UNIVERSIDAD NACIONAL DE LANUS
LICENCIATURA EN SISTEMAS
Ingeniería de Software II
Prof. Tit.: Dr. Ramón García-Martínez
JTP: Lic. Darío Rodríguez

GUIA DE PREGUNTAS

Material "METRICAS TECNICAS DEL SOFTWARE.
Ingeniería del Software (Capítulo 19) de Roger S. Pressman (5ta Edición)"

1. Enuncie puntos de relación entre calidad y requisitos de software.
2. Enuncie factores que afectan a a la calidad del software.
3. Enuncie y de un esquema de clasificación de los factores que afectan a la calidad del software según McCall.
4. Defina factor de calidad del software como relación matemática entre un conjunto de métricas.
5. Enuncie las métricas valorables en la escala del O (bajo) al 10n(alto) del esquema de puntuación propuesto por McCall. Proporcione una tabla de relación entre los factores de calidad del software y las métricas.
6. Enuncie Factores de Calidad del Software FURPS propuestos por Hewlett-Packard.
7. Enuncie los Factores de calidad ISO 9126.
8. Enuncie las actividades que Roche sugiere para un proceso de medición.
9. Enuncie principios que se pueden asociar con la formulación de las métricas técnicas.
10. Enuncie atributos que deberían acompañar a las métricas efectivas del software.
11. Enuncie los conceptos y medidas sobre las que se basan las métricas de cohesión.
12. Enuncie medidas para calcular el acoplamiento de flujo de datos y control, el acoplamiento global y el acoplamiento de entorno. Enuncie la ecuación de calculo de acoplamiento de módulo.
13. Enuncie el índice de madurez propuesto por IEEE que funciona como indicador del producto software.

19 MÉTRICAS TÉCNICAS DEL SOFTWARE

UN elemento clave de cualquier proceso de ingeniería es la medición. Empleamos medidas para entender mejor los atributos de los modelos que creamos. Pero, fundamentalmente, empleamos las medidas para valorar la calidad de los productos de ingeniería o de los sistemas que construimos.

Adiferencia de otras disciplinas, la ingeniería del software no está basada en leyes cuantitativas básicas de la Física. Las medidas absolutas, tales como el voltaje, la masa, la velocidad o la temperatura no son comunes en el mundo del software. En su lugar, intentamos obtener un conjunto de medidas indirectas que dan lugar a métricas que proporcionan una indicación de la calidad de algún tipo de representación del software. Como las medidas y métricas del software no son absolutas, están abiertas a debate. Fenton [FEN91] trata este aspecto cuando dice:

> La medición es el proceso por el que se asignan números o símbolos a los atributos de las entidades en el mundo real, de tal manera que las definan de acuerdo con unas reglas claramente definidas .... En las ciencias físicas, medicina y, más recientemente, en las ciencias sociales, somos ahora capaces de medir atributos que previamente pensábamos que no eran medibles... Por supuesto, tales mediciones no están tan refinadas como las de las ciencias físicas..., pero existen [y se toman importantes decisiones basadas en ellas]. Sentimos que la obligación de intentar «medir lo no medible» para mejorar nuestra comprensión de entidades particulares es tan poderosa en la ingeniería del software como en cualquier disciplina.

Pero algunos miembros de la comunidad de software continúan argumentando que el software no es medible o que se deberían posponer los intentos de medición hasta que comprendamos mejor el software y los atributos que habría que emplear para describirlo. Esto es un error.

VISTAZO RÁPIDO

¿Qué es? Por su naturaleza, la ingeniería es una disciplina cuantitativa. Los ingenieros usan números para ayudarse en el diseño y cálculo del producto a construir. Hasta hace poco tiempo, los ingenieros del software disponían de pocas referencias cuantitativas en su trabajo —pero esto está cambiando—. Las métricas técnicas ayudan a los ingenieros del software a profundizar en el diseño y construcción de los productos que desarrollan.

¿Quién lo hace? Los ingenieros del software usan las métricas técnicas para ayudarse en el desarrollo de software de mayor calidad.

¿Por qué es importante? Siempre habrá elementos cualitativos para la creación del software. El problema estriba en que la valoración cualitativa puede no ser suficiente. Un ingeniero del software necesita criterios objetivos para guiarse en el diseño de datos, de la arquitectura, de las interfaces y de los componentes. El verificador necesita una referencia cuantitativa que le ayude en la selección de los casos de prueba y de sus objetivos. Las métricas técnicas facilitan una base para que el análisis, diseño, codificación y prueba puedan ser conducidas más objetivamente y valoradas más cuantitativamente.

¿Cuáles son los pasos? La primera etapa en el proceso de medida consiste en extraer las medidas y métricas del software que son apropiadas para la representación del software que está siendo considerado. A continuación, se requieren datos para extraer la formulación de métricas agregadas. Una vez calculadas, las métricas apropiadas son analizadas en base a directrices preestablecidas y datos históricos. El resultado del análisis es interpretado para profundizar en la calidad del software, y el resultado de la interpretación orienta las modificaciones a originarse en los resultados obtenidos en el análisis, diseño, codificación y prueba.

¿Cuál es el producto obtenido? Las métricas del software serán calculadas sobre datos agregados del análisis, de los modelos de diseño, del código fuente y de los casos de prueba.

¿Cómo puedo estar seguro de que lo he hecho correctamente? Hay que establecer los objetivos y medidas antes de comenzar la acumulación de datos, definiendo sin ambigüedad cada métrica técnica. Define pocas métricas y utilízalas para profundizar en la calidad del resultado obtenido en la ingeniería del software.

Aunque las métricas técnicas para el software de computadora no son absolutas, nos proporcionan una manera sistemática de valorar la calidad basándose en un conjunto de «reglas claramente definidas». También le proporcionan al ingeniero del software una visión interna en el acto, en vez de a posteriori. Esto permite al ingeniero descubrir y corregir problemas potenciales antes de que se conviertan en defectos catastróficos.

En el Capítulo 4 estudiábamos las métricas del software tal y como se aplican a nivel del proceso y del proyecto. En este capítulo, nuestro punto de atención se desplaza a las medidas que se pueden emplear para valorar la calidad del producto según se va desarrollando. Estas medidas de atributos internos del producto le proporcionan al desarrollador de software una indicación en tiempo real de la eficacia del análisis, del diseño y de la estructura del código, la efectividad de los casos de prueba, y la calidad global del software a construir.

19.1 CALIDAD DEL SOFTWARE

Incluso los desarrolladores del software más hastiados estarán de acuerdo en que un software de alta calidad es una de las metas más importantes. Pero, ¿cómo definimos la calidad? En el Capítulo **8** propusimos diferentes maneras de interpretar la calidad del software e introdujimos una definición que hacía hincapié en la concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, los estándares de desarrollo explícitamente documentados y las características implícitas que se esperan de todo software desarrollado profesionalmente.

Cita:
Todo programa hace algo correctamente, que puede no ser lo que necesitamos que haga.
Anónimo

No cabe duda de que la definición anterior podría modificase o ampliarse y discutirse eternamente. Para los propósitos de este libro, la definición sirve para hacer énfasis en tres puntos importantes:

1. Los requisitos del software son la base de las medidas de la calidad. La falta de concordancia con los requisitos es una falta de calidad[1].
2. Unos estándares específicos definen un conjunto de criterios de desarrollo que guían la manera en que se hace la ingeniería del software. Si no se siguen los criterios, habrá seguramente poca calidad.
3. Existe un conjunto de requisitos implícitos que a menudo no se nombran (por ejemplo, facilidad de mantenimiento). Si el software cumple con sus requisitos explícitos pero falla en los implícitos, la calidad del software no será fiable.

La calidad del software es una compleja mezcla de factores que variarán a través de diferentes aplicaciones y según los clientes que las pidan. En las siguientes secciones, se identifican los factores de la calidad del software y se describen las actividades humanas necesarias para conseguirlos.

19.1.1. Factores de calidad de McCall

Los factores que afectan a a la calidad del software se pueden categorizar en dos amplios grupos: (1) factores que se pueden medir directamente (por ejemplo, defectos por punto de función) y (2) factores que se pueden medir sólo indirectamente (por ejemplo, facilidad de uso o de mantenimiento). En todos los casos debe aparecer la medición. Debemos comparar el software (documentos, programas, datos) con una referencia y llegar a una conclusión sobre la calidad.

PUNTO CLAVE
Es interesante anotar que los factores de calidad de McCall son tan válidos hoy como cuando fueron los primeros propuestos en los años 70. Además, es razonable indicar que los factores que afectan a la calidad del software no cambian.

McCall y sus colegas [MCC77] propusieron una útil clasificación de factores que afectan a la calidad del software. Estos factores de calidad del software, mostrados en la Figura 19.1, se concentran en tres aspectos importantes de un producto software: sus características operativas, su capacidad de cambios y su adaptabilidad a nuevos entornos.

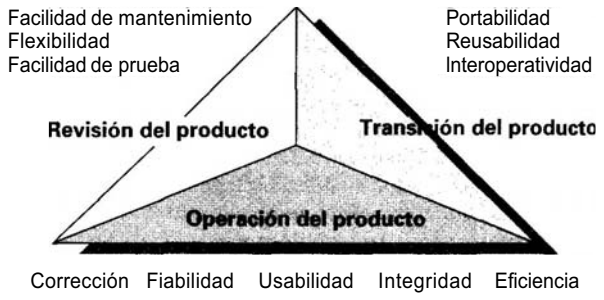


FIGURA 19.1. Factores de calidad de McCall

Refiriéndose a los factores anotados en la Figura 19.1, McCall proporciona las siguientes descripciones:

Corrección. Hasta dónde satisface un programa **su** especificación y logra los objetivos propuestos por el cliente.

Fiabilidad. Hasta dónde se puede esperar que **un** programa lleve a cabo su función con la exactitud requerida. Hay que hacer notar que se han propuesto otras definiciones de fiabilidad **más** completas (vea el Capítulo 8).

[1] Es importante resaltar que la calidad se extiende a los atributos técnicos de los modelos de análisis, de diseño y de codificación. Los modelos que presentan una alta calidad (en el sentido técnico) darán lugar a un software de una alta calidad desde el punto de vista del cliente.

Eficiencia. La cantidad de recursos informáticos y de código necesarios para que un programa realice su función.

Integridad. Hasta dónde se puede controlar el acceso al software o a **los** datos por personas no autorizadas.

Cita:
La calidad de un producto es una función de los muchos cambios del mundo por mejorar.
Tom DeMarco

Usabilidad (facilidad de manejo). El esfuerzo necesario para aprender a operar con el sistema, preparar **los** datos de entrada e interpretar las salidas (resultados) de un programa.

Facilidad de mantenimiento. El esfuerzo necesario para localizar y arreglar un error en un programa. (Esta es una definición muy limitada).

Flexibilidad. El esfuerzo necesario para modificar un programa que ya está en funcionamiento.

Facilidad de prueba. El esfuerzo necesario para probar un programa y asegurarse de que realiza correctamente su función.

Portabilidad. El esfuerzo necesario para transferir el programa de un entorno hardware/software a otro entomo diferente.

Reusabilidad (capacidad de reutilización). Hasta dónde se puede volver a emplear un programa (o partes de un programa) en otras aplicaciones, en relación al empaquetamiento y alcance de las funciones que realiza el programa.

Interoperatividad. **El** esfuerzo necesario para acoplar un sistema con otro.

Es difícil, y en algunos casos imposible, desarrollar medidas directas de los factores de calidad anteriores. Por tanto, se definen y emplean un conjunto de métricas para desarrollar expresiones para todos los factores, de acuerdo con la siguiente relación:

$$F_q = c_1 \times m_1 + c_2 \times m_2 + ... + c_n \times m_n$$

donde F_q es un factor de calidad del software, c, son coeficientes de regresión y m, son las métricas que afectan al factor de calidad. Desgraciadamente, muchas de las métricas definidas por McCall pueden medirse solamente de manera subjetiva. Las métricas pueden ir en forma de lista de comprobación que **se** emplea para «puntuar» atributos específicos del software [CAV78]. El esquema de puntuación propuesto por McCall es una escala del 0 (bajo) al 10 (alto). Se emplean las siguientes métricas en el esquema de puntuación:

Referencia cruzada
Las métricas indicadas pueden ser valoradas en las revisiones técnicas formales referenciadas en el Capítulo 8.

Facilidad de auditoría. La facilidad con la que se puede comprobar el cumplimeinto de **los** estándares.

Exactitud. La exactitud de **los** cálculos y del control.

Estandarización de comunicaciones. El grado de empleo de estándares de interfaces, protocolos y anchos de banda.

Complección. El grado con que se ha logrado la implementación total de una función.

Concisión. Lo compacto que es el programa en términos de líneas de código.

Consistencia. El empleo de un diseño uniforme y de técnicas de documentación **a** lo largo del proyecto de desarrollo del software.

Estandarizaciónde datos. El empleo de estructuras y tipos de datos estándares a lo largo del programa.

Tolerancia al error. El daño causado cuando un programa encuentra un error.

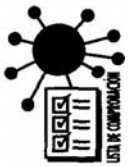
Eficiencia de ejecución. **El** rendimiento del funcionamiento de un programa.

Capacidad de expansión. El grado con que se pueden ampliar el diseño arquitectónico, de datos o procedimental.

Generalidad. La amplitud de aplicación potencial de **los** componentes del programa.

Independencia del hardware. El grado con que se desacopla el software del hardware donde opera.

Instrumentación. El grado con que el programa vigila su propio funcionamiento e identifica los errores que ocurren.



Factores de Calidad

Modularidud. La independencia funcional (Capítulo **13)** de componentes de programa.

Operatividud. La facilidad de operación de un programa.

Seguridad. La disponibilidad de mecanismos que controlan o protegen los programas y los datos.

Autodocumentación. El grado en que el código fuente proporciona documentación significativa.

Simplicidad. **El** grado de facilidad con que se puede entender un programa.

Independencia del sistema software. El grado de independencia de programa respecto a las características del lenguaje de programación no estándar, características del sistema operativo y otras restricciones del entorno.

Trazabilidad. La capacidad de seguir una representacióndel diseño o un componente real del programa hasta **los** requisitos.

Formación. **El** grado en que ayuda el software a manejar el sistema a los nuevos usuarios.

La relación entre los factores de calidad del software y las métricas de la lista anterior se muestra en la Figura 19.2. Debería decirse que el peso que se asigna a cada métrica depende de los productos y negocios locales.

19.1.2. FURPS

Los factores de calidad descritos por McCall y **sus** colegas [MCC77] representan sólo una de las muchas listas de comprobación sugeridas para la calidad del software. Hewlett-Packard [GRA87] ha desarrollado un conjunto de factores de calidad del software al que se le ha

dado el acrónimo de ***FURPS:*** funcionalidad, *facilidad de uso,* fiabilidad, rendimiento y capacidad de soporte. Los factores de calidad FURPS provienen de trabajos anteriores, definiendo los siguientes atributos para cada uno de los cinco factores principales:

- La *funcionalidad* se valora evaluando el conjunto de características y capacidades del programa, la generalidad de las funciones entregadas y la seguridad del sistema global.
- La facilidad de uso se valora considerando factores humanos (capítulo 15), la estética, la consistencia y la documentación general.
- La *fiabilidad* se evalúa midiendo la frecuencia y gravedad de los fallos, la exactitud de las salidas (resultados), el tiempo de medio de fallos (TMDF), la capacidad de recuperación de un fallo y la capacidad de predicción del programa.
- El rendimiento se mide por la velocidad de procesamiento, el tiempo de respuesta, consumo de recursos, rendimiento efectivo total y eficacia.
- La capacidad de soporte combina la capacidad de ampliar **el** programa (extensibilidad), adaptabilidad y servicios (estos tres atributos representan un término más común —mantenimiento—), así como capacidad de hacer pruebas, compatibilidad, capacidad de configuración (la capacidad de organizar y controlar elementos de la configuración del software [Capítulo9]), la facilidad de instalación de un sistema y la facilidad con que se pueden localizar los problemas.

Los factores de calidad FURPS y atributos descritos anteriormente pueden usarse para establecer métricas de la calidad para todas las actividades del proceso del software.

19.1.3. Factores de calidad ISO 9126

El estándar ISO 9126 ha sido desarrollado en un intento de identificar los atributos clave de calidad para el software. El estándar identifica seis atributos clave de calidad:

Funcionalidad. **El** grado en que el software satisface las necesidades indicadas por los siguientes subatributos: idoneidad, corrección, interoperatividad, conformidad y seguridad.

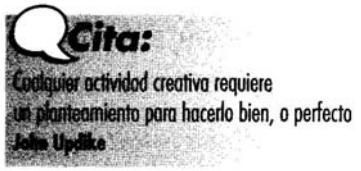
Confiabilidad. Cantidad de tiempo que el software está disponible para su uso. Está referido por los siguientes subatributos: madurez, tolerancia a fallos y facilidad de recuperación.

Usabilidad. Grado en que el software es fácil de usar. Viene reflejado por los siguientes subatributos: facilidad de comprensión, facilidad de aprendizaje y operatividad.

Eficiencia. Grado en que el software hace Óptimo el uso de los recursos del sistema. Está indicado por los siguientes subatributos: tiempo de uso y recursos utilizados.

Facilidad de mantenimiento. La facilidad con que una modificación puede ser realizada. Está indicada por los siguientes subatributos: facilidad de análisis, facilidad de cambio, estabilidad **y** facilidad de prueba.

Portabilidad. La facilidad con que el software puede ser llevado de un entorno a otro. Está referido por los siguientes subatributos: facilidad de instalación, facilidad de ajuste, facilidad de adaptación al cambio.



Del mismo modo que los factores de calidad estudiados en las secciones 19.1.1. y 19.1.2., los factores ISO 9126 no necesariamente son utilizados para medidas directas. En cualquier caso, facilitan una valiosa base para medidas indirectas **y** una excelente lista para determinar la calidad de un sistema.

Métrica de la calidad del software / Factor de calidad	Correción	Fiabialidad	Eficiencia	Integridad	Mantenimiento	Flexibilidad	Capacidad de pruebas	Portabilidad	Reusabilidad	Interoperativilidad	Usabilidad
Facilidad de auditoría				X			X				
Exactitud		X									
Estandarización de comunicaciones										X	
Compleción	X										
Complejidad		X				X	X				
Concisión			X		X	X					
Consistencia	X	X			X	X					
Estandarización de datos										X	
Tolerancia a errores		X									
Eficiencia de ejecución			X								
Capacidad de expansión						X					
Generalidad						X		X	X	X	
Independiencia del hardware								X	X		
Instrumentación				X	X		X				
Modularidad		X			X	X	X	X	X	X	
Operatividad			X								X
Seguridad				X							
Autodocumentación					X	X	X	X	X		
Simplicidad		X			X	X	X				
Independencia del sistema								X	X		
Trazabilidad	X										
Facilidad de formación											X

FIGURA 19.2. Factores y métricas de calidad.

19.1.4. La transición a una visión cuantitativa

En las secciones precedentes se estudiaron un conjunto de factores cualitativos para la «medición» de la calidad del software. Intentamos desarrollar medidas exactas de la calidad del software frustradas a veces por la naturaleza subjetiva de la actividad. Cavano y McCall [CAV78] estudian esta situación:

> La determinación de la calidad es un factor clave en los acontecimientos diarios: concursos de cata de vinos, acontecimientos deportivos (por ejemplo, la gimnasia), concursos de talento, etc. En estas situaciones, la calidad se juzga de la manera más fundamental y directa: comparación de objetos unos al

lado de los otros bajo condiciones idénticas y con conceptos predeterminados. El vino puede ser juzgado de acuerdo con su claridad, color, bouquet, sabor, etc. Sin embargo, este tipo de juicio es muy subjetivo; para que tenga algo de valor, debe hacerse por un experto.

La subjetividad y la especialización también influyen en la determinación de la calidad del software. Para resolver este problema, se necesita una definición de calidad del software más exacta, así como una manera de obtener medidas cuantitativas de la calidad del software para hacer un análisis objetivo.... Como no existe el conocimiento absoluto, no deberíamos esperar poder medir la calidad del software exactamente, ya que cada medición es parcialmente imperfecta. Jacob Bronkowski describió esta paradoja del conocimiento de la siguiente manera: «Año tras año ingeniamos instrumentos más exactos con los que observar la naturaleza con más exactitud. Y cuando miramos las observaciones estamos desconcertados de ver que todavía son confusas, y tenemos la sensación de que son tan inciertas como siempre.»

En las siguientes secciones examinamos un conjunto de métricas del software que pueden aplicarse a la valoración cuantitativa de la calidad del software. En todos los casos, las métricas representan medidas indirectas; es decir, realmente nunca medimos la calidad sino alguna manifestación de la calidad. El factor que lo complica es la relación exacta entre la variable que se mide y la calidad del software.

19.2 UNA ESTRUCTURA PARA LAS MÉTRICAS TÉCNICAS DEL SOFTWARE

Como dijimos en la introducción de este capítulo, la medición asigna números o símbolos a atributos de entidades en el mundo real. Para conseguirlo se necesita un modelo de medición que comprenda un conjunto consistente de reglas. Aunque la teoría de la medición (por ejemplo, [KYB84]) y su aplicación al software (por ejemplo, [DEM81], [BRI96]) son temas que están más allá del alcance de este libro, merece la pena establecer una estructura fundamental y un conjunto de principios básicos para la medición de métricas técnicas para el software.

Cita:

Lo mismo que las medidas de temperatura comienzan con un índice de referencia y evolucionan por sofisticadas escalas, herramientas y técnicas, de la misma forma están madurando las medidas del software.

Shari Pfleeger

19.2.1. El reto de las métricas técnicas

Durante las pasadas tres décadas, muchos investigadores han intentado desarrollar una sola métrica que proporcione una medida completa de la complejidad del software. Fenton [FEN94] caracteriza esta investigación como una búsqueda del «imposible santo grial». Aunque se han propuesto docenas de medidas de complejidad [ZUS90], cada una tiene un punto de vista diferente de lo que es la complejidad y qué atributos de un sistema llevan a la complejidad.

Referencia Web

Voluminosa información sobre métricas técnicas han sido recopiladas por Horst Zuse: **irb.cs.tu-berlin.de/~zuse/**

Pero existe la necesidad de medir y controlar la complejidad del software. Y si es difícil de obtener un solo valor de esta «métrica de calidad», si debería ser posible desarrollar medidas de diferentes atributos internos del programa (por ejemplo, modularidad efectiva, independencia funcional y otros atributos tratados desde el Capítulo 13 al Capítulo 16). Estas medidas y las métricas obtenidas de ellas pueden usarse como indicadores independientes de la calidad de los modelos de análisis y diseño. Pero también surgen problemas aquí. Fenton [FEN94] lo advierte cuando dice:

> El peligro de intentar encontrar medidas que caractericen tantos atributos diferentes es que, inevitablemente, las medidas tienen que satisfacer objetivos incompatibles. Esto es contrario a la teoría de representación de la medición.

Aunque la declaración de Fenton es correcta, mucha gente argumenta que la medición técnica llevada a cabo durante las primeras fases del proceso de software les proporciona a los desarrolladores de software un consistente y objetivo mecanismo para valorar la calidad.

Conviene preguntarse, no obstante, qué validez tienen las métricas técnicas. Es decir, ¿cómo están de próximas las métricas técnicas y la fiabilidad y la calidad a largo plazo de un sistema basado en computadora? Fenton [FEN91] trata esta cuestión de la siguiente manera:

> A pesar de las intuitivas conexiones entre la estructura interna de los productos software (métricas técnicas) y su producto externo, y los atributos del proceso, ha habido, de hecho, muy pocos intentos científicos para establecer relaciones específicas. Hay varias razones para ello; la que se menciona más a menudo es la imposibilidad de llevar a cabo experimentos.

Todos los desafíos mencionados anteriormente son motivo de precaución, pero no es motivo de desprecio de las métricas técnicas[2]. La medición es esencial si se desea conseguir calidad.

[2] Se ha producido una gran cantidad de literatura sobre las métricas del software (por ejemplo, vea [FEN94], [ROC94], [ZUS97] para obtener unas extensas bibliografías) y es común la crítica de métricas específicas (incluyendo algunas de las presentadas en este capítulo). Sin embargo, muchas de las críticas se concentran en aspectos esotéricos y pierden el objetivo primario de la medición en el mundo real: ayudar al ingeniero a establecer una manera sistemática y objetiva de conseguir una visión interna de su trabajo y mejorar la calidad del producto como resultado.

19.2.2. Principios de medición

Antes de introducir una serie de métricas técnicas que (1) ayuden a la evaluación de los modelos de análisis y diseño, (2) proporcionen una indicación de la complejidad de los diseños procedimentales y del código fuente, y (3) ayuden en el diseño de pruebas más efectivas, es importante entender los principios básicos de la medición. Roche [ROC94] sugiere un proceso de medición que se puede caracterizar por cinco actividades:

- *formulación:* la obtención de medidas y métricas del software apropiadas para la representación del software en cuestión.
- *colección:* el mecanismo empleado para acumular datos necesarios para obtener las métricas formuladas.
- *análisis:* el cálculo de las métricas y la aplicación de herramientas matemáticas.

¿Cuáles son los pasos para un proceso de medición correcto?

- *interpretación:* la evaluación de los resultados de las métricas en un esfuerzo por conseguir una visión interna de la calidad de la representación.
- *realimentación (feedback):* recomendaciones obtenidas de la interpretación de métricas técnicas transmitidas al equipo que construye el software.

Los principios que se pueden asociar con la formulación de las métricas técnicas son los siguientes [ROC94]:

- Los objetivos de la medición deberían establecerse antes de empezar la recogida de datos.
- Todas las técnicas sobre métricas deberían definirse sin ambigüedades.

¿Qué reglas debemos observar cuando establecemos medidas técnicas?

- Las métricas deberían obtenerse basándose en una teoría válida para el dominio de aplicación (por ejemplo, las métricas para el diseño han de dibujarse sobre conceptos y principios básicos de diseño y deberían intentar proporcionar una indicación de la presencia de un atributo que se considera beneficioso).
- Hay que hacer las métricas a medida para acomodar mejor los productos y procesos específicos [BAS84].

Aunque la formulación es un punto de arranque crítico, la recogida y análisis son las actividades que dirigen el proceso de medición. Roche [ROC94] sugiere los siguientes principios para estas actividades:

- Siempre que sea posible, la recogida de datos y el análisis debe automatizarse.

Por encima de todo, intento obtener medidos técnicas simples. No te obsesiones por lo métrica «perfecta» porque no existe.

- Se deberían aplicar técnicas estadísticas válidas para establecer las relaciones entre los atributos internos del producto y las características externas de la calidad (por ejemplo, ¿está correlacionado el nivel de complejidad arquitectónico con el número de defectos descubiertos en la producción?).
- Se deberían establecer una directrices de interpretación y recomendaciones para todas las métricas.

Además de los principios apuntados anteriormente, el éxito de una actividad de métrica está ligada al soporte de gestión. Se deben considerar los fondos, la formación y la promoción si se quiere establecer y mantener un programa de medición técnica.

19.2.3. Características fundamentales de las métricas del software

Se han propuesto cientos de métricas para el software, pero no todas proporcionan un soporte práctico para el desarrollador de software. Algunas demandan mediciones que son demasiado complejas, otras son tan esotéricas que pocos profesionales tienen la esperanza de entenderlas y otras violan las nociones básicas intuitivas de lo que realmente es el software de alta calidad.

Ejiogu [EJI91] define un conjunto de atributos que deberían acompañar a las métricas efectivas del software. La métrica obtenida y las medidas que conducen a ello deberían ser:

- *simples y fáciles de calcular.* Debería ser relativamente fácil aprender a obtener la métrica y su cálculo no debería demandar un esfuerzo o cantidad de tiempo inusuales.

¿Cómo podemos valorar la calidad de una métrica del software propuesta?

- *empírica e intuitivamente persuasivas.* La métrica debería satisfacer las nociones intuitivas del ingeniero sobre el atributo del producto en cuestión (por ejemplo, una métrica que mide la cohesión de un módulo debería aumentar su valor a medida que crece el nivel de cohesión).
- *consistentes* y *objetivas.* La métrica debería siempre producir resultados sin ambigüedad. Un tercer equipo debería ser capaz de obtener el mismo valor de métrica usando la misma información del software.
- *consistentes en el empleo de unidades y tamaños.* El cálculo matemático de la métrica debería emplear medidas que no conduzcan a extrañas combinaciones de unidades. Por ejemplo, multiplicando el número de personas de un equipo por las variables del lenguaje de programación en el programa resulta una sospechosa mezcla de unidades que no son intuitivamente persuasivas.
- *independientes del lenguaje de programación.* Las métricas deberían basarse en el modelo de análisis, modelo de diseño o en la propia estructura del

programa. No deberían depender de los caprichos de la sintaxis o semántica del lenguaje de programación.

- *un eficaz mecanismo para la realimentación de calidad.* La métrica debería proporcionar, al desarrollador de software, información que le lleve a un producto final de mayor calidad.

La experiencia indica que una métrica técnica se usa únicamente si es intuitiva y fácil de calcular. Si se requiere docenas de «contadores» y se han de utilizar complejos cálculos, la métrica no será ampliamente utilizado.

Aunque la mayoría de las métricas de software satisfacen las características anteriores, algunas de la métricas comúnmente empleadas dejan de cumplir una o dos. Un ejemplo es el punto de función (tratado en el Capítulo **4** y en este capítulo). Se puede argumentar[3] que el atributo consistente y objetivo falla porque un equipo ajeno independiente puede no ser capaz de obtener el mismo valor del punto de función que otro equipo que use la misma información del software. ¿Deberíamos entonces rechazar la medida PF? La respuesta es «¡por supuesto que no!». El PF proporciona una útil visión interna y por tanto proporciona un valor claro, incluso si no satisface un atributo perfectamente.

19.3 MÉTRICAS DEL MODELO DE ANÁLISIS

El trabajo técnico en la ingeniería del software empieza con la creación del modelo de análisis. En esta fase se obtienen los requisitos y se establece el fundamento para el diseño. Por tanto, son deseables las métricas técnicas que proporcionan una visión interna a la calidad del modelo de análisis.

Referencia cruzada
Los modelos de datos, funciones y comportamientos son tratados en los Capítulos 11 y 12.

Aunque han aparecido en la literatura relativamente pocas métricas de análisis y especificación, es posible adaptar métricas obtenidas para la aplicación de un proyecto (Capítulo **4**) para emplearlas en este contexto. Estas métricas examinan el modelo de análisis con la intención de predecir el «tamaño» del sistema resultante. Es probable que el tamaño y la complejidad del diseño estén directamente relacionadas.

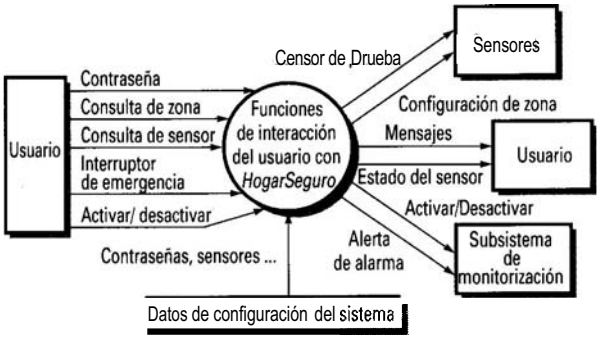


FIGURA 19.3. Parte del modelo de análisis del software de *Hogar Seguro.*

19.3.1. Métricas basadas en la función

La métrica de punto de función (PF) (Capítulo **4**) se puede usar como medio para predecir el tamaño de un sistema que se va a obtener de un modelo de análisis. Para ilustrar el empleo de la métrica de PF en este contexto, consideramos una sencilla representación del modelo de análisis mostrada en la Figura 19.3. En la figura se representa un diagrama de flujo de datos (Capítulo 12) de una función del sistema *HogarSeguro*[4]. La función gestiona la interacción con el usuario, aceptando una contraseña de usuario para activar/desactivar el sistema y permitiendo consultas sobre el estado de las zonas de seguridad y varios sensores de seguridad. La función muestra una serie de mensajes de petición y envía señales apropiadas de control a varios componentes del sistema de seguridad.

Para ser útil para el trabajo técnica, las medidas técnicas que apoyan la toma de decisión (ej: errores encontrados durante la prueba de unidad) deben ser agrupadas y normalizadas utilizando la métrica PF.

El diagrama de flujo de datos se evalúa para determinar las medidas clave necesarias para el cálculo de la métrica de punto de función (Capítulo **4**):

- número de entradas del usuario
- número de salidas del usuario
- número de consultas de usuario
- número de archivos
- número de interfaces externas

Tres entradas del usuario: **contraseña, interruptor de emergencia** y **activar/desactivar** aparecen en la figura junto con dos consultas: **consulta de zona** y **consulta de sensor.** Se muestra un archivo **(archivo de**

[3] Fíjese que se puede hacer un contra argumento igualmente vigoroso. Tal es la naturaleza de las métricas del software

[4] *HogarSeguro* es un sistema de seguridad para el hogar que se ha usado como aplicación de ejemplo en capítulos anteriores

configuración del sistema). También están presentes dos salidas de usuarios **(mensajes** y **estados del sensor)** y cuatro interfaces externas **(sensor de prueba, configuración de zona, activar/desactivar** y **alerta de alarma).** Estos datos, junto con la complejidad apropiada, se muestran en la Figura 19.4.

Parámetro de medición	Cuenta		Factor de ponderación: Simple	Media	Compleja		
Número de entradas del usuario	3	X	3	4	6	=	9
Número de salidas del usuario	2	X	4	5	7	=	8
Número de consultas del usuario	2	X	3	4	6	=	6
Número de archivos	1	X	7	10	15	=	7
Número de interfaces externas	4	X	5	7	10	=	20
Cuenta total						→	50

FIGURA 19.4. Cálculo de puntos de función para una función de Hogar *Seguro.*

La *cuenta total* mostrada en la Figura 19.4 debe ajustarse usando la ecuación (4.1):

$$PF = \text{cuenta-total} \times (0{,}65 + 0{,}01 \times \sum[Fi])$$

Donde cuenta-total es la suma de todas las entradas PF obtenidas en la Figura 19.3 y F_i (i=1 a 14) son «los valores de ajuste de complejidad». Para el propósito de este ejemplo, asumimos que $\sum[Fi]$ es 46 (un producto moderadamente complejo). Por tanto:

$$PF = 50 \times [0{,}65 + (0{,}01 \times 46)] = 56$$

Referencia Web

Una introducción útil al PF ha sido elaborada por Capers Jones y puede ser localizada en:
www.spr.com/ library/ Ofuncmet.htm

Basándose en el valor previsto de PF obtenido del modelo de análisis, el equipo del proyecto puede estimar el tamaño global de implementación de las funciones de interacción de *HogarSeguro.* Asuma que los datos de los que se disponen indican que un PF supone 60 líneas de código (se va a usar un lenguaje orientado a objetos) y que en un esfuerzo de un mes-persona se producen 12 PF. Estos datos históricos proporcionan al gestor del proyecto una importante información de planificación basada en el modelo de análisis en lugar de en estimaciones preliminares.

Considerar que de los proyectos anteriores se han encontrado una media de **3** errores por punto de función durante las revisiones de análisis y diseño, y 4 errores por punto de función durante las pruebas unitaria y de integración. Estos datos pueden ayudar a valorar a los ingenieros del software la completitud de sus revisiones y las actividades de prueba.

19.3.2. La Métrica *bang*

Al igual que la métrica de punto de función, la *métrica bang* puede emplearse para desarrollar una indicación del tamaño del software a implementar como consecuencia del modelo de análisis.

Desarrollada por DeMarco [DEM82], la métrica *bang* es «una indicación independiente de la implementación del tamaño del sistema». Para calcular la métrica *bang,* el desarrollador de software debe evaluar primero un conjunto de primitivas (elementos del modelo de análisis que no se subdividen más en el nivel de análisis). Las primitivas [DEM82] se determinan evaluando el modelo de análisis y desarrollando cuentas para los siguientes elementos?

Primitivas funcionales (PFu). Transformaciones (burbujas) que aparecen en el nivel inferior de un diagrama de flujo de datos (Capítulo 12).

Elementos de datos (ED). Los atributos de un objeto **de** datos, los elementos de datos son datos no compuestos y aparecen en el diccionario de datos.

Objetos (OB). Objetos de datos tal y como se describen **en** el Capítulo 11.

Relaciones (RE). Las conexiones entre objetos de datos **tal** y como se describen en el Capítulo 12.

Estados (ES). El número de estados observables **por** el usua**no** en el diagrama de transición de estados (Capítulo **12).**

Transiciones (TR). El número de transiciones de estado en el diagrama de transición de estados (Capítulo 12).

Cita:
Antes que reflexionemos sobre una «nueva métrica» debemos aplicar ... podemos preguntarnos nosotros mismos sobre los puntos básicos, «Qué pretendemos con las métricas.
Michael Mah y Larry Putnam

Además de las seis primitivas apuntadas arriba, se determinan las cuentas adicionales para:

Primitivas modificadas de función manual (PMFu). Funciones que caen fuera del límite del sistema y que deben modificarse para acomodarse al nuevo sistema.

[5] El acrónimo apuntado entre paréntesis a continuación de la primitiva se emplea para denotar **la** *cuenta* de la primitiva particular; por ejemplo, PFu indica el número de primitivas funcionales presente en un modelo de análisis.

Elementos de datos de entrada (EDE). Aquellos elementos de datos que se introducen en el sistema.

Elementos de datos de salida (EDS). Aquellos elementos de datos que se sacan del sistema.

Elementos de datos retenidos (EDR). Aquellos elementos de datos que son retenidos (almacenados) por el sistema.

Muestras *(tokens)* de datos (TC_i). Las muestras de datos (elementos de datos que no se subdividen dentro de una primitiva funcional) que existen en el límite de la i-ésima primitiva funcional (evaluada para cada primitiva).

Conexiones de relación (RE_i). Las relaciones que conectan el i-ésimo objeto en el modelo de datos con otros objetos.

DeMarco [DEM82] sugiere que la mayoría del software se puede asignar a uno de los dos dominios siguientes, ***dominio de función o dominio de datos,*** dependiendo de la relación RE/PFu. Las aplicaciones de dominio de función (encontradas comúnmente en aplicaciones de ingeniería y científicas) hacen hincapié en la transformación de datos y no poseen generalmente estructuras de datos complejas. Las aplicaciones de dominio de datos (encontradas comúnmente en aplicaciones de sistemas de información) tienden a tener modelos de datos complejos.

RE/PFu < 0,7 implica una aplicación de dominio de función

0,8 < RE/PFu < 1,4 indica una aplicación híbrida

RE/PFu > 1,5 implica una aplicación de dominio de datos

Como diferentes modelos de análisis harán una partición del modelo con mayor o menor grado de refinamiento. DeMarco sugiere que se emplee una cuenta media de muestra ***(token)*** por primitiva

$$TC_{avg} = \Sigma TC_i / PFu$$

para controlar la uniformidad de la partición a través de muchos diferentes modelos dentro del dominio de una aplicación.

Para calcular la métrica *bang* para aplicaciones de dominio de función, se emplea el siguiente algoritmo:

```
Asignar a bang un valor inicial = ∅;
Mientras queden primitivas funcionales por evaluar
    Calcular cuenta-token alrededor del límite de la
    primitiva i;
    Calcular el incremento PFu corregido (IPFuC)
    Asignar la primitiva a una clase
    Evaluar la clase y anotar el peso valorado
    Multiplicar IPFuC por el peso valorado
    bang = bang + ponderación IPFuC;
FinMientras
```

La cuenta-token se calcula determinando cuántos símbolos léxicos ***(tokens)*** diferentes son «visibles» [DEM82] dentro de la primitiva. Es posible que el número de símbolos léxicos ***(tokens)*** y el número de elementos de datos sea diferente, si los elementos de datos pueden moverse desde la entrada a la salida sin ninguna transformación interna. La IPFuC corregida se determina de una tabla publicada por DeMarco. A continuación, se presenta una versión muy abreviada:

Tci	IPFuC
2	1,0
5	5,8
10	16,6
15	29,3
20	43,2

La ponderación valorada apuntada en el algoritmo anterior se calcula de dieciséis clases diferentes de primitivas funcionales definidas por DeMarco. Se asigna una ponderación que va de 0,6 (encaminamiento simple de datos) a 2,5 (funciones de gestión de datos) dependiendo de la clase de la primitiva.

Para aplicaciones de dominio de datos, se calcula la métrica ***bang*** mediante el siguiente algoritmo:

```
Asignar a bang el valor inicial = ∅;
Mientras queden objetos por evaluar en el modelo de
datos
    Calcular la cuenta de relaciones' del objeto i
    Calcular el incremento de OB corregido (IOBC);
    bang = bang + IOBC;
FinMientras
```

El IOBC corregido se determina también de una tabla publicada por DeMarco. A continuación se muestra una versión abreviada:

REi	IOBC
1	1,0
3	4,0
6	9,0

Una vez que se ha calculado la métrica ***bang,*** se puede emplear el historial anterior para asociarla con el esfuerzo y el tamaño. DeMarco sugiere que las organizaciones se construyan sus propias versiones de tablas IPFuC e IOBC para calibrar la información de proyectos completos de software.

19.3.3. Métricas de la calidad de la especificación

Davis y sus colegas [DAV93] proponen una lista de características que pueden emplearse para valorar la calidad del modelo de análisis y la correspondiente especificación de requisitos: especificidad (ausencia de ambigüedad), compleción, corrección, comprensión, capacidad de verificación, consistencia interna y externa, capacidad de logro, concisión, trazabilidad, capacidad de modificación, exactitud y capacidad de reutilización. Además, los autores apuntan que las especificaciones de alta calidad deben estar almacenadas electrónicamente, ser ejecutables o al menos interpretables, anotadas por importancia y estabilidad relativas, con su versión correspondiente, organizadas, con referencias cruzadas y especificadas al nivel correcto de detalle.

Aunque muchas de las características anteriores parecen ser de naturaleza cualitativa, Davis [DAV93] sugie-

re que todas puedan representarse usando una o más métricas[6]. Por ejemplo, asumimos que hay n_r requisitos en una especificación, tal como

$$n_r = n_f + n_{nf}$$

donde n_f es el número de requisitos funcionales y n_{nf} es el número de requisitos no funcionales (por ejemplo, rendimiento).

PUNTO CLAVE

Para medir las características de la especificación, es necesario conseguir profundizar cuantitativamente en la especificidad y en la completitud.

Para determinar **la** *especificidad* (ausencia de ambigüedad) de los requisitos. Davis sugiere una métrica basada en la consistencia de la interpretación de los revisores para cada requisito:

$$Q_1 = n_{ui} / n_r$$

donde n_{ui} es el número de requisitos para los que todos los revisores tuvieron interpretaciones idénticas. Cuanto más cerca de 1 esté el valor de Q, menor será la ambigüedad de la especificación.

La *compleción* de los requisitos funcionales pueden determinarse calculando la relación

$$Q_2 = u_n / (n_i \times n_s)$$

donde u, es el número de requisitos únicos de función, n_i es el número de entradas (estímulos) definidos o implicados por la especificación y n_s es el número de estados especificados. La relación Q, mide el porcentaje de funciones necesarias que se han especificado para un sistema. Sin embargo, no trata los requisitos no funcionales. Para incorporarlos a una métrica global completa, debemos considerar el grado de validación de los requisitos.

$$Q_3 = n_c / (n_c + n_{nv})$$

donde n_c es el número de requisitos que se han validado como correctos y n_{nv} el número de requisitos que no se han validado todavía.

19.4 MÉTRICAS DEL MODELO DE DISEÑO

Es inconcebible que el diseño de un nuevo avión, un nuevo chip de computadora o un nuevo edificio de oficinas se realizara sin definir las medidas del diseño, sin determinar las métricas para varios aspectos de la calidad del diseño y usarlas para guiar la evolución del diseño. Y sin embargo, el diseño de sistemas complejos basados en computadora a menudo consigue proseguir sin virtualmente ninguna medición. La ironía es que las métricas de diseño para el software están disponibles, pero la gran mayoría de los desarrolladores de software continúan sin saber que existen.

Cita:
La medida que es apreciable,
y que no es apreciable, se hace apreciable.
Galileo

Las métricas de diseño para el software, como otras métricas del software, no son perfectas. Continúa el debate sobre la eficacia y cómo deberían aplicarse. Muchos expertos argumentan que se necesita más experimentación hasta que se puedan emplear las métricas de diseño. Y sin embargo, el diseño sin medición es una alternativa inaceptable.

En las siguientes secciones examinamos algunas de las métricas de diseño más comunes para el software de computadora. Aunque ninguna es perfecta, todas pueden proporcionar al diseñador una mejor visión interna y ayudar a que el diseño evolucione a un nivel superior de calidad.

19.4.1. Métricas del diseño arquitectónico

Las métricas de diseño de alto nivel se concentran en las características de la arquitectura del programa (Capítulo 14) con especial énfasis en la estructura arquitectónica y en la eficiencia de los módulos. Estas métricas son de caja negra en el sentido que no requieren ningún conocimiento del trabajo interno de un módulo en particular del sistema.

Card y Glass [CAR90] definen tres medidas de la complejidad del diseño del software: complejidad estructural, complejidad de datos y complejidad del sistema.

La *complejidad estructural, S(i)*, de un módulo *i* se define de la siguiente manera:

$$S(i) = f^2_{out}(i) \qquad (19\text{-}1)$$

donde $f_{out}(i)$ es la expansión[7] del módulo i.

[6] Un estudio completo de las métricas de calidad de especificación está más allá del alcance de este capítulo. Vea [DAV93] para más detalles.

[7] Como se dijo en el estudio introducido en el Capítulo 13, la expansión (f_{out}) indica el número de módulos inmediatamente subordinados al módulo i, es decir, el número de módulos que son invocados directamente por el módulo i.

PUNTO CLAVE

Las métricas pueden profundizar en la estructura, en los datos, y en la complejidad del sistema asociado con el diseño arquitectónico.

La *complejidad de datos, D(i),* proporciona una indicación de la complejidad en la interfaz interna de un módulo *i* y se define como:

$$D(i)=v(i) / [f_{out}(i) + 1] \qquad (19.2)$$

donde *v(i)* es el número de variables de entrada y salida que entran y salen del módulo i.

Finalmente, la *complejidad del sistema, C(i),* se define como la suma de las complejidades estructural y de datos, y se define como:

$$C(i)= S(i) + D(i) \qquad (19.3)$$

A medida que crecen los valores de complejidad, la complejidad arquitéctónica o global del sistema también aumenta. Esto lleva a una mayor probabilidad de que aumente el esfuerzo necesario para la integración y las pruebas.

¿Hay un camino para valorar la Complejidad de ciertos modelos arquitectónicos?

Una métrica de diseño arquitectónico de alto nivel, propuesta por Henry y Kafura [HEN81], también emplea la expansión y la concentración. Los autores definen una métrica de complejidad de la forma:

$$\text{MHK} = \text{longitud}(i) \times [f_{in}(i) + f_{out}(i)]^2 \qquad (19.4)$$

donde la *longitud(i)* es el número de sentencias en lenguaje de programación en el módulo i y f_{in}(i) es la concentración del módulo i. Henry y Kafura amplían la definición de concentración y expansión presentadas en este libro para incluir no sólo el número de conexiones de control del módulo (llamadas al módulo), sino también el número de estructuras de datos del que el módulo i recoge (concentración) o actualiza (expansión) datos. Para calcular el MHK durante el diseño puede emplearse el diseño procedimental para estimar el número de sentencias del lenguaje de programación del módulo *i*. Como en las métricas de Card y Glass mencionadas anteriormente, un aumento en la métrica de Henry-Kafura conduce a una mayor probabilidad de que también aumente el esfuerzo de integración y pruebas del módulo.

Fenton [FEN91] sugiere varias métricas de morfología simples (por ejemplo, forma) que permiten comparar diferentes arquitecturas de programa mediante un conjunto de dimensiones directas. En la Figura 19.5, se pueden definir las siguientes métricas:

tamaño = n + a 19.5

donde n es el número de nodos (módulos) y a es el número de arcos (líneas de control). Para la arquitectura mostrada en la Figura 19.5,

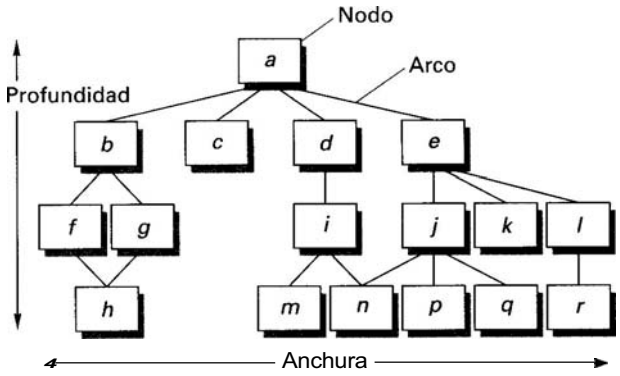


FIGURA 19.5. Métricas de morfología

tamaño = *17* + 18 = *35*

profundidad= el camino más largo desde el nodo raíz (parte más alta) a un nodo hoja. Para la arquitectura mostrada en la Figura 19.5, la profundidad = **4.**

anchura = máximo número de nodos de cualquier nivel de la arquitectura. Para la arquitectura mostrada en la Figura 19.5, la anchura = 6.

Relación arco-a-nodo, r = a / n.

que mide la densidad de conectividad de la arquitectura y puede proporcionar una sencilla indicación del acoplamiento de la arquitectura. Para la arquitectura mostrada en la Figura 19.5, r = 18/ 17 = 1,06.

Cita:

La medición puede ser vista como un rodeo. Este rodeo es necesario porque los humanos muchas veces no estamos capacitados para tomar decisiones con claridad y objetividad [sin un soporte cuantitativo].

Horst Zuse

19.4.2. Métricas de diseño a nivel de componentes

Las métricas de diseño a nivel de componentes se concentran en las características internas de los componentes del software e incluyen medidas de las «3Cs» —la cohesión, acoplamiento y complejidad del módulo—. Estas tres medidas pueden ayudar al desarrollador de software a juzgar la calidad de un diseño a nivel de los componentes.

Las métricas presentadas en esta sección son de caja blanca en el sentido de que requieren conocimiento del trabajo interno del módulo en cuestión. Las métricas de diseño de los componentes se pueden aplicar una vez que se ha desarrollado un diseño procedimental. También se pueden retrasar hasta tener disponible el código fuente.

PUNTO CLAVE

Es posible calcular medidos de la independencia funcional, acoplamiento y cohesión de un componente y usarlos paro valorar la calidad y el diseño.

Métricas de cohesión. Bieman y Ott [BIE94] definen una colección de métricas que proporcionan una indicación de la cohesión (Capítulo **13)** de un módulo. Las métricas se definen con cinco conceptos y medidas:

> ***Porción de datos***. Dicho simplemente,una porción de datos es una marcha atrás a través de un módulo que busca valores de datos que afectan a la localización de módulo en el que empezó la marcha atrás. Debería resaltarse que se pueden definir tanto porciones de programas (que se centran en enunciados y condiciones) como porciones de datos.
>
> ***Muestras (tokens) de datos.*** Las variables definidas para un módulo pueden definirse como muestras de datos para el módulo.
>
> ***Señales de unión.*** El conjunto de muestras de datos que se encuentran en una o más porciones de datos.
>
> ***Señales de superunión.*** La muestras de datos comunes a todas las porciones de datos de un módulo.
>
> ***Pegajosidad***. La pegajosidad relativa de una muestra de unión es directamente proporcional al número de porciones de datos que liga.

Bieman y Ott desarrollan métricas para *cohesiones funcionalesfuertes* (CFF), *cohesionesfuncionales débiles* (CFD) y *pegajosidad* (el grado relativo con el que las señales de unión ligan juntas porciones de datos). Estas métricas se pueden interpretar de la siguiente manera [BIE94]:

> Todas estas métricas de cohesión tienen valores que van de 0 a 1. Tienen un valor de 0 cuando un procedimiento tiene más de una salida y no muestra ningún atributo de cohesión indicado por una métrica particular. Un procedimiento sin muestras de superunión, sin muestras comunes a todas las porciones de datos, no tiene una cohesión funcional fuerte (no hay señales de datos que contribuyan a todas las salidas). Un procedimiento sin muestras de unión, es decir, sin muestras comunes a más de una porción de datos (en procedimientos con más de una porción de datos), no muestra una cohesión funcional débil y ninguna adhesividad (no hay señales de datos que contribuyan a más de una salida).

La cohesión funcional fuerte y la pegajosidad se obtienen cuando las métricas de Bieman y Ott toman un valor máximo de **1.**

Se deja un estudio más detallado de las métricas de Bieman y Ott para que sean revisadas sus fuentes [BIE94]. Sin embargo, para ilustrar el carácter de estas métricas, considere la métrica para la cohesión funcional fuerte:

$$CFF(i) = SU\ (\ SA(i))\ /\ muestra(i) \qquad (19.6)$$

donde SU (SA(*i*)) denota muestra de superunión (el conjunto de señales de datos que se encuentran en todas las porciones de datos de un módulo *i)*. Como la relación de muestras de superunión con respecto al número total de muestras en un módulo *i* aumenta hasta un valor máximo de 1, la cohesión funcional del módulo también aumenta.

Métricas de acoplamiento. El acoplamiento de módulo proporciona una indicación de la «conectividad» de un módulo con otros módulos, datos globales y el entorno exterior. En el Capítulo **13,** el acoplamiento se estudió en términos cualitativos.

Dhama [DHA95] ha propuesto una métrica para el acoplamiento del módulo que combina el acoplamiento de flujo de datos y de control, acoplamiento global y acoplamiento de entorno. Las medidas necesarias para calcular el acoplamiento de módulo se definen en términos de cada uno de los tres tipos de acoplamiento apuntados anteriormente.

Para el acoplamiento de flujo de datos y de control:

d_i = número de parámetros de datos de entrada

c_i = número de parámetros de control de entrada

d_o = número de parámetros de datos de salida

c_o = número de parámetros de control de salida

Referencia Web

El documento, «Sistema de métricas del software para el acoplamiento de módulos» podéis bajarlo de **www.isse.gmu.edu/faculty/ofut/rsrch/abstracts/mj-coupling.html**

Para el acoplamiento global:

g_d = número de variables globales usadas como datos

g_c = número de variables globales usadas como control

Para el acoplamiento de entorno:

w = número de módulos llamados (expansión)

r = número de módulos que llaman al módulo en cuestión (concentración)

Usando estas medidas, se define un indicador de acoplamiento de módulo, m_c, de la siguiente manera:

$$m_c = k / M$$

donde **k=l** es una constante de proporcionalidad'.

$$M = d_i + a \times c_i + d_o + b \times c_o + g_d + c \times g_c + w + r$$

donde $a = b = c = 2$.

Cuanto mayor es el valor de m_c, menor es el acoplamiento de módulo. Por ejemplo, si un módulo tiene un solo parámetro de entrada y salida de datos, no accede a datos globales y es llamado por un solo módulo:

$$m_c = 1 / (\ 1 + 0 + \ 1 + 0 + 0 + 0 + 1 + 0\ ) = 1 / 3 = 0{,}33.$$

Deberíamos esperar que un módulo como éste presentara un acoplamiento bajo. De ahí que, un valor de $m_c = 0{,}33$

[8] El autor [DHA95] advierte que los valores de k y a, b y c (tratados en la siguiente ecuación) pueden ajustarse a medida que se van verificando experimentalmente.

implica un acoplamiento bajo. Por el contrario, si un módulo tiene 5 parámetros de salida y 5 parámetros de entrada de datos, un número igual de parámetros de control, accede a 10 elementos de datos globales y tiene una concentración de 3 y una expansión de 4,

$$m_c = 1/(5 + 2 \times 5 + 5 + 2 \times 5 + 10 + 0 + 3 + 4) = 0{,}02$$

el acoplamiento implicado es grande.

Para que aumente la métrica de acoplamiento a medida que aumenta el grado de acoplamiento, se puede definir una métrica de acoplamiento revisada como:

$$C = 1 - m_c$$

donde el grado de acoplamiento no aumenta linealmente entre un valor mínimo en el rango de 0,66 hasta un máximo que se aproxima a 1,0.

Métricas de complejidad. Se pueden calcular una variedad de métricas del software para determinar la complejidad del flujo de control del programa. Muchas de éstas se basan en una representación denominada grafo de flujo. Tal y como se dijo en el Capítulo 17, un grafo *es* una representación compuesta de nodos y enlaces (también denominados aristas). Cuando se dirigen los enlaces (aristas), el grafo de flujo es un grafo dirigido.

McCabe [MCC94] identifica un número importante de usos para las métricas de complejidad:

> Las métricas de complejidad pueden emplearse para predecir la información crítica sobre la fiabilidad y mantenimiento de sistemas software de análisis automáticos de código fuente (o información de diseño procedimental). Las métricas de complejidad también realimentan la información durante el proyecto de software para ayudar a controlar la [actividad del diseño]. Durante las pruebas y el mantenimiento, proporcionan una detallada información sobre los módulos software para ayudar a resaltar las áreas de inestabilidad potencial.

La métrica de complejidad más ampliamente usada (y debatida) para el software es la complejidad *ciclomática,* originalmente desarrollada por Thomas McCabe [MCC76] y estudiando con detalle en la Sección 17.4.2.

La métrica de McCabe proporciona una medida cuantitativa para probar la dificultad y una indicación de la fiabilidad última. Estudios experimentales indican una fuerte correlación entre la métrica de McCabe y el número de errores que existen en el código fuente, así como el tiempo requerido para encontrar y corregir dichos errores.

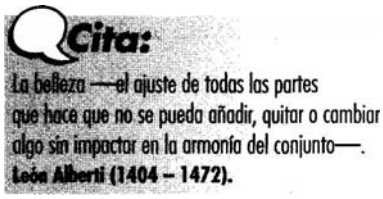
La complejidad ciclomática es solamente una más de los muchos métricas de complejidad.

McCabe también defiende que la complejidad ciclomática puede emplearse para proporcionar una indicación cuantitativa del tamaño máximo del módulo. Recogiendo datos de varios proyectos de programación reales, ha averiguado que una complejidad ciclomática de 10 parece ser el límite práctico superior para el tamaño de un módulo. Cuando la complejidad ciclomática de los módulos excedían ese valor, resultaba extremadamente difícil probar adecuadamente el módulo. Vea en el Capítulo 17 un estudio sobre la complejidad ciclomática como guía para el diseño de casos de prueba de caja blanca.

19.4.3. Métricas de diseño de interfaz

Aunque existe una significativa cantidad de literatura sobre el diseño de interfaces hombre-máquina (vea el Capítulo 15), se ha publicado relativamente poca información sobre métricas que proporcionen una visión interna de la calidad y facilidad de empleo de la interfaz.

Sears [SEA93] sugiere la conveniencia de la representación (CR) como una valiosa métrica de diseño para interfaces hombre-máquina. Una IGU (Interfaz Gráfica de Usuario) típica usa entidades de representación —iconos gráficos, texto, menús, ventanas y otras— para ayudar al usuario a completar tareas. Para realizar una tarea dada usando una IGU, el usuario debe moverse de una entidad de representación a otra. Las posiciones absolutas y relativas de cada entidad de representación, la frecuencia con que se utilizan y el «coste» de la transición de una entidad de representación a la siguiente contribuirán a la conveniencia de la interfaz.



Para una representación específica (por ejemplo, un diseño de una IGU específica), se pueden asignar costes a cada secuencia de acciones de acuerdo con la siguiente relación:

$$\text{Costes} = \sum [\text{frecuencia de transición } (k) \times \text{coste de transición } (k)] \quad (19.7)$$

donde k es la transición específica de una entidad de representación a la siguiente cuando se realiza una tarea específica. Esta suma se da con todas las transiciones de una tarea en particular o conjunto de tareas requeridas para conseguir alguna función de la aplicación. El coste puede estar caracterizado en términos de tiempo, retraso del proceso o cualquier otro valor razonable, tal como la distancia que debe moverse el ratón entre entidades de la representación.

La conveniencia de la representación se define como:

$$CR = 100 \times [(\text{coste de la representación Óptima CR}) / (\text{coste de la representación propuesta})] \quad (19.8)$$

donde CR = 100 para una representación Óptima.

Para calcular la representación óptima de una IGU, la superficie de la interfaz (el área de la pantalla) se divide en una cuadrícula. Cada cuadro de la cuadncula representa una posible posición de una entidad de la representación. Para una cuadrícula con N posibles posi-

ciones y K diferentes entidades de representación para colocar, el número posible de distribuciones se representa de la siguiente manera [SEA93]:

número posible de distribuciones =
$= [N! / (K! \times (N - K)!] \times K!$ (19.9)

A medida que crece el número de posiciones de representación, el número de distribuciones posibles se hace muy grande. Para encontrar la representación óptima (menor coste). Sears [SEA93] propone un algoritmo de búsqueda en árbol.

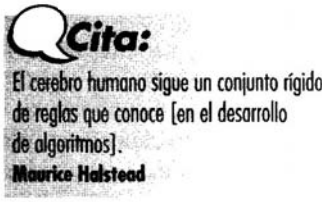
las métricas del diseno de interface son válidas, pero sobre todo, son absolutamente necesarios para asegurar que a los usuarios finales les agrada la inteface y estén satisfechos con los interacciones requeridas.

La **CR** se emplea para valorar diferentes distribuciones propuestas de IGU y la sensibilidad de una representación en particular a los cambios en las descripciones de tareas (por ejemplo, cambios en la secuencia y/o frecuencia de transiciones). El diseñador de interfaces puede emplear el cambio en la conveniencia de la representación, **ACR,** como guía en la elección de la mejor representación de IGU para una aplicación en particular.

Es importante apuntar que la selección de un diseño de IGU puede guiarse con métricas tales como **CR,** pero el árbitro final debería ser la respuesta del usuario basada en prototipos de IGU. Nielsen y Levy [NIE94] informan que «existe una gran posibilidad de éxito si se elige una interfaz basándose solamente en la opinión del usuario. El rendimiento medio de tareas de usuario y su satisfacción con la IGU están altamente relacionadas.»

19.5 MÉTRICAS DEL CÓDIGO FUENTE

La teoría de Halstead de la ciencia del software [HAL77] es «probablemente la mejor conocida y más minuciosamente estudiada... medidas compuestas de la complejidad (software)» **[CURSO].** La ciencia del software propone las primeras «leyes» analíticas para el software de computadora'.



La ciencia del software asigna leyes cuantitativas al desarrollo del software de computadora, usando un conjunto de medidas primitivas que pueden obtenerse una vez que se ha generado o estimado el código después de completar el diseño. Estas medidas se listan a continuación.

n_1: el número de operadores diferentes que aparecen en el programa

n_2: el número de operandos diferentes que aparecen en el programa

N_1: el número total de veces que aparece el operador

N_2: el número total de veces que aparece el operando

los operadores incluyen todos las construcciones del flujo de control, condiciones y operaciones matemáticas. los operandos agrupan todos las variables y constantes del programo.

Halstead usa las medidas primitivas para desarrollar expresiones para la *longitud* global del programa; *volumen mínimo potencial* para un algoritmo; el *volumen real* (número de bits requeridos para especificar un programa); *el nivel del programa* (una medida de la complejidad del software); *nivel del lenguaje* (una constante para un lenguaje dado); y otras características tales como esfuerzo de desarrollo, tiempo de desarrollo e incluso el número esperado de fallos en el software.

Halstead expone que la longitud N se puede estimar como:

$$N = n_1 \log_2 n_1 + n_2 \log_2 n_2 \quad (19.10)$$

y el volumen de programa se puede definir como:

$$V = N \log_2 (n_1 + n_2) \quad (19.11)$$

Se debería tener en cuenta que V variará con el lenguaje de programación y representa el volumen de información (en bits) necesarios para especificar un programa.

Teóricamente, debe existir un volumen mínimo para un algoritmo. Halstead define una relación de volumen L como la relación de volumen de la forma más compacta de un programa con respecto al volumen real del programa. Por tanto, L debe ser siempre menor de uno. En términos de medidas primitivas, la relación de volumen se puede expresar como

$$L = 2/n_1 \times n_2/N_2 \quad (19.12)$$

[9] Se debería resaltar que las «leyes» de Halstead han generado una sustancial controversia y que no todo el mundo está de acuerdo con que la teoría subyacente sea correcta. Sin embargo, se han realizado verificaciones experimentales de las averiguaciones de Halstead para varios lenguajes de programación (por ejemplo, [FEL89]).

El trabajo de Halstead se presta a la verificación experimental y de hecho se ha desarrollado una gran labor de investigación en la ciencia del software. Un estudio de este trabajo estaría fuera del alcance de este libro, pero puede decirse que se ha llegado a un buen acuerdo entre lo previsto analíticamente y los resultados experimentales. Para más información, vea [ZUS90], [FEN91] y [ZUS97].

19.6 MÉTRICAS PARA PRUEBAS

Aunque se ha escrito mucho sobre las métricas del software para pruebas (por ejemplo, [HET93]), la mayoría de las métricas propuestas se concentran en el proceso de prueba, no en las características técnicas de las pruebas mismas. En general, los responsables de las pruebas deben fiarse de las métricas de análisis, diseño y código para que les guíen en el diseño y ejecución de los casos de prueba.

PUNTO CLAVE

Los métricos de lo prueba desembocan en las siguientes categorías: (1) métricas que ayudan o determinar el número de pruebas requeridos en los distintos niveles de la prueba; (2) métricas poro cubrir la pruebo de un componente dado.

Las métricas basadas en la función (Sección 19.3.1) pueden emplearse para predecir el esfuerzo global de las pruebas. Se pueden juntar y correlacionar varias características a nivel de proyecto (por ejemplo, esfuerzo y tiempo para las pruebas, errores descubiertos, número de casos de prueba producidos) de proyectos anteriores con el número de PF producidos por un equipo del proyecto. El equipo puede después predecir los valores «esperados» de estas característicasdel proyecto actual.

La métrica *bang* puede proporcionar una indicación del número de casos de prueba necesarios para examinar las medidas primitivas tratadas en la sección 19.3.2. El número de primitivas funcionales (PFu), elementos de datos (DE), objetos (OB), relaciones (RE), estados **(ES) y** transiciones (TR) pueden emplearse para predecir el número y tipos de prueba del software de caja negra y de caja blanca. Por ejemplo, el número de pruebas asociadas con la interfaz hombre-máquina se puede estimar examinando: (1) el número de transiciones (TR) contenidas en la representación estado-transición del IHM y evaluando las pruebas requeridas para ejecutar cada transición, (2) el número de objetos de datos (OB) que se mueven a través de la interfaz y (3) el número de elementos de datos que se introducen o salen.

Las métricas del diseño arquitectónico proporcionan información sobre la facilidad o dificultad asociada con la prueba de integración (Capítulo 18) y la necesidad de software especializado de prueba (por ejemplo, matrices y controladores). La complejidad ciclomática (una métrica de diseño de componentes) se encuentra en el núcleo de las pruebas de caminos básicos, un método de diseño de casos de prueba presentado en el Capítulo 17. Además, la complejidad ciclomática puede usarse para elegir módulos como candidatos para pruebas más profundas (Capítulo 18). Los módulos con gran complejidad ciclomática tienen más probabilidad de tendencia a errores que los módulos con menor complejidad ciclomática.

Por esta razón, el analista debería invertir un esfuerzo extra para descubrir errores en el módulo antes de integrarlo en un sistema. Es esfuerzo de las pruebas también se puede estimar usando métricas obtenidas de medidas de Halstead (Sección 19.5). Usando la definición del volumen de un programa, V, y nivel de programa, NP, el esfuerzo de la ciencia del software, e, puede calcularse como:

$$\text{NP} = 1 / [(n_1/2) \times (N_2/n_2)] \qquad (19.13a)$$

$$e = V / \text{NP} \qquad (19.13b)$$

El porcentaje del esfuerzo global de pruebas a asignar a un módulo **k** se puede estimar usando la siguiente relación:

$$\text{porcentaje de esfuerzo de pruebas (k)} = e(k)/\Sigma e(i) \qquad (19.14)$$

donde *e(k)* se calcula para el módulo k usando las ecuaciones (19.13) y la suma en el denominador de la ecuación (19.14) es la suma del esfuerzo de la ciencia del software a lo largo de todos los módulos del sistema.

A medida que se van haciendo las pruebas, tres medidas diferentes proporcionan una indicación de la compleción de las pruebas. Una medida de la *amplitud de las pruebas* proporciona una indicación de cuantos requisitos (del número total de ellos) se han probado. Esto proporciona una indicación de la compleción del plan de pruebas. La profundidad de las pruebas es una medida del porcentaje de los caminos básicos independientes probados en relación al número total de estos caminos en el programa. Se puede calcular una estimación razonablemente exacta del número de caminos básicos sumando la complejidad ciclomática de todos los módulos del programa. Finalmente, a medida que se van haciendo las pruebas y se recogen los datos de los errores, se pueden emplear los perfiles *defallos* para dar prioridad y categorizar los errores descubiertos. La prioridad indica la severidad del problema. Las categorías de los fallos proporcionan una descripción de un error, de manera que se puedan llevar a cabo análisis estadísticos de errores.

19.7 MÉTRICAS DEL MANTENIMIENTO

Todas las métricas del software presentadas en este capítulo pueden usarse para el desarrollo de nuevo software y para el mantenimiento del existente. Sin embargo, se han propuesto métricas diseñadas explícitamente para actividades de mantenimiento.

El estándar E E E 982.1-1988 [IEE94] sugiere un índice de madurez del software (IMS) que proporciona una indicación de la estabilidad de un producto software (basada en los cambios que ocurren con cada versión del producto). Se determina la siguiente información:

M_T = número de módulos en la versión actual

F_c = número de módulos en **la** versión actual que se han cambiado

F_a = número de módulos en la versión actual que se han añadido

F_d = número de módulos de la versión anterior **que** se han borrado en **la** versión actual

El índice de madurez del software se calcula de la siguiente manera:

$$IMS = [M_T - (F_a + F_c + F_d)] / M_T \qquad (19.15)$$

A medida que el IMS se aproxima a **1**,0 el producto se empieza a estabilizar. EL IMS puede emplearse también como métrica para la planificación de las actividades de mantenimiento del software. El tiempo medio para producir una versión de un producto software puede correlacionarse con el IMS desarrollándose modelos empíricos para el mantenimiento.

RESUMEN

Las métricas del software proporcionan una manera cuantitativa de valorar la calidad de los atributos internos del producto, permitiendo por tanto al ingeniero valorar la calidad antes de construir el producto. Las métricas proporcionan la visión interna necesaria para crear modelos efectivos de análisis y diseño, un código sólido y pruebas minuciosas.

Para que sea útil en el contexto del mundo real, una métrica del software debe ser simple y calculable, persuasiva, consistente y objetiva. Debería ser independiente del lenguaje de programación y proporcionar una realimentación eficaz para el desarrollador de software.

Las métricas del modelo de análisis se concentran en la función, los datos y el comportamiento (los tres componentes del modelo de análisis). El punto de función **y** la métrica *bang* proporcionan medidas cuantitativas para evaluar el modelo de análisis. Las métricas del diseño consideran aspectos de alto nivel, del nivel de componentes y de diseño de interfaz. Las métricas de diseño de alto nivel consideran los aspectos arquitectónicos y estructurales del modelo de diseño. Las métricas de diseño de nivel de componentes proporcionan una indicación de la calidad del módulo estableciendo medidas indirectas de la cohesión, acoplamiento y complejidad. Las métricas de diseño de interfaz proporcionan una indicación de la conveniencia de la representación de una IGU.

La ciencia del software proporciona un intrigante conjunto de métricas a nivel de código fuente. Usando el número de operadores y operandos presentes en el código, la ciencia del software proporciona una variedad de métricas que pueden usarse para valorar la calidad del programa.

Se han propuesto pocas métricas técnicas para un empleo directo en las pruebas y mantenimiento del software. Sin embargo, se pueden emplear muchas otras métricas técnicas para guiar el proceso de las pruebas y como mecanismo para valorar la facilidad de mantenimiento de un programa de computadora.

REFERENCIAS

[BAS84] Basili, y V.R. D.M. Weiss, «A Methodology for Collecting Valid Software Engineering Data», *IEEE Trans. Software Engineering,* vol. SE-10, 1984, **pp.** 728-738.

[BIE94] Bieman, J.M., y L.M. Ott, «Measuring Functional Cohesion», *IEEE Trans. Software Engineering,* vol. *20,* n.º 8, Agosto 1994, **pp.** 308-320.

[BRI96] Briand, L.C., S. Morasca, y V.R. Basili, «Property-Based Software Engineering Measurement», ***IEEE*** *Trans. Software Engineering,* vol *22,* n.º 1, Enero 1996, **pp.** 68-85.

[CAR90] Card, D., y N. R. L. Glass, *Measuring Software Design Quality*, Prentice-Hall, 1990.

[CAV78] Cavano, J.P., y J.A. McCall, «A Framework for the Measurement of Software Quality», *Proc. ACM Software Quality Assurance Wokshop*, Noviembre 1978, pp. 133-139.

[CHA89] Charette, R.N., Software Engineering Risk Analysis and Management, McGraw-Hill/Intertext, 1989.

[CURSO] Curtis, W., «Management and Experimentation in Software Engineering», *Proc. IEEE,* vol. 68, n.º 9, Septiembre 1980.

[DAV93] Davis, A., et al., «Identifying and Measuring Quality in a Software Requirements Specification», Proc. 1st

Intl. Software Metrics Symposium, IEEE, Baltimore, MD, May 1993, pp. 141-152.

[DEM81] DeMillo, R.A., y R.J. Lipton, «Software Project Forecasting», *Software Metrics* (A.J. Perlis, F.G. Sayward y M. Shaw, ed.), MIT Press, 1981, pp. 77-89.

[DEM82] DeMarco, T., *Controlling Software Projects,* Yourdon Press, 1982.

[DHA95] Dhama, H., «Quantitative Models of Cohesion and Coupling in Software», *Journal of Systems and Software*, vol. 29, n.° 4, Abril 1995.

[EJI91] Ejiogu, L., *Software Engineering with Formal Metrics,* QED Publishing, 1991.

[FEL89] Felican, L., y G. Zalateu, «Validating Halstead's Theory for Pascal Programs», IEEE Trans. Software Engineering, vol. 15, n.° 2, Diciembre 1989, pp. 1630-1632.

[FEN91] Fenton, N., Software Metrics, Chapman & Hall, 1991.

[FEN94] Fenton, N., «Software Measurement: A Necessary Scientific Basis», *IEEE Trans. Software Engineering,* vol. *20,* n.° 3, Marzo 1994, pp. 199-206.

[GRA87] Grady, R. B., y D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program,* Prentice-Hall, 1987.

[HA77] Halstead, M., *Elements of Software Science,* N.ºrth Holland, 1977.

[HEN81] Henry, S., y D. Kafura, «Software Structure Metrics Based on information Flow», *IEEE Trans. Software Engineering,* vol. SE-7, n.° 5, Septiembre 1981, pp. 510-518.

[HET93] Hetzel, B., *Making Software Measurement Work,* QED Publishing, 1993.

[IEE94] *Software Engineering Standards,* 1994, IEEE, 1994.

[KYB84] Kyburg, H.E., *Theory and Measurement,* Cambridge University Press, 1984.

[MCC76] McCabe, T.J., «A Software Complexity Measure», IEEE Trans. Software Engineering, vol. 2, Diciembre 1976, pp. 308-320.

[MCC77] McCall, J., P. Richards, y G. Walters, «Factors in Software Quality», 3 vols., NTIS AD-A049-014, 015, 055, Noviembre 1977.

[MCC89] McCabe, T.J., y C.W. Butler, «Design Complexity Measurement and Testing», CACM, vol. 32, n.° 12, Diciembre 1989, pp. 1415-1425.

[MCC94] McCabe, T.J., y A.H. Watson, «Software Complexity», Crosstalk, vol. 7, n.° 12, Diciembre 1994, pp. 5-9.

[NIE94] Nielsen, J., y J. Levy, «Measuring Usability: Preference vs. Performance», CACM, vol. 37, n.° 4, Abril 1994, pp. 76-85.

[ROC94] Roche, J.M., «Software Metrics and Measurement Principles», *Software Engineering Notes,* ACM, vol. 19, n.° 1, Enero 1994, pp. 76-85.

[SEA93] Sears, A., «Layout Appropriateness: A Metric for Evaluating User Interface Widget Layout», *IEEE Trans. Software Engineering,* vol. 19, n.° 7, Julio 1993, pp. 707-719.

[USA87] *Management Quality Insight*, AFCSP 800-14 (U.S. Air Force), 20 Enero, 1987.

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

[ZUS97] Zuse, H., *A Framework of Software Measurement,* DeGruyter, Nueva York, 1997.

PROBLEMAS Y PUNTOS A CONSIDERAR

19.1. La teoría de la medición es un tema avanzado que tiene una gran influencia en las métricas del software. Mediante [FEN91], [ZUS90] y [KYB84] u otras fuentes, escriba una pequeña redacción que recoja los principales principios de la teoría de la medición. Proyecto individual: Prepare una conferencia sobre el tema y expóngala en clase.

19.2. Los factores de calidad de McCall se desarrollaron durante los años setenta. Casi todos los aspectos de la informática han cambiado dramáticamente desde que se desarrollaron, y sin embargo, los factores de McCall todavía se aplican en el software moderno. ¿Puede sacar alguna conclusión basada en este hecho?

19.3. Por qué no se puede desarrollar una única métrica que lo abarque todo para la complejidad o calidad de un programa?

19.4. Revise el modelo de análisis que desarrolló como parte del Problema 12.13. Mediante las directrices de la Sección 19.3.1., desarrolle una estimación del número de puntos función asociados con SSRB.

19.5. Revise el modelo de análisis que desarrolló como parte del Problema 12.13. Mediante las directrices de la Sección 19.3.2., desarrolle cuentas primitivas para la métrica bang. ¿El sistema SSRB es de dominio de función o de datos?

19.6. Calcule el valor de la métrica bang usando las medidas que desarrolló en el Problema 19.5.

19.7. Cree un modelo de diseño completo para un sistema propuesto por su profesor. Calcule la complejidad estructural y de datos usando las métricas descritas en la Sección 19.4.1. Calcule también las métricas de Henry-Kafura y de morfología para el modelo de diseño.

19.8. Un sistema de información grande tiene 1.140 módulos. Hay 96 módulos que realizan funciones de control y coordinación, y 490 cuya función depende de un proceso anterior. El sistema procesa aproximadamente 220 objetos de datos con una media de tres atributos por objeto de datos. Hay 140 elementos de la base de datos Únicos y 90 segmentos de base de datos diferentes. 600 módulos tienen un solo punto de entrada y de salida. Calcule el ICDE del sistema.

19.9. Investigue el trabajo de Bieman y Ott [BIE94] y desarrolle un ejemplo completo que ilustre el cálculo de su métrica de cohesión. Asegúrese de indicar cómo se determinan las porciones de datos, señales de datos y señales de unión y superunión.

19.10. Seleccione cinco módulos existentes de un programa de computadora. Mediante la métrica de Dhama descrita en la Sección 19.4.2., calcule el valor de acoplamiento de cada módulo.

19.11. Desarrolle una herramienta de software que calcule la complejidad ciclomática de un módulo de lenguaje de programación. Elija el lenguaje.

19.12. Desarrolle una herramienta de software que calcule la conveniencia de la representación de una IGU. Esa herramienta debería permitirle asignar el coste de transición entre las entidades de la representación. (Nota: fíjese en que el tamaño potencial del número de las altemativas de representación crece mucho a medida que aumenta el número de posibles posiciones de cuadrícula.)

19.13. Desarrolle una pequeña herramienta de software que realice un análisis Halstead de un código fuente de un lenguaje de programación a su elección.

19.14. Haga una investigación y escriba un documento sobre la relación entre las métricas de la calidad del software de Halstead y la de McCabe (con respecto a la cuenta de errores). ¿Son convincentes los datos? Recomiende unas directrices para la aplicación de estas métricas.

19.15. Investigue documentos recientes en busca de métricas específicamente diseñadas para el diseño de casos de prueba. Exponga los resultados en clase.

19.16. Un sistema heredado tiene 940 módulos. La última versión requiere el cambio de 90 de estos módulos. Además, se añaden 40 módulos nuevos y se quitaron 12 módulos antiguos. Calcule el índice de madurez del software del sistema.

OTRAS LECTURAS Y FUENTES DE INFORMACIÓN

Sorprendentemente hay un gran número de libros dedicados a las métricas del software, aunque la mayoría están enfocados a métricas de proceso y proyecto excluyendo las métricas técnicas. Zuse [ZUS97] ha escrito el más completo tratado de métricas técnicas publicado hasta la fecha.

Los libros de Card y Glass [CAR90], Zuse [ZUS90], Fenton [FEN91], Ejiogu [EJI91], Moeller y Paulish (Métricas del Software, Chapman & Hall, 1993) y Hetzel [HET93] son referencias sobre las métricas técnicas en detalle. Además, merece la pena examinar los siguientes libros:

Conte, S.D., H.E. Dunsmore, y V.Y. Shen, *Software Engineering Metrics and Models,* Benjamin/Cummings, 1984.

Fenton, N.E., y S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2.ª ed., PWS Publishing Co., 1998.

Grady, R.B. *Practical Software Metrics for Project Mangement and Process Improvement,* Prentice Hall, 1992.

Perlis, A., et al., *Software Metrics: An Analysis and Evaluation*, MIT Press, 1981.

Sheppard, M., *Software Engineering Metrics,* McGraw-Hill, 1992.

La teoría de la medición del software la presentan Denvir, Herman, y Whitty en una colección de documentos *(Proceedings of the International BCS-FACS Workshop: Formal Aspects of Measurement*, Springer-Verlag, 1992). Shepperd *(Foundations of Software Measurement,* Prentice Hall, 1996) también tratan la teoría de la medición en detalle.

Una relación de docenas de usos de las métricas del software están presentadas en [IEE94]. En general, una discusión de cada métrica ha sido reducida a lo esencial «las primitivas» (medidas) requeridas para calcular las métricas y las adecuadas relaciones a efectos de los cálculos. Un apéndice del libro suministra más información y referencias sobre el tema.

Una amplia variedad de fuentes de información sobre métricas técnicas y elementos relacionados están disponibles en Intemet. Una lista actualizada de referencias de páginas web sobre métricas técnicas la puedes encontrar en **http://www.pressman5.com.**