¿Cómo valorar la calidad de los atributos del software?

Linda María Fernández Olvera
UPIICSA-IPN
Ingeniería de Diseño
lfernandezo1500@alumno.ipn.mx

1

Abstract – Las métricas de software proporcionan una forma cuantitativa para valorar la calidad de los atributos internos de producto y, por tanto, permiten valorar la calidad antes de construir el producto. En este ensayo se verán los principios y tipos de métricas que pueden aplicarse.

Índice de Términos – mediciones, métricas, datos, acoplamiento, pruebas.

I. INTRODUCCIÓN

Las mediciones que pueden usarse para valorar la calidad del producto conforme se someten a ingeniería. Estas mediciones de atributos internos del producto ofrecen una indicación en tiempo real de la eficacia de los modelos de requerimientos, diseño y código, así como de la efectividad de los casos de prueba y de la calidad global del software que se va a construir.

En el contexto de la ingeniería del software, una medida proporciona un indicio cuantitativo de la extensión, cantidad, dimensión, capacidad o tamaño de algún atributo de un producto o proceso. La medición es el acto de determinar una medida. El IEEE Standard Glosary of Software Engineering Terminology define métrica como "una medida cuantitativa del grado en el que un sistema, componente o proceso posee un atributo determinado".

II. DESARROLLO

Cuando se ha recolectado un solo punto de datos (por ejemplo, el número de errores descubiertos dentro de un solo componente de software), se establece una medida. La medición ocurre como resultado de la recolección de uno o más puntos de datos.

Un ingeniero de software recolecta medidas y desarrolla métricas de modo que se obtengan indicadores. Un indicador es una métrica o combinación de métricas que proporcionan comprensión acerca del proceso de software, el proyecto de software o el producto en sí. Un indicador proporciona comprensión que permite al gerente de proyecto o a los ingenieros de software ajustar el proceso, el proyecto o el producto para hacer mejor las cosas.

Aunque existe la necesidad de medir y de controlar la complejidad del software, y si bien un solo valor de esta métrica de calidad es difícil de derivar, es posible desarrollar medidas de diferentes atributos internos de programa. Estas medidas y las métricas derivadas de ellas pueden usarse como indicadores independientes de la calidad de los modelos de requerimientos y de diseño.

A. Principios de medición

Un proceso de medición que puede caracterizarse mediante cinco actividades:

• Formulación. La derivación de medidas y métricas de software apropiadas para la representación del software que se está construyendo.

• Recolección. Mecanismo que se usa para acumular datos requeridos para derivar las métricas formuladas.

• Análisis. El cálculo de métricas y la aplicación de herramientas matemáticas.

• Interpretación. Evaluación de las métricas resultantes para comprender la calidad de la representación.

• Retroalimentación. Recomendaciones derivadas de la interpretación de las métricas del producto, transmitidas al equipo de software.

Las métricas de software serán útiles sólo si se caracterizan efectivamente y si se validan de manera adecuada. Los siguientes principios son representativos para la caracterización y validación de métricas:

• Una métrica debe tener propiedades matemáticas deseables, es decir, el valor de la métrica debe estar en un rango significativo.

• Cuando una métrica representa una característica de software que aumenta cuando ocurren rasgos positivos o que disminuye cuando se encuentran rasgos indeseables, el valor de la métrica debe aumentar o disminuir en la misma forma.

• Cada métrica debe validarse de manera empírica en una gran variedad de contextos antes de publicarse o utilizarse para tomar decisiones.

B. Medición de software orientado a meta

El paradigma Meta/Pregunta/Métrica (MPM) fue desarrollado por Basili y Weiss como una técnica para identificar métricas significativas para cualquier parte del proceso de software.

MPM enfatiza la necesidad de:

1) establecer una meta de medición explícita que sea específica para la actividad del proceso o para la característica del producto que se quiera valorar.

2) definir un conjunto de preguntas que deban responderse con la finalidad de lograr la meta.

3) identificar métricas bien formuladas que ayuden a responder dichas preguntas.

Con una meta de medición definida de manera explícita, se desarrolla un conjunto de preguntas. Las respuestas ayudarán al equipo de software (o a otros participantes) a determinar si se logró la meta de medición. Entre las preguntas que pueden plantearse se encuentran:

P1: ¿Los componentes arquitectónicos se caracterizan de forma que compartimentalizan la función y los datos relacionados?

P2: ¿La complejidad de cada componente dentro de las fronteras facilitará la modificación y la extensión?

C. Atributos de las métricas de software efectivas

La métrica derivada y las medidas que conducen a ella deben ser:

• Simple y calculable. Debe ser relativamente fácil aprender cómo derivar la métrica y su cálculo no debe demandar esfuerzo o tiempo excesivo.

• Empírica e intuitivamente convincente. Debe satisfacer las nociones intuitivas del ingeniero acerca del atributo de producto que se elabora.

• Congruente y objetiva. Siempre debe producir resultados que no tengan ambigüedades.

Una tercera parte independiente debe poder derivar el mismo valor de métrica usando la misma información acerca del software.

• Constante en su uso de unidades y dimensiones. El cálculo matemático de la métrica debe usar medidas que no conduzcan a combinaciones extrañas de unidades.

• Independiente del lenguaje de programación. Debe basarse en el modelo de requerimientos, el modelo de diseño o la estructura del programa en sí. No debe depender de los caprichos de la sintaxis o de la semántica del lenguaje de programación.

• Un mecanismo efectivo para retroalimentación de alta calidad. Debe proporcionar información que pueda conducir a un producto final de mayor calidad.

D. Métricas para el modelo de requerimientos

El trabajo técnico en la ingeniería del software comienza con la creación del modelo de requerimientos. En esta etapa se derivan los requerimientos y se establece un cimiento para el diseño. Por tanto, son deseables métricas de producto que proporcionen comprensión acerca de la calidad del modelo de análisis.

Métrica basada en funciones

La métrica de punto de función (PF) puede usarse de manera efectiva como medio para medir la funcionalidad que entra a un sistema. Al usar datos históricos, la métrica PF puede entonces usarse para:

1) estimar el costo o esfuerzo requerido para diseñar, codificar y probar el software.

2) predecir el número de errores que se encontrarán durante las pruebas.

3) prever el número de componentes y/o de líneas fuente proyectadas en el sistema implementado.

Los puntos de función se derivan usando una relación empírica basada en medidas contables (directas) del dominio de información del software y en valoraciones cualitativas de la complejidad del software. Los valores de dominio de información se definen en la forma siguiente:

Número de entradas externas (EE). Cada entrada externa se origina de un usuario o se transmite desde otra aplicación, y proporciona distintos datos orientados a aplicación o información de control. Con frecuencia, las entradas se usan para actualizar archivos lógicos internos (ALI).

Número de salidas externas (SE). Cada salida externa es datos derivados dentro de la aplicación que ofrecen información al usuario. En este contexto, salida externa se refiere a reportes, pantallas, mensajes de error, etc. Los ítems de datos individuales dentro de un reporte no se cuentan por separado.

Número de consultas externas (CE). Una consulta externa se define como una entrada en línea que da como resultado la generación de alguna respuesta de software inmediata en la forma de una salida en línea (con frecuencia recuperada de un ALI).

Número de archivos lógicos internos (ALI). Cada archivo lógico interno es un agrupamiento lógico de datos que reside dentro de la frontera de la aplicación y se mantiene mediante entradas externas.

Número de archivos de interfaz externos (AIE). Cada archivo de interfaz externo es un agrupamiento lógico de datos que reside fuera de la aplicación, pero que proporciona información que puede usar la aplicación.

Con base en el valor PF proyectado, derivado del modelo de requerimientos, el equipo del proyecto puede estimar el tamaño global implementado de la función de interacción del usuario.

Métricas para calidad de la especificación

Las características que pueden usarse para valorar la calidad del modelo de requerimientos y la correspondiente especificación de requerimientos son especificidad (falta de ambigüedad), completitud, corrección, comprensibilidad, verificabilidad, consistencia interna y externa, factibilidad, concisión, rastreabilidad, modificabilidad, precisión y reusabilidad.

E. Métricas para el modelo de diseño

Las métricas de diseño para software de computadora, al igual que todas las demás métricas de software, no son perfectas. Se requiere más experimentación antes de poder usar las medidas de diseño, aunque el diseño sin medición es una alternativa inaceptable. Algunas de las métricas de diseño más comunes para software de computadora son:

Métricas del diseño arquitectónico

Las métricas del diseño arquitectónico se enfocan en características de la arquitectura del programa con énfasis en la estructura arquitectónica y en la efectividad de los módulos o componentes dentro de la arquitectura. Dichas métricas son "caja negra" en tanto no requieren conocimiento alguno del funcionamiento interior de un componente de software particular.

Métricas para diseño orientado a objetos

Hay mucho de subjetivo en el diseño orientado a objetos: un diseñador experimentado "sabe" cómo caracterizar un sistema OO de modo que implemente de manera efectiva los requerimientos del cliente. Pero, conforme un modelo de diseño OO crece en tamaño y complejidad, una visión más objetiva de las características del diseño puede beneficiar tanto al diseñador experimentado (quien adquiere comprensión adicional) como al novato.

En un tratamiento detallado de las métricas de software para sistemas OO, Whitmire describe nueve características distintas y mensurables de un diseño OO:

Tamaño. El tamaño se define en función de cuatro visiones: población, volumen, longitud y funcionalidad. La población se mide al realizar un conteo estático de entidades OO, tales como clases u operaciones. Las medidas de volumen son idénticas a las medidas de población, pero se recolectan de manera dinámica: en un instante de tiempo determinado. La longitud es una 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 un indicio indirecto del valor entregado al cliente por una aplicación OO.

Complejidad. Como el tamaño, existen muchas visiones diferentes de la complejidad del software. Whitmire ve la complejidad en términos de características estructurales al examinar cómo se relacionan mutuamente las clases de un diseño OO.

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

Suficiencia. Whitmire define suficiencia como "el grado en el que una abstracción posee las características requeridas de él o en el que un componente de diseño posee características en su abstracción, desde el punto de vista de la aplicación actual". Dicho de otra forma, se pregunta: "¿Qué propiedades debe poseer esta abstracción (clase) para serme útil?"

Completitud. La única diferencia entre completitud y suficiencia es "el conjunto de características contra las cuales se compara la abstracción o el componente de diseño".La suficiencia compara la abstracción desde el punto de vista de la aplicación actual. La completitud considera múltiples puntos de vista, y plantea la pregunta: "¿qué propiedades
se requieren para representar por completo al objeto de dominio problema?".

Cohesión. Como su contraparte en software convencional, un componente OO debe diseñarse de manera que tenga todas las operaciones funcionando en conjunto para lograr un solo propósito bien definido. La cohesividad de una clase se determina al examinar el grado en el que "el conjunto de propiedades que posee es parte del problema o dominio de diseño".

Primitivismo. Una característica que es similar a la simplicidad, el primitivismo (aplicado tanto a operaciones como a clases), es el grado en el que una operación es atómica, es decir, la operación no puede construirse a partir de una secuencia de otras operaciones contenidas dentro de una clase. Una clase que muestra un alto grado de primitivismo encapsula sólo operaciones primitivas.

Similitud. El grado en el que dos o más clases son similares en términos de su estructura, función, comportamiento o propósito se indica mediante esta medida.

Volatilidad. Como se menciona muchas veces en este libro, los cambios en el diseño pueden ocurrir cuando se modifican los requerimientos o cuando ocurren modificaciones en otras partes de una aplicación, lo que da como resultado la adaptación obligatoria del componente de diseño en cuestión.

La volatilidad de un componente de diseño OO mide la probabilidad de que ocurrirá un cambio.

En realidad, las métricas de producto para sistemas OO pueden aplicarse no sólo al modelo de diseño, sino también al de requerimientos.

Métricas orientadas a clase: la suite de métricas CK

La clase es la unidad fundamental de un sistema OO. Por tanto, las medidas y métricas para una clase individual, la jerarquía de clase y las colaboraciones de clase serán invaluables cuando se requiera valorar la calidad del diseño OO. Una clase encapsula datos y la función que los manipula. Con frecuencia es el "padre" de las subclases (en ocasiones llamadas hijos) que heredan sus atributos y operaciones. Usualmente colabora con otras clases. Cada una de estas características puede usarse como la base para la medición.

Métricas orientadas a clase: La suite de métricas MOOD

Harrison, Counsell y Nithi [Har98b] proponen un conjunto de métricas para diseño orientado a objeto que proporciona indicadores cuantitativos para características de diseño OO.

Aunque muchos factores afectan la complejidad, comprensibilidad y mantenimiento del software, es razonable concluir que, conforme el valor de FA aumenta, la complejidad del software OO también aumentará, y como resultado pueden sufrir la comprensibilidad, el mantenimiento y el potencial de reuso.

Métricas OO propuestas por Lorenz y Kidd

En su libro acerca de métricas OO, Lorenz y Kidd dividen las métricas basadas en clase en cuatro amplias categorías; cada una tiene una relación en el diseño en el nivel de componentes: tamaño, herencia, internos y externos. Las métricas orientadas a tamaño para una clase de diseño OO se enfocan en conteos de atributos y operaciones para una clase individual y en valores promedio para el sistema OO como un todo. Las métricas basadas en herencia se enfocan en la forma en la que las operaciones se reutilizan a lo largo de la jerarquía de clases. Las métricas para interiores de clase se fijan en la cohesión y en los conflictos orientados a código. Y las métricas externas examinan el acoplamiento y el reuso.

Métricas de diseño en el nivel de componente

Las métricas de diseño en el nivel de componente para componentes de software convencional se enfocan en las características internas de un componente de software e incluyen medidas de cohesión de módulo, acoplamiento y complejidad. Dichas medidas pueden ayudarlo a juzgar la calidad de un diseño en el nivel de componente.

Las métricas de diseño en el nivel de componente pueden aplicarse una vez desarrollado el diseño procedural y son "cajas de cristal" en tanto requieren conocimiento del funcionamiento interior del módulo en el que se está trabajando. Alternativamente, pueden demorarse hasta que el código fuente esté disponible.

Métricas orientadas a operación

Puesto que la clase es la unidad dominante en los sistemas OO, se han propuesto menos métricas para operaciones que residen dentro de una clase. Lo que sugiere que la estructura de conectividad de un sistema puede ser más importante que el contenido de los módulos individuales." Sin embargo, puede obtenerse algo de comprensión al examinar las características promedio para los métodos (operaciones). Tres métricas simples, propuestas por Lorenz y Kidd, son apropiadas:

Tamaño promedio de operación (TOprom). El tamaño puede determinarse al contar el número de líneas de código o el de mensajes enviados por la operación. Conforme aumenta el número de mensajes enviados por una sola operación, es probable que las responsabilidades no se hayan asignado bien dentro de una clase.

Complejidad de la operación (CO). La complejidad de una operación puede calcularse usando cualquiera de las métricas de complejidad propuestas para software convencional. Puesto que las operaciones deben limitarse a una responsabilidad específica, el diseñador debe luchar por mantener la CO tan baja como sea posible.

Número promedio de parámetros por operación (NPprom). Mientras más grande sea el número de parámetros de operación, más compleja es la colaboración entre objetos. En general, el NPprom debe mantenerse tan bajo como sea posible.

Métricas de diseño de interfaz de usuario

Aunque hay considerable literatura acerca del diseño de interfaces hombre/computadora, se ha publicado relativamente poca información acerca de las métricas que proporcionarían comprensión de la calidad y de la usabilidad de la interfaz.

La corrección de la plantilla (CP) es una métrica de diseño valioso para las interfaces hombre/computadora. Una GUI típica usa entidades de plantilla (íconos gráficos, texto, menús, ventanas y similares) para auxiliar al usuario a completar tareas. Para lograr una tarea dada usando una GUI, el usuario debe moverse de una entidad de plantilla a la siguiente. La posición absoluta y relativa de cada entidad de plantilla, la frecuencia con la que se usa y el "costo" de la transición desde una entidad de plantilla a la siguiente contribuirán a la corrección de la interfaz.

F. Métricas para código fuente

Halstead asignó leyes cuantitativas al desarrollo de software de computadora, usando un conjunto de medidas primitivas que pueden derivarse después de generar el código o de que el diseño esté completo. Las medidas son:

n1 = número de operadores distintos que aparecen en un programa

n2 = número de operandos distintos que aparecen en un programa

N1 = número total de ocurrencias de operador

N2 = número total de ocurrencias de operando

Halstead usa estas medidas primitivas para desarrollar: expresiones para la longitud de programa global, volumen mínimo potencial para un algoritmo, volumen real (número de bits requeridos para especificar un programa), nivel del programa (una medida de complejidad del software), nivel del lenguaje (una constante para un lenguaje determinado) y otras características, como esfuerzo de desarrollo, tiempo de desarrollo e incluso el número proyectado de fallas en el software.

Halstead muestra que la longitud N puede estimarse

$N = n1 \log2 n1 + n2 \log2 n2$

y el volumen del programa puede definirse

$V = N \log2 (n1 + n2)$

Debe observarse que V variará con el lenguaje de programación y representa el volumen de información (en bits) requerido para especificar un programa.

Teóricamente, debe existir un volumen mínimo para un algoritmo particular. Halstead define una razón de volumen L como la razón del volumen de la forma más compacta de un programa al volumen del programa real. En realidad, L siempre debe ser menor que 1.

G. Métricas para pruebas

Las métricas del diseño arquitectónico proporcionan información acerca de la facilidad o dificultad asociada con las pruebas de integración y de la necesidad de software de pruebas especializado (por ejemplo, resguardos y controladores). La complejidad ciclomática (una métrica de diseño en el nivel de componente) yace en el centro de la prueba de ruta base, un método de diseño de casos de prueba. Además, la complejidad ciclomática puede usarse para dirigirse a módulos como candidatos para prueba de unidad extensa. Los módulos con alta complejidad ciclomática tienen más probabilidad de ser proclives al error que los módulos donde su complejidad ciclomática es menor. Por esta razón, debe emplear esfuerzo por arriba del promedio para descubrir errores en tales módulos antes de que se integren en un sistema.

H. Métricas para mantenimiento

IEEE Std. 982.1-1988 [IEE93] sugiere un índice de madurez de software (IMS) que proporcione un indicio de la estabilidad de un producto de software (con base en cambios que ocurran para cada liberación del producto).

Conforme el IMS tiende a 1.0, el producto comienza a estabilizarse. El IMS también puede usarse como una métrica para planificar actividades de mantenimiento de software. El tiempo medio para producir una liberación de un producto de software puede correlacionarse con el IMS, y es posible desarrollar modelos empíricos para esfuerzo de mantenimiento.

III. CONCLUSIÓN

Las métricas de software proporcionan una forma cuantitativa para valorar la calidad de los atributos internos de producto y, por tanto, permiten valorar la calidad antes de construir el producto. Las métricas proporcionan la comprensión necesaria para crear modelos efectivos de requerimientos y diseño, código sólido y pruebas amplias.

Las métricas para el modelo de requerimientos se enfocan en los tres componentes del modelo: la función, los datos y el comportamiento. Las métricas para diseño consideran arquitectura, diseño en el nivel de componentes y conflictos en el diseño de interfaz. Las métricas de diseño arquitectónico consideran los aspectos estructurales del modelo de diseño. Las métricas de diseño en el nivel de componente proporcionan un indicio de la calidad del módulo al establecer medidas indirectas para cohesión, acoplamiento y complejidad. Las métricas de diseño de interfaz de usuario ofrecen un indicio de la facilidad con la que puede usarse una GUI. Las métricas webapp consideran aspectos de la interfaz de usuario, así como estética, contenido y navegación de la webapp.

Pocas métricas de producto se han propuesto para uso directo en las pruebas de software y en el mantenimiento. Sin embargo, muchas otras métricas de producto pueden usarse para guiar el proceso de pruebas y como mecanismo para valorar la capacidad de mantenimiento de un programa de cómputo. Para valorar la comprobabilidad de un sistema OO, se ha propuesto una gran variedad de métricas OO.

REFERENCIAS

[1] Roger S. Pressman, “Ingeniería del Software, un enfoque práctico” 7ª ed.Connecticut University, New York: McGraw-Hill, 2010.