS MÉTRICAS ORIENTADAS A OBJETOS

Las métricas para cualquier producto de ingeniería son reguladas por las características únicas del producto. Por ejemplo, sería inútil o sin sentido contar las millas por galón de consumo para un automóvil eléctrico. La métrica es firme para los automóviles convencionales (es decir, impulsados por sistemas de combustión interna a gasolina) pero no se aplica cuando el modo de propulsión cambia radicalmente. El software orientado a objetos es fundamentalmente diferente del software desarrollado con el uso de métodos convencionales. Por esta razón, las métricas para sistemas OO deben ser afinadas a las características que distinguen al software OO del software convencional.

Berard [BER95] define cinco características que regulan las métricas especializadas: Localización, encapsulación, ocultamiento de información, herencia **y** técnicas de abstracción de objetos. Cada una de estas características se discute brevemente en las secciones siguientes'.

Referencia Web

Una extensión de búsqueda para métricas OO la puedes obtener en: **www.objenv.com /cetus / oo_metrics.html**

24.2.1. Localización

La localización es una característica del software que indica la manera en que la información se concentra en un programa. Por ejemplo, los métodos convencionales para la descomposición funcional organizan la información en torno a las funciones que son típicamente implementadas como módulos procedimentales. Los métodos manejados por datos localizan la información en tomo a estructuras de datos específicas. En el contexto **OO,** la información se concentra por la encapsulación de datos y procesos dentro de los límites de una clase u objeto.

Ya que el software convencional enfatiza la función como un mecanismo de localización, las métricas de software se han enfocado a la estructura interna o funciones de complejidad (por ejemplo, longitud del módulo, cohesión o complejidad ciclomática), o a la manera como las funciones se conectan entre sí (acoplamiento de módulos).

Referencia cruzada

Métricos técnicas para software convencional se describen en el Capítulo 19.

Puesto que la clase es la unidad básica de un sistema **OO,** la localización se basa en los objetos. Por esta razón, las métricas deben aplicarse a la clase (objeto), como a una entidad completa. En suma, la relación entre operaciones (funciones) y clases no es necesariamente uno a uno. Por lo tanto, las métricas que reflejan la manera en la que las clases colaboran deben ser capaces de acomodarse a relaciones uno a muchos y muchos a uno.

24.2.2. Encapsulación

Berard [BER95] define la encapsulación como el «empaquetamiento (o ligamento) de una colección de elementos. Ejemplos de bajo nivel de encapsulación [para software convencional] incluyen registros y matrices, [y] subprogramas (por ejemplo, procedimientos, funciones, subrutinas **y** párrafos), son mecanismos de nivel medio para la encapsulación.»

Referencia cruzada

Los conceptos básicos de diseño se describen en el Capítulo 13. Su aplicación al software OO se discute en el Capítulo 20.

Para los sistemas **OO,** la encapsulación engloba las responsabilidades de una clase, incluyendo sus atributos **(y** otras clases para objetos agregados) y operacio-

[1] Este estudio ha sido adaptado de [BER95]

nes, y los estados de la clase, definidos por valores de atributos específicos.

La encapsulación influye en las métricas, cambiando el enfoque de las mediciones de un módulo simple, a un paquete de datos (atributos) y módulos de proceso (operaciones).

En suma, la encapsulación eleva la medición a un nivel de abstracción más alto.

Por ejemplo, más adelante en este capítulo se introducirán las métricas asociadas con el número de operaciones por clase. Contrasta este nivel de abstracción con las métricas que se centran en contar condiciones booleanas (complejidad ciclomática) o en contar líneas de código.

24.2.3. Ocultación de información

La ocultación de información suprime (u oculta) los detalles operacionales de un componente de programa. Solo se proporciona la información necesaria para acceder al componente a aquellos otros componentes que deseen acceder.

Un sistema OO bien diseñado debe implementar ocultación de información. Por esta razón, las métricas que proporcionan una indicación del grado de ocultación logrado suministran un'indicio de la calidad del diseño OO.

24.2.4. Herencia

La herencia es un mecanismo que habilita las responsabilidades de un objeto, para propagarse a otros objetos. La herencia ocurre a través de todos los niveles de una jerarquía de clases. En general, el software convencional no cumple esta característica.

Ya que la herencia es una característica vital en muchos sistemas **OO,** muchos métodos OO se centran en ella. Los ejemplos (discutidos más adelante en este capítulo) incluyen múltiples hijos (instancias inmediatas de una clase), múltiples padres (generalizaciones inmediatas), y jerarquías de clase a un nivel de anidamiento (profundidad de una clase en la jerarquía de herencia).

24.2.5. Abstracción

La abstracción es un mecanismo que permite al diseñador concentrarse en los detalles esenciales de un componente de programa (ya sean datos o procesos), prestando poca atención a los detalles de bajo nivel. Como Berard declara: «la abstracción es un concepto relativo. A medida que se mueve a niveles más altos de abstracción, se ignoran más y más detalles, es decir, se tiene una visión más general de un concepto o elemento. **A** medida que se mueve a niveles de abstracción más bajos, se introducen más detalles, es decir, se tiene una visión más específica de un concepto o elemento.»

Ya que una clase es una abstracción, que puede visualizarse a diferentes niveles de detalle de diferentes maneras (por ejemplo, como una lista de operaciones, como una secuencia de estados, como una serie de colaboraciones), las métricas OO representan abstracciones en términos de mediciones de una clase (por ejemplo, número de instancias por clase por aplicación, número o clases parametrizadas por aplicación, y proporción de clases parametrizadas con clases no parametrizadas).

24.3 MÉTRICAS PARA EL MODELO DE DISEÑO OO

Mucho acerca del diseño orientado a objetos es subjetivo —un diseñador experimentado «sabe» como caracterizar a un sistema **OO,** para que implemente efectivamente los requerimientos del cliente—. Pero, cuando un modelo de diseño OO crece en tamaño y complejidad, una visión más objetiva de las características del diseño puede beneficiar al diseñador experimentado (que adquiere vista adicional), y al novato (que obtiene indicadores de calidad que de otra manera no estarían disponibles).

¿Qué características pueden medirse cuando se evalúa un diseño OO?

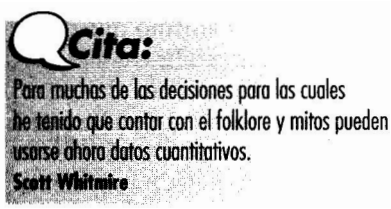
Como parte de un tratamiento detallado de las métricas de software para sistemas **OO,** Whitmire [WHI97] describe nueve características distintas y medibles de un diseño **OO:**

Tamano. El tamaño se define en términos de cuatro enfoques: población, volumen, longitud y funcionalidad. La población se mide haciendo un recuento de las entidades **OO,** como las clases u operaciones. Las medidas de volumen son idénticas a las de población, pero se realizan dinámicamente - e n un instante de tiempo dado— . La longitud es la medida de una cadena de elementos de diseño interconectados (por ejemplo, la profundidad de un árbol de herencia es una medida de longitud). Las métricas de *funcionalidad* proporcionan una indicación indirecta del valor entregado al cliente por una aplicación OO.

Complejidad. Así como el tamaño, existen diferentes enfoques de la complejidad del software [ZUS97]. Whitmire la define en términos de características estructurales, examinando cómo se interrelacionan las clases de un diseño OO con otras.

Acoplamiento. Las conexiones físicas entre los elementos del diseño OO (por ejemplo, el número de colaboraciones entre clases o el número de mensajes intercambiados entre objetos), representan el acoplamiento dentro de un sistema OO.

Suficiencia. Whitmire define la suficiencia como «el grado en que una abstracción posee los rasgos mínimos necesarios, o el grado en que una componente de diseño posee características en su abstracción, desde el punto de vista de la aplicación actual». Dicho de otro modo, se hace la pregunta: «¿qué propiedades tiene que poseer esta abstracción (clase) para que sea Útil?» [WHI97]. En esencia, un componente de diseño (por ejemplo, una clase) es suficiente si refleja completamente todas las propiedades del objeto dominio de la aplicación que se modela; esto es lo que significa que la abstracción (clase), posea los rasgos imprescindibles.



Integridad. La Única diferencia entre integridad y suficiencia es «el conjunto de características, contra las que se comparan la abstracción o componente de diseño [WHI97]». La suficiencia compara la abstracción, desde el punto de vista de la aplicación en cuestión. La integridad considera muchos puntos de vista, preguntándose: «¿Qué propiedades se requieren para representar completamente el objeto dominio del problema?» Ya que los criterios de integridad consideran diferentes puntos de vista, hay una implicación indirecta, acerca del grado en que la abstracción o componente de diseño puede ser reutilizada.

Cohesión. Así como su correspondiente en el software convencional, un componente OO debe diseñarse de manera que posea todas las operaciones trabajando conjuntamente para alcanzar un propósito Único y bien definido. La cohesión de una clase se determina examinando el grado en que «el conjunto de propiedades que posee sea parte del diseño o dominio del problema» [WHI97].

Referencia Web

Un informe técnico de la NASA, que aborda métricas de calidad para sistemas OO, puede descargarse del satc.gsfc.nasa.gov/support/index.html

Originalidad. Una característica similar a la simplicidad, la originalidad (aplicada tanto a operaciones como a clases) es el grado en que una operación es atómica; esto significa que la operación no puede ser construida fuera de una secuencia de otras operaciones contenidas dentro de una clase. Una clase que exhibe un alto grado de originalidad encapsula solamente las operaciones primitivas.

Similitud. Esta métrica indica el grado en que dos o más clases son similares en términos de estructura, comportamiento, función o propósito.

Volatilidad. Como se mencionó anteriormente en este libro, pueden ocurrir cambios en el diseño cuando se modifiquen los requisitos, o cuando ocurran modificaciones en otras partes de la aplicación, resultando una adaptación obligatoria del componente de diseño en cuestión. La volatilidad de un componente de diseño OO mide la probabilidad de que un cambio ocurra.

La descripción de métricas de Whitmire para estas características de diseño se encuentra fuera del ámbito de este libro.'Los lectores interesados deben consultar [WHI97], para más detalles.

En realidad, las métricas técnicas para sistemas OO pueden aplicarse no sólo al modelo de diseño, sino también al modelo de análisis. En las secciones siguientes, se exploran las métricas que proporcionan un indicador de calidad, al nivel de las clases OO y al nivel de operación. En suma, las métricas aplicables al manejo de proyectos y pruebas también se comentarán.

La clase es la unidad fundamental de un sistema OO.

24.4 MÉTRICAS ORIENTADAS A CLASES

Por esta razón, las medidas y métricas para una clase individual, la jerarquía de clases y las colaboraciones de clases poseen un valor incalculable, para el ingeniero del software que ha de evaluar la calidad del diseño. En capítulos anteriores se estudió que la clase encapsula operaciones (procesamiento) y atributos (datos). **Así** mismo, la clase «padre» es de las que heredan las subclases (algunas veces llamadas hijas), sus atributos y operaciones. La clase, normalmente, colabora con otras clases. Cada una de estas características pueden usarse como base de la medición[2],

[2] Nótese que la validez de algunas métricas, discutidas en este capítulo, actualmente se debate en la literatura técnica. Aquellos que abanderan la teoría de medición requieren de un grado de formalismo que algunas de las métricas OO no proporcionan. De cualquier manera, es razonable declarar que todas las métricas proporcionan una visión útil para el ingeniero de software.

24.4.1. La serie de métricas CK

Uno de los conjuntos de métricas OO más ampliamente referenciados, ha sido el propuesto por Chidamber y Kemcrer [CHI94]. Normalmente conocidas como la ***serie de métricas*** *CK*, los autores han propuesto seis métricas basadas en clases para sistemas OO[3].

Métodos ponderados por clase (MPC). Asumen que *n* métodos de complejidad c_1, c_2, ... c_n se definen para la clase **C**. La métrica de complejidad específica que se eligió (por ejemplo, complejidad ciclomática) debe normalizarse de manera que la complejidad nominal para un método toma un valor de 10.

$$MPC = \Sigma c_i$$

para cada $i = 1$ hasta n.

El número de métodos y su complejidad son indicadores razonables de la cantidad de esfuerzo requerido para implementar y verificar una clase. En suma, cuanto mayor sea el número de métodos, más complejo es el árbol de herencia (todas las subclases heredan métodos de sus padres). Finalmente, a medida que crece el número de métodos para una clase dada, más probable es que se vuelvan más y más específicos de la aplicación, así que se limita el potencial de reutilización. Por todas estas razones, el MPC debe mantenerse tan bajo como sea razonable.

El número de métodos y su complejidad está directamente relacionado con el esfuerzo requerido para comprobar una clase.

Aunque podría parecer relativamente claro llevar la cuenta del número de métodos en una clase, el problema es más complejo de lo que parece, Churcher y Shepperd [CHU95] discuten este aspecto cuando escriben:

> Para contar métodos, se debe contestar **la** pregunta fundamental: «¿pertenece un método únicamente a la clase que lo define, o pertenece a cada clase que la hereda directa o indirectamente?». Las preguntas como esta pueden parecer triviales, **ya** que el sistema de ejecución **las** resolverá finalmente. De cualquier manera, las implicaciones para las métricas deben ser significativas.
>
> Una posibilidad es restringir el contador de **la** clase actual ignorando los miembros heredados. La motivación para esto podría ser que los miembros heredados **ya** han sido contados en **las** clases donde fueron definidos, **así** que el incremento de **la** clase es **la** mejor medida de su funcionalidad, lo que refleja su razón para existir. Para entender lo que la clase lleva **a** cabo, la fuente de información más importante son sus propias operaciones. Si una clase no puede responder **a** un mensaje (por ejemplo, le falta un método correspondiente de sí), entonces pasará su mensaje **a** sus clases padres.
>
> En el otro extremo, el recuento podría incluir **a** todos aquellos métodos definidos en la clase en cuestión, junto con **los** métodos heredados. Este enfoque enfatiza la importancia del espacio de estados, en lugar del incremento de **la** clase, para **la** comprensión de **la** clase.
>
> Entre estos extremos, existe cierto número de posibilidades diferentes. Por ejemplo, una podría restringir el recuento a la clase en cuestión y los miembros heredados directamente de sus padres. Este enfoque se basa en el argumento de que la especialización de clases padres es lo más directamente relevante en el comportamiento de **las** clases descendientes.

Así como la mayoría de las convenciones de recuento en métricas de software, cualquiera de los enfoques resumidos con anterioridad es aceptable, siempre que el enfoque de recuento sea aplicado consistentemente al momento de recolectar métricas.

Árbol de profundidad de herencia (APH). Esta métrica se define como «la máxima longitud del nodo a la raíz del árbol» [CHI94]. Con referencia a la Figura 24.1, el valor del APH para la jerarquía de clases mostrada es de 4.

A medida que el APH crece, es posible que clases de más bajos niveles heredarán muchos métodos. Esto conlleva dificultades potenciales, cuando se intenta predecir el comportamiento de una clase. Una jerarquía de clases profunda (el APH es largo) también conduce a una complejidad de diseño mayor. Por el lado positivo, los valores APH grandes implican un gran número de métodos que se reutilizarán.

Número de descendientes (NDD). Las subclases inmediatamente subordinadas a una clase en la jerarquía de clases se denominan sus descendientes. Con referencia a la Figura 24.1, la clase C2 tiene tres descendientes —subclases C21, C22 y C23—.

A medida que el número de descendientes crece, la reutilización se incrementa, pero además es cierto que cuando el NDD crece, la abstracción representada por la clase predecesora puede diluirse. Esto significa que existe una posibilidad de que algunos descendientes no sean miembros, realmente apropiados, de la clase predecesora. **A** medida que el NDD crece, la cantidad de pruebas (requeridas para ejercitar cada descendiente en su contexto operativo) se incrementará también.

La herencia es una habilidad extremadamente poderasa, que puede meterlo en problemas, si no se usa ron cuidada. Utilice el APH y otros métricas relacionadas para darse una lectura de la complejidad de la jerarquía de clases.

[3] Chidamber, Darcy y Kemerer usan el término *métodos* en vez de *operaciones*. Su utilización del término es reflejada en esta sección.

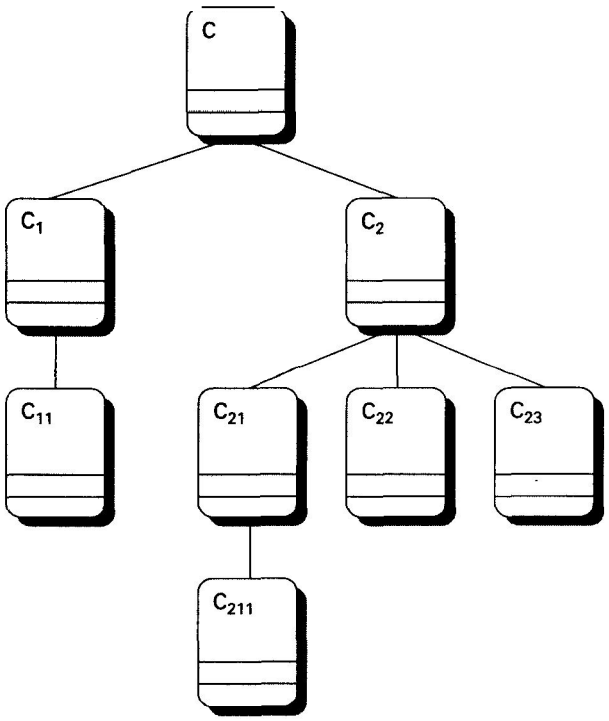


FIGURA 24.1. Una jerarquía de **clases.**

Acoplamiento entre clases objeto (ACO). El modelo CRC (Capítulo 21) debe utilizarse para determinar el valor de ACO. En esencia, ACO es el número de colaboraciones listadas para una clase, en la tarjeta índice CRC.

A medida que ACO se incrementa, es más probable que el grado de reutilización de una clase decrezca. Valores altos de ACO además complican las modificaciones y las pruebas, que se producen cuando se realizan modificaciones. En general, los valores de ACO para cada clase deben mantenerse tan bajos como sea razonable. Esto es consistente con la regla general para reducir el acoplamiento, para el software convencional.

CONSEJO

los conceptos de acoplamiento y cohesión se aplican tanto al software convencional como al OO. Mantenga el acoplamiento de clases bajo y la cohesión de operación alto.

Respuesta para una clase (RPC). El conjunto de respuesta de una clase es «una serie de métodos que pueden potencialmente ser ejecutados, en respuesta a un mensaje recibido por un objeto, en la clase» [CHI94]. RPC se define como el número de métodos en el conjunto de respuesta.

A medida que la RPC aumenta, el esfuerzo requerido para la comprobación también se incrementa, ya que la secuencia de comprobación (Capítulo 23) se incrementa también. Así mismo, se dice que, así como la RPC aumenta, la complejidad del diseño global de la clase se incrementa.

Carencia de cohesión en los métodos (CCM). Cada método dentro de una clase, C, accede a uno o más atributos (también llamados variables de instancia) CCM es el número de métodos que accede a uno o más de los mismos atributos[4]. Si no existen métodos que accedan a los mismos atributos, entonces CCM = 0.

Para ilustrar el caso en el que CCM es diferente de 0, considérese una clase con 6 métodos. Cuatro de los métodos tienen uno o más atributos en común (es decir, acceden a atributos comunes). De esta manera, CCM = 4.

Si CCM es alto, los métodos deben acoplarse a otro, por medio de los atributos. Esto incrementa la complejidad del diseño de clases. En general, los valores altos para CCM implican que la clase debe diseñarse mejor descomponiendo en dos o más clases distintas. Aunque existan casos en los que un valor alto para CCM es justificable, es deseable mantener la cohesión alta, es decir, mantener CCM bajo.

Cita:

Las ponderaciones orientadas a objetos, son una parte integral de la tecnología de objetos y de la buena práctica de la ingeniería de software
Brian Henderson-Sellers

24.4.2. Métricas propuestas por Lorenz y Kidd

En su libro sobre métricas **OO,** Lorenz y Kidd [LOR94] separan las métricas basadas en clases en cuatro amplias categorías: tamaño, herencia, valores internos y valores externos. Las métricas orientadas al tamaño para las clases OO se centran en el recuento de atributos y operaciones para cada clase individual, y los valores promedio **para** el sistema OO como un todo. Las métricas basadas en la herencia se centran en la forma en que las operaciones se reutilizan en la jerarquía de clases. Las métricas para valores internos de clase examinan la cohesión (Sección 24.4.1) y los aspectos orientados al código; las métricas orientadas a valores externos, examinan el acoplamiento y la reutilización. A continuación, una muestra de métricas propuestas por Lorenz y Kidd [5]:

Tamano de clase (TC). El tamaño general de una clase puede medirse determinando las siguientes medidas:

- el total de operaciones (operaciones tanto heredadas como privadas de la instancia), que se encapsulan dentro de la clase.
- el número de atributos (atributos tanto heredados como privados de la instancia), encapsulados por la clase.

[4] **La** definición formal es un poco más compleja. Véase [CHI94] para más detalle.

[5] Para un tratamiento más completo, véase [LOR94].

Durante la revisión del modelo de AOO, las tarjetas CRC proporcionan una indicación razonable de las valores esperados paro TC. Si encuentra uno clase con demasiada responsobilidod durante AOO, considere el particionarla.

La métrica MPC propuesta por Chidamber y Kemerer (Sección 24.4.1) es también una métrica ponderada del tamaño de clase.

Como se indicó con anterioridad, valores grandes para TC indican que la clase debe tener bastante responsabilidad. Esto reducirá la reutilización de la clase y complicará la implementación y las pruebas. En general, operaciones y atributos heredados o públicos deben ser ponderados con mayor importancia, cuando se determina el tamaño de clase. [LOR94] Operaciones y atributos privados, permiten la especialización y son más propios del diseño.

También se pueden calcular los promedios para el número de atributos y operaciones de clase. Cuanto menor sea el valor promedio para el tamaño, será más posible que las clases dentro del sistema puedan reutilizarse.

Número de operaciones redefinidas para una subclase (NOR). Existen casos en que una subclase reemplaza una operación heredada de su superclase por una versión especializada para su propio uso. A esto se le llama *redefinición*. Los valores grandes para el NOR, generalmente indican un problema de diseño. Tal como indican Lorenz y Kidd:

> Dado que una subclase debe ser **la** especialización de sus superclases, deben, sobre todo, extender los servicios (operaciones) de **las** superclases. Esto debe resultar en nuevos nombres de métodos únicos.

Si el NOR es grande, el diseñador ha violado la abstracción representada por la superclase. Esto provoca una débil jerarquía de clases y un software **OO,** que puede ser difícil de probar y modificar.

Número de operaciones añadidas por una subclase (NOA). Las subclases se especializan añadiendo operaciones y atributos privados. A medida que el valor NOA se incrementa, la subclase se aleja de la abstracción representada por la superclase. En general, a medida que la profundidad de la jerarquía de clases incrementa (APH se vuelve grande), el valor para NOA a niveles más bajos en la jerarquía debería disminuir.

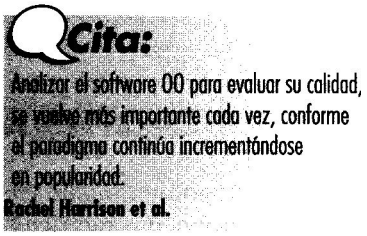
Índice de especialización (IES). El índice de especialización proporciona una indicación aproximada del grado de especialización, para cada una de las subclases en un sistema OO. La especialización se puede alcanzar añadiendo o eliminando operaciones, pero también redefiniendo.

$$IE = [NOR \times nivel\ ] / M_{total}$$

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

24.4.3. La colección de métricas MDOO

Harrison, Counseil y Nithi [HAR98] propusieron un conjunto de métricas para el diseño orientado a objetos (MDOO), que proporcionan indicadores cuantitativos para el diseño de características OO. A continuación, una muestra de métricas MDOO:



Factor de herencia de métodos (FHM). El grado en que la arquitectura de clases de un sistema OO hace uso de la herencia tanto para métodos (operaciones) como atributos, se define como:

$$FHM = \Sigma M_i(C_i) / \Sigma M_a(C_i)$$

donde el sumatorio va desde i=l hasta TC. TC se define como el número total de clases en la arquitectura, C_i es una clase dentro de la arquitectura, y

$$M_a(C_i) = M_d(C_i) + M_i(C_i)$$

donde

$M_a(C_i)$ = al número de métodos que pueden ser invocados en relación con C_i

$M_d(C_i)$ = al número de métodos declarados en la clase C_i

$M_i(C_i)$ = al número de métodos heredados (y no redefinidos) en C_i

El valor de FHM (el factor de herencia de atributo, FHA, se define de manera análoga), proporciona una referencia sobre impacto de la herencia en software OO.

Factor de acoplamiento (FA). Con anterioridad, en este capítulo se indicó que el acoplamiento es un indicador de las conexiones entre los elementos del diseño OO. La colección de métricas MDOO define el factor de acoplamiento de la siguiente manera:

$$FA = [\Sigma_i \Sigma_j\ es\_cliente\ (C_i, C_j)] / (TC^2 - TC)$$

donde los sumatorios van desde $i = 1$ hasta TC y desde $j = 1$ hasta TC. La función

$Es\ cliente = 1$, si existe una relación entre la clase cliente, C_c y la clase servidora C_s y $C_c \neq C_s$.

$Es\_cliente = 0$, en cualquier otro caso.

A pesar de que muchos factores afectan la complejidad, comprensión y mantenimiento del software, es razonable concluir que conforme el valor del **FA** crece, la complejidad del software OO también crece, y la comprensión, el mantenimiento y el potencial de reutilización, pueden resentirse como resultado.

Factor de polimorfismo (FP). Harrison, Counsell y Nithi [HAR98] definen FP como «el número de métodos que redefinen métodos heredados, dividida por el máximo número de posibles situaciones polimórficas distintas...; de esta manera, el FP es una medida indirecta de la cantidad relativa de ligadura dinámica en un sistema». La colección de métricas MDOO define el FP de la siguiente manera:

$$FP = \Sigma_i M_o(C_i) / \Sigma_i [M_n(C_i) \times DC(C_i)]$$

donde los sumatorios van desde $i = 1$ hasta TC y

$$M_d(C_i) = M_n(C_i) + M_o(C_i)$$

y

$M_n(C_i)$ = al número de métodos nuevos.
$M_o(C_i)$ = al número de métodos redefinidos.
$DC(C_i)$ = al número de descendientes (el número de clases descendientes de una clase base).

Harrison, Counsell y Nithi [HAR98] presentan un análisis detallado de FHM, FA y **FP** en conjunto con otras métricas, y examinan su validez de uso, en la evaluación de la calidad del diseño.

24.5 MÉTRICAS ORIENTADAS A OPERACIONES

Ya que la clase es la unidad dominante en los sistemas **OO,** se han propuesto menos métricas para operaciones que las relacionadas con las clases. Churcher y Shepperd [CHU95] discuten lo anterior cuando declaran que:

> **Resultados de estudios recientes indican que los métodos tienden a ser pequeños, tanto en términos de número de sentencias como en complejidad lógica [WIL93], sugiriendo que la estructura de conectividddde un sistema debe ser mas importante que el contenido de los módulos individuales.**

De cualquier modo, existen algunas ideas que pueden llegar a apreciarse, examinando las características promedio para los métodos (operaciones). A continuación se resumen tres simples métricas, propuestas por Lorenz y Kidd [LOR94]:

Tamaño medio de operación (TO_{medio}). Aunque las líneas de código podrían ser usadas como un indicador para el tamaño de operación, la medida LDC adolece de todos los problemas discutidos en el Capítulo 4. Por esta razón, el número de mensajes enviados por la operación proporciona una alternativa para el tamaño de operación. A medida que el número de mensajes enviados por una sola operación se incrementan, es más probable que las responsabilidades no hayan sido correctamente asignadas dentro de la clase.

Referencia cruzada

Las métricas pueden aplicarse a nivel de componentes, pero también pueden aplicarse a operaciones. Véase el Capítulo 19 para más detalles.

Complejidad de operación (CO). La complejidad de una operación puede ser calculada usando cualquiera de las métricas de complejidad (Capítulo 19) propuestas para el software convencional [ZUS90]. Ya que las operaciones deben limitarse a una responsabilidad específica, el diseñador debería esforzarse por mantener la CO tan baja como sea posible.

Número de parámetros de media por operación (NP_{media}). Tan largo como sea el número de parámetros de operación, más compleja será la colaboración entre objetos. En general, NP_{media} debe mantenerse tan baja como sea posible.

24.6 MÉTRICAS PARA PRUEBAS ORIENTADAS A OBJETOS

Las métricas de diseño anotadas en las Secciones 24.4 y 24.5 proporcionan una indicación de la calidad de diseño. También proveen una indicación general de la cantidad de esfuerzo de pruebas requerido para probar un sistema OO.

Binder [BIN94] sugiere que una amplia gama de métricas de diseño tienen una influencia directa en la «comprobabilidad» de un sistema OO. Las métricas se organizan en categorías, que reflejan características de diseño importantes.

Encapsulación

Carencia de cohesión en métodos (CCM)[6]. Cuanto más alto sea el valor CCM será necesario probar más estados para asegurar que los métodos no generan efectos colaterales.

[6] **Véase la Sección** 24.4.1 **para una descripción de** CCM

Porcentaje público y protegido (PPP). Los atributos públicos que se heredan de otras clases son visibles para esas clases. Los atributos protegidos son una especialización y son privados a subclases específicas. Esta métrica indica el porcentaje de atributos de una clase que son públicos. Valores altos para PPP incrementan la probabilidad de efectos colaterales entre clases. Las pruebas deben diseñarse para asegurar que ese tipo de efectos colaterales sean descubiertos.

Acceso público a datos miembros (APD). Esta métrica indica el número de clases (o métodos) que pueden acceder a los atributos de otras clases, una violación de encapsulación. Valores altos para APD producen potencialmente efectos colaterales entre clases. Las pruebas deben diseñarse para estar seguros de que ese tipo de efectos colaterales serán descubiertos.

Herencia

Número de clases raíz (NCR). Esta métrica es un recuento de las distintas jerarquías de clases, que se describen en el modelo de diseño. Se deben desarrollar las colecciones de pruebas para cada clase raíz, y la jerarquía de clases correspondiente. A medida que el NCR se incrementa, el esfuerzo de comprobación también se incrementa.

La comprobación OO puede ser un poco compleja. las métricas pueden ayudarle a la asignación de recursos de pruebas a hilos, escenarios y grupas de clases, que son «sujetos» basados en características ponderadas. Utilícelas.

Número de Padres Directos (NPD). Cuando es utilizado en el contexto OO, el NPD es una indicación de herencia múltiple. NPD > 1 indica que la clase hereda sus atributos y operaciones de más de una clase raíz. Se debe evitar que NPD > 1 tanto como sea posible.

Número de descendientes (NDD) y árbol de profundidad de herencia (APH)[7]. Tal como se explicó en el Capítulo 23, los métodos de la superclase tendrán que ser probados nuevamente para cada subclase.

Además de las métricas anteriores, Binder [BIN94] también define métricas para la complejidad y polimorfismo de las clases. Las métricas definidas para la complejidad de clase, incluyen tres métricas CK (Sección 24.4.1): Métodos ponderados por clase (MPC), el acoplamiento entre clases de objetos (ACO) y la respuesta para una clase (RPC). En resumen, también se definen las métricas asociadas con el recuento de métodos. Las métricas asociadas con el polimorfismo se especifican en detalle. Lo mejor es dejar su descripción a Binder.

24.7 MÉTRICAS PARA PROYECTOS ORIENTADOS A OBIETOS

Como se presentó en la Parte Dos de este libro, el trabajo del jefe de proyecto es planear, coordinar, registrar y controlar un proyecto de software. En el Capítulo 20 se presentaron algunos de los aspectos especiales asociados con la gestión de proyecto para proyectos OO. Pero ¿qué hay acerca de las métricas'? ¿Existen métricas OO especializadas que puedan ser utilizadas por el jefe de proyecto para proporcionar una visión interna adicional sobre el progreso de su proyecto? [8]. La respuesta, desde luego es «sí».

La primera actividad ejecutada por el jefe de proyecto es planificar, y una de las primeras tareas de planificación es la estimación. Retomando el modelo evolutivo de procesos, la planificación se vuelve a revisar después de cada iteración del software. De este modo, la planificación (y sus estimaciones de proyecto) es revisada nuevamente después de cada iteración de AOO, DOO e incluso POO.

Uno de los aspectos clave, al que debe hacer frente un jefe de proyecto durante la planificación, es una estimación del tamaño de implementación del software. El tamaño es directamente proporcional al esfuerzo y la duración. Las siguientes métricas [LOR94] pueden proporcionar una visión sobre el tamaño del software:

Referencia cruzada

La aplicabilidad de un modelo de procesos evolutivos, llamado el modelo recursivo/paralelo, se discute en el Capítulo 20.

Número de escenario (NE). El número de escenarios o casos uso (Capítulos 11 y 21) es directamente proporcional al número de clases requeridas para cubrir los requisitos, el número de estados para cada clase, el número de métodos, atributos y colaboraciones. El NE es un importante indicador del tamaño de un programa.

Número de clases clave (NCC). Una clase clave se centra directamente en el dominio del negocio para el problema, y tendrá una menor probabilidad de ser imple-

[7] Véase la Sección 24.4.1 para una descripción de NCC y APM

[8] Una descripción interesante de la colección de métricas CK (Sección 24.4.1) para el uso en la administración de la toma de decisiones puede encontrarse en [CHI98].

mentada por medio de la reutilización[9]. Por esta razón, valores altos para NCC indican gran trabajo de desarrollo substancial. Lorenz y Kidd [LOR94] sugieren que entre el 20 y el 40 por 100 de todas las clases en un sistema OO típico corresponde a las clases clave. El resto es infraestructura de soporte (GUI, comunicaciones, bases de datos, etc.).

Número de subsistemas (NSUB). El número de subsistemas proporciona una visión sobre la asignación de recursos, la planificación (con énfasis particular en el desarrollo paralelo) y el esfuerzo de integración global.

Las métricas NE, NCC y NSUB pueden recolectarse sobre proyectos OO pasados, y están relacionados con el esfuerzo invertido en el proyecto como un todo, y en actividades de procesos individuales (por ejemplo, AOO, DOO, POO y pruebas **OO).** Estos datos pueden también utilizarse junto con métricas de diseño discutidas con anterioridad en este capítulo, para calcular «métricas de productividad», tales como el número de clases promedio por desarrollador o promedio de métodos por persona/mes. Colectivamente, estas métricas pueden usarse para estimar el esfuerzo, duración, personal y otra información de proyecto para el proyecto actual.

RESUMEN

El software orientado a objetos es fundamentalmentediferente al software desarrollado con el uso de métodos convencionales. Es por esto que las métricas para sistemas OO se enfocan en la ponderación que puede aplicarse a las clases y a las características del diseño —localización, encapsulación, ocultamiento de información, herencia y técnicas de abstracción de objetos—, que definen a la clase como única.

La colección de métricas CK define seis métricas de software orientadas a la clase que se centran en la clase y en la jerarquía de clases. La colección de métricas también incorpora métricas para evaluar las colaboraciones entre clases y la cohesión de métodos que residen dentro de la clase. Al nivel orientado a clases, la colección CK puede complementarse con las métricas propuestas por Lorenz y Kidd y la colección de métricas MDOO. Estas incluyen ponderaciones de «tamaño» de clase, y otras métricas que proporcionan una visión acerca del grado de especialización de las subclases.

Las métricas orientadas a operaciones se centran en el tamaño y complejidad de las operaciones individuales. Sin embargo, es importante hacer notar que la primera para las métricas de diseño OO es a nivel de clases.

Se ha propuesto una amplia variedad de métricas OO para evaluar la comprobabilidad de un sistema OO. Estas métricas se centran en la encapsulación, herencia, complejidad de las clases y polimorfismo. Muchas de estas métricas han sido adaptadas de la colección CK y de las métricas propuestas por Lorenz y Kidd. Otras han sido propuestas por Binder.

Las características ponderables del modelo de análisis y diseño pueden ayudar al jefe de proyecto de un sistema OO en la planificación y registro de las actividades. El número de escenarios (casos de uso), clases clave y subsistemas proporcionan información acerca del nivel de esfuerzo requerido para implementar el sistema.

REFERENCIAS

[BER95] Berard, E., *Metrics for Object-Oriented Software Engineering,* publicado en internet en **comp.software-eng,** 28 de enero de 199.5.

[CHI94] Chidamber, S.R., y C.F. Kemerer, «A Metrics Suite for Object-Oriented Design», *IEEE Trans. Software Engineering*, vol. 20, n.° 6, Junio de 1994, pp. 476-493.

[CHI98] Chidamber, S.R., D.P. y C.F. Kemerer, «Management Use of Metrics for Object-Oriented Software: An Exploratory Analysis», *IEEE Trans. Software Engineering,* vol. 24, n.° 8, Agosto de 1998, pp. 629-639.

[CHU95] Churcher, N.I., y **M.J.** Shepperd, «Towards a Conceptual Framework for Object-Oriented Metrics», ***ACM** Software Engineering Notes,* vol. *20,* n.° 2, Abril de 199.5, pp. 69-76.

[HAR98] Harrison, R., **S.J.** Counseil y R.V. Nithi, «An Evaluation of the MOOD Set of Object-Oriented Software Metrics», *IEEE Trans. Software Engineering,* vol. 24, n.° 6, Junio de 1998, pp. 491-496.

[LOR94] Lorenz, M., y J. Kidd, *Object-Oriented Software Metric,* Prentice-Hall, 1994.

[WHI97] Whitmire, S., *Object-Oriented Design Measuremente,* Wiley, 1997.

[ZUS90] Zuse, H., *Software Complexity: Measures and Methods,* DeGruyter, Nueva York, 1990.

[ZUS97] Zuse, H., *A framework of Software Mesurement,* DeGruyter, Nueva York, 1997.

[9] Esto sólo es verdad hasta que una robusta librería de componentes reutilizables se desarrolla para un dominio particular.

PROBLEMAS Y PUNTOS A CONSIDERAR

24.1. Revise las métricas presentadas en este capítulo y en el Capítulo 19. ¿Cómo podía caracterizar las diferencias semánticas y sintácticas entre las métricas para software convencional y OO?

24.2. ¿Cómo es que la localización afecta las métricas desarrolladas para software convencional y OO?

24.3. ¿Por qué no se hace más énfasis en las métricas OO que abordan las características específicas de las operaciones residentes dentro de una clase?

24.4. Revise las métricas descritas en este capítulo y sugiera algunas que aborden directa o indirectamente el ocultamiento de información.

24.5. Revise las métricas descritas en este capítulo y sugiera algunas que aborden directa o indirectamente la abstracción.

24.6. Una clase **x** posee 12 operaciones. Se ha calculado la complejidad ciclomática para todas las operaciones del sistema OO y el valor promedio de la complejidad del módulo es **4.** Para la clase **x** la complejidad de las operaciones de la 1 a la 12 es 5,4,3,6,8,2,2,5,5,4,4 respectivamente. Calcular MPC.

24.7. Con respecto a las Figuras 20.8a y b, calcule el valor APH para cada uno de los árboles de herencia. ¿Cuál es el valor de NDD para la clase **x2** de ambos árboles?

24.8. Acuda a [CHI94] y presente una descripción de una página referente a la definición formal de la métrica CCM.

24.9. En la Figura 20.8b, ¿cuál es el valor de NOA para las clases **x3** y **x4?**

24.10. Con respecto a la Figura 20.8b suponga que las cuatro operaciones han sido invalidadas en el árbol de herencia (jerarquía de clases), ¿cuál es el valor de IE para esa jerarquía?

24.11. Un equipo de software ha finalizado cinco proyectos hasta la fecha. Los datos siguientes han sido recogidos para todos los tamaños de los proyectos:

Número de proyecto	NGE	NCC	NSUB	Esfuerzo (días)
1	34	60	3	900
2	55	75	6	1.575
3	122	260	8	4.420
4	45	66	2	990
5	80	124	6	2.480

Se dispone de un nuevo proyecto que se encuentra en las primeras fases de **AOO.** Se estima que para este proyecto se desarrollarán 95 casos prácticos. Estimar:

a. el número total de clases que serán necesarias para implementar el sistema;

b. la cantidad total de esfuerzo que será necesaria para implementar el sistema.

24.12. Su profesor le proporciona una lista de métricas OO procedente de este capítulo. Calcule los valores de estas métricas para uno o más de los problemas que se indican a continuación:

a. el modelo de diseño para el diseño *HogarSeguro.*

b. el modelo de diseño para el sistema **SSRB** descrito en el problema 12.13.

c. el modelo de diseño para el juego de vídeo considerado en el problema 22.14.

d. el modelo de diseño para el correo electrónico considerado en el problema 22.15.

e. el modelo de diseño para el problema **CTA** considerado en el problema 22.16.

OTRAS LECTURAS Y FUENTES DE INFORMACIÓN

Una variedad de libros de AOO, DOO y Comprobación OO (véase *Otras lecturas y fuentes de información* en los Capítulos 20, 21 y 22) que hacen referencia de paso a las métricas OO, pero hay pocos que abordan el tema con detalle. Los libros escritos por Jacobson *(Ohject-Oriented Software Engineering,* Addison-Wesley, 1994) y Graham *(Ohjecto-Oriented Methods,* Addison-Wesley, segunda edición, 1993). Proporcionan un tratamiento más extenso que la mayoría.

Whitmire [WHI97] presenta el tratamiento matemática y extensamente más sofisticado de las métricas OO publicadas a la fecha. Lorenz y Kidd [LOR94] y Hendersen-Sellers *(Object-Oriented Metrics: Measures of Complexity,* Prentice-Hall, 1996) ofrecen los únicos libros dedicados a métricas OO. Otros libros dedicados a las métricas de software convencional (véase *otras lecturas y fuentes de información* en los Capítulos 4 y 19) contienen discusiones limitadas de métricas OO.

Una amplia variedad de fuentes de información para métricas orientadas a objetos y temas relacionados se encuentra disponible en intemet. Una lista reciente de referencias a sitios (páginas) web relevantes a las métricas OO pueden encontrarse en **http://www. mhhe.pressman5.com**