Atributos de Calidad para Componentes COTS

Manuel F. Bertoa y Antonio Vallecillo
Departamento de Lenguajes y Ciencias de la Computación
Universidad de Málaga. 29071 Málaga, España
{bertoa,av}@lcc.uma.es

Resumen

Conforme cobra auge el desarrollo de aplicaciones basadas en componentes COTS, y empiezan a aparecen las primeras casas comerciales que venden componentes COTS con éxito, aumenta la necesidad de disponer de parámetros de calidad que permitan evaluarlos y seleccionarlos de acuerdo a los requisitos de calidad de los usuarios. Partiendo de las distintas propuestas existentes sobre características de calidad para los productos genéricos software (especialmente las recogidas por ISO e IEEE), este trabajo ofrece una lista de atributos y de sus posibles métricas que son aplicables específicamente a los componentes COTS, y que permiten describir (y medir) no sólo su funcionalidad, sino también muchas de sus propiedades extra-funcionales. El modelo de calidad propuesto se contrasta también con los ya existentes, cuyo carácter demasiado generalista dificulta su implantación y utilización efectiva para el caso concreto de los componentes COTS.

1. Introducción

Actualmente asistimos a grandes cambios en la forma en la que se desarrollan las aplicaciones software. La creciente necesidad de realizar sistemas complejos en cortos periodos de tiempo, a la vez que con menores esfuerzos tanto humanos como económicos, está favoreciendo el desarrollo de lo que se conoce como Ingeniería del Software Basada en Componentes (ISBC). Esta nueva disciplina se apoya en componentes software ya desarrollados y de índole comercial, que se conocen como componentes COTS (*commercial off-the-shelf*).

Construir una aplicación se convierte por tanto en la búsqueda y emsamblaje de piezas prefabricadas, desarrolladas en su mayoría por terceras casas, y cuyo código no puede modificarse. Bajo este nuevo planteamiento, cobran especial interés los procesos de búsqueda y selección de los componentes COTS apropiados para construir las aplicaciones.

Los principales esfuerzos de la comunidad de software en estos temas se han basado hasta ahora en los aspectos funcionales de los componentes, es decir, en la funcionalidad que ofrecen. Sin embargo, por lo general se han venido obviando muchos de los aspectos de calidad que intervienen a la hora de seleccionar componentes y ensamblarlos para construir aplicaciones que satisfagan los requisitos del cliente. Este tipo de aspectos, que llamaremos "'extra-funcionales" cada vez acaparan más la atención de los arquitectos e ingenieros del software. Por un lado, los requisitos extra-funcionales —por su naturaleza global— pueden afectar a varias partes del sistema, y por ello tendrán prioridad si entran en conflicto con los requisitos funcionales. Además, el cuidadoso análisis de los atributos de calidad puede mejorar el proceso de discriminación de los productos COTS candidatos que cumplan el núcleo de requisitos funcionales. Por ejemplo, si dos componentes implementan misma funcionalidad, sus atributos extra-funcionales pueden ser el criterio decisivo en el proceso de selección.

Podemos considerar varias causas para esta omisión de la valoración en los requisitos extra-funcionales. La primera es que no existe una definición consensuada de las características o atributos de calidad que hay que medir. Así, podemos encontrar diversas clasificaciones en la literatura: desde los factores de calidad propuestos en 1.977 por McCall [12], el modelo de calidad de Boehm de 1.976 [3], los estándares internacionales ISO 9126 [11] e ISO 14598 [10], la lista de atributos para la valoración citada para el modelo COCOTS en [1] basada en estándares de IEEE [6], y otras varias [4, 14].

La siguiente causa de dificultad es la falta de información acerca de este tipo de atributos que suministran los vendedores de componentes COTS. Una visita virtual a los portales de los principales vendedores lo pone en evidencia, como ocurre con Componentsource (`www.componentsource.com`), o Flashline (`www.flashline.com`).

Añadido a todo esto, podemos encontrar una ausencia casi total de métricas que pudieran dar una estimación más objetiva de estos atributos. Este hecho se ve empeorado por la situación actual de los estándares internacionales relativos a la calidad del producto software. Las normas ISO 9126 e ISO 14598, encargadas de especificar estos temas, se encuentran ahora mismo en proceso de revisión. El proyecto SquaRE [2] es el encargado de tratar de hacerlas con-

verger, eliminando algunas de las lagunas e inconsistencias que presentan. Por otro lado, es importante señalar que los estándares internacionales proporcionan guías y modelos de calidad para temas muy generales y su aplicación se suele centrar en temas de un gran espectro. Por tanto, no suelen estar pensadas para ofrecer soluciones en temas muy concretos. En este sentido, nuestro trabajo se concentra en un ámbito muy particular, el de las métricas para los componentes COTS.

Nuestro objetivo en este articulo es caracterizar los atributos de calidad para que sirvan de ayuda en la selección de componentes COTS y proponer algunas métricas en los casos en los que esto sea posible.

Este documento está estructurado en 7 secciones. Tras esta introducción, la sección 2 define los términos utilizados y presentan las distintas clasificaciones de características de calidad para productos software. En la sección 3 se clasifican las características de calidad particulizándolas para los componentes COTS y se propone un modelo de calidad para componentes. La sección 4 describe un conjunto de atributos de calidad con sus posibles métricas según las clasificaciones vistas. La sección 5 propone la documentación de los atributos utilizando plantillas XML. Finalmente, en la sección 6 se comentan algunos trabajos relacionados con la presente propuesta y en la sección 7 se presentan las conclusiones.

2. Características de Calidad de Componentes

En general, no existe un consenso a la hora de definir y clasificar las características de calidad que debe presentar un producto software. Por tanto, hemos basado nuestra propuesta en los estándares internacionales, fundamentalmente el ISO 9126. Siguiendo su terminología, entendemos por *característica de calidad* de un producto software a un conjunto de propiedades mediante las cuales se evalúa y describe su calidad. Un característica se puede refinar en múltiples niveles de *sub-características*.

Llamaremos *atributo* a una propiedad de calidad a la que puede asignársele una *métrica*, donde una métrica es un procedimiento que examina un componente y produce un dato simple, un símbolo (p.e. Excelente, Sí, No) o un número . Hay que tener en cuenta que no todas las propiedades que son medibles (p.e. la "demostrabilidad").

Un *modelo de calidad* es el conjunto de características y sub-características, y de cómo estas se relacionan entre sí. Por supuesto, el modelo de calidad a utilizar va a depender del tipo de producto a evaluar. En este sentido, los estándares y propuestas existentes definen modelos de calidad generales. Una de nuestras contribuciones es la definición de un modelo de calidad para componentes software, que desarrollamos en la sección 3.

Características	Sub-características
Funcionalidad	*Adecuación* *Corrección* *Interoperatividad* *Conformidad* *Seguridad*
Fiabilidad	*Madurez* *Recuperabilidad* *Tolerancia a Fallos*
Facilidad de Uso	*Facilidad de Aprendizaje* *Facilidad de Comprensión* *Operatividad*
Eficiencia	*Comportamiento Temporal* *Utilización de Recursos*
Mantenibilidad	*Estabilidad* *Analizabilidad* *Cambiabilidad* *Facilidad de Prueba*
Portabilidad	*Facilidad de Instalación* *Adecuación* *Remplazabilidad* *Adaptabilidad*

Cuadro 1: Características de calidad según ISO 9126

Nuestro principal objetivo es detectar los atributos pueden describir los proveedores (externos o internos) de componentes COTS en la información que suministran acerca de los mismos y que, por tanto, permitirían facilitar su valoración y selección por parte de los diseñadores y desarrolladores de productos software.

Las características y sub-características que definen el modelo general de calidad del software de la ISO 9126 se muestra en la Tabla 1.

Partiendo de ese modelo de calidad, vamos a tratar de particularizarlo realizando distintos tipos de clasificaciones.

1. En primer lugar, necesitamos discriminar entre aquellas características que tienen sentido para los componentes aislados (características *locales*) o bien deben ser valoradas a nivel de la arquitectura software de la aplicación (que llamaremos características *globales*). Por ejemplo, la "tolerancia a fallos" es una típica característica que va a depender de la arquitectura de la aplicación, mientras que la "madurez" es más propia de los componentes.

2. El instante en el cual una característica puede ser observada o medida, permite establecer otra clasificación de las características de un producto. Así, tenemos dos posibles categorías dependiendo de si la característica es observable en tiempo de ejecución (p.e. el rendimiento) o durante

el ciclo de vida del producto (p.e. la mantenibilidad) [13].

3. Como se menciona en los estándares de ISO, es importante identificar los usuarios a los que se dirige el modelo. En nuestra propuesta, los usuarios son fundamentalmente los arquitectos software, que necesitan evaluar los componentes COTS que van a formar parte de su aplicación. Así, las interfaces de los componentes objeto de nuestro estudio son más las interfaces programáticas (es decir, las APIs que definen las formas de acceder desde otros programas a los servicios que ofrecen los componentes), que las interfaces de usuario.

4. Para componentes COTS, es fundamental distinguir entre métricas internas y externas. Las internas miden los atributos internos del producto final o de los productos intermedios (p.e. la especificación o el código fuente) durante el diseño y la codificación. Las externas son las que realizan las mediciones en función del comportamiento del sistema durante las pruebas y la operación del componente. Por tanto, debido al carácter de caja negra de los componentes COTS, son las métricas externas las que interesan. Esto no quita que algunas de las características internas den una medida indirecta de las externas, e incluso que puedan tener efectos sobre la arquitectura final. Así por ejemplo, el tamaño de un componente puede ser importante a la hora de tener en cuenta los requisitos de espacio de la aplicación.

Por último, es importante reseñar que además de las características de calidad en un componente, hay otro conjunto de características no relacionadas directamente con la calidad como pueden ser el precio, la asistencia técnica, las condiciones de licencia, etc., que también son necesarias a la hora de valorar el componente más adecuado. En este artículo nos centramos inicialmente en las características de calidad, dejando su extensión a otro tipo de características para trabajos futuros.

3. Clasificación de las características de calidad

Como hemos indicado anteriormente, no todas las características de un producto software son aplicables a un componente COTS. La tabla 2 muestra el modelo de calidad que proponemos para este tipo de componentes.

Básicamente se trata del modelo de calidad de ISO (ver Tabla 1), donde desaparecen la característica *Portabilidad* y las sub-características *Tolerancia a Fallos*, *Estabilidad* y *Analizabilidad*. Además, aparecen como nuevas las sub-características *Compatibilidad* y *Complejidad*. Es importante observar que algunas de ellas cambian de sentido, como detallamos a continuación (las características que cambian su sentido aparecen en negrita en la tabla).

Funcionalidad Esta característica mantiene el mismo sentido para los componentes que para un producto software. Podríamos expresarla como la capacidad del componente para proporcionar las funciones que satisfagan las necesidades establecidas o implícitas cuando se usa bajo las condiciones especificadas.

Se añade la *Compatibilidad* que indica con que versiones predecesoras es compatible un componente, es decir, si se mantiene la funcionalidad del mismo al integrarse en el contexto donde estaba la versión anterior que se desea sustituir.

Fiabilidad Es aplicable directamente a los componentes, y fundamental para su reutilización. La sub-característica de *Madurez* la medimos en función de los cambios que sufren las versiones comerciales y la velocidad a la que aparecen. La *Recuperabilidad*, en función de una serie de atributos que pueden estar presentes o no en su diseño, indicando los métodos que se utilizan para implementarlos.

Facilidad de Uso Esta característica, y todas sus sub-características cambian de sentido, dado que un componente no será utilizado por un usuario final directamente sino por los diseñadores y desarrolladores de aplicaciones software. La facilidad de uso real de un componente debe interpretarse como la capacidad del componente para ser utilizado en la construcción de un producto o sistema software.

En este sentido buscaremos atributos que midan la facilidad de uso del componente durante el diseño de las aplicaciones, añadiendo la *Complejidad* como una nueva sub-característica cuyo sentido es dar una medida de la complejidad del uso e integración del componente en un producto o sistema software.

Eficiencia Vamos a respetar la definición y clasificación que hace la ISO de esta característica (en *Comportamiento Temporal* y *Utilización de Recursos*), aunque la mayoría de las propuestas de otros autores prefieren hablar de *Rendimiento (Performance)* y usan otra sub-clasificación. En cualquier caso, los atributos que vamos a definir para medir estas características son aplicables de forma independiente al nombre y clasificación que se utilice.

Mantenibilidad La facilidad de mantenimiento o mantenibilidad mide la capacidad de un producto software de ser modificado, entendiendo

Características	Sub-características (Tiempo de ejecución)	Sub-características (ciclo de vida)
Funcionalidad	Corrección Seguridad	Idoneidad Interoperatividad Conformidad *Compatibilidad*
Fiabilidad	Recuperabilidad	Madurez
Facilidad de Uso		**Facilidad de Aprendizaje** **Facilidad de Comprensión** **Operatividad** *Complejidad*
Eficiencia	Comportamiento Temporal Utilización de Recursos	
Mantenibilidad		Cambiabilidad Facilidad de Prueba

Cuadro 2: Modelo de Calidad para componentes COTS

por modificación cualquier corrección, mejora o adaptación del software. En este sentido y aunque las modificaciones internas no serán realizadas por el usuario del componente (desarrollador), sí necesitará probar el componente antes de incluirlo en su aplicación o cambiar alguno de las parámetros que se pueden particularizar. Por ello, las sub-características *Cambiabilidad* y *Facilidad de Prueba* son las que deben ser medidas para los componentes.

Portabilidad Esta característica se define como la capacidad de poder ser reutilizado en distintos entornos. En COTS, esa es la esencia misma de los componentes, que son diseñados y desarrollados específicamente para ser reutilizados. (Es importante observar que en la Ingeniería del software Basada en Componentes, reutilizar significa no sólo usar más de una vez, sino usar en distintos contextos [14]).

4. Atributos de los Componentes

En esta sección vamos a describir los atributos de calidad para medir las distintas características que forman parte del modelo de calidad que proponemos. Para ello vamos a dividirlos según sean medibles durante la ejecución de los componentes, o formen parte de su ciclo de vida.

Para medir los atributos utilizaremos varias métricas que describimos a continuación:

Presencial Esta métrica indica si un atributo está presente en el componente o no. Para ello utilizaremos una variable booleana y una variable de tipo *string* que especifique como o con que método, formato etc..., se implementa el atributo en cuestión en caso de estar presente.

Tiempo Esta métrica se utiliza para medir intervalos de tiempo, utiliza una variable de tipo entero para indicar el valor absoluto y una variable de tipo string para indicar las unidades, por ejemplo, 10 segundos o 4 meses.

Nivel Se utiliza para indicar un grado de esfuerzo, habilidad, etc... cuando se da una medida subjetiva dentro de una escala de valores. Es una variable entera que puede tomar el siguiente rango de valores: *0 (Muy Bajo), 1 (Bajo), 2 (Medio), 3 (Alto), 4 (Muy Alto).*

Ratio Se utiliza para dar un porcentaje (entre 0 y 100) mediante una variable de tipo entero.

Además de estas métricas básicas, para medir algunos atributos utilizaremos también ***índices***, que se obtienen a partir de dos métricas básicas, generando un "indicador". Un ejemplo de este tipo de medidas lo encontramos en el "índice de complejidad", un atributo que relaciona el número de parámetros configurables de un componente con el número de interfaces que implementa. En general, es conveniente distinguir entre métricas básicas e indicadores, pues estos últimos son valores derivados de los primeros. Sin embargo, la expresividad de alguno de estos índices

justifica que los hayamos incluido en nuestro modelo de calidad.

4.1. Atributos medibles en tiempo de ejecución

La tabla 3 resume los atributos de calidad para componentes COTS que son observables durante la ejecución del componente, agrupados según la subcaracterística que miden e indicando el tipo de variable que utilizan.

4.1.1. Atributos asociados a "Corrección (Accuracy)"

- **Precisión (Precision)**

  Este atributo evalúa el porcentaje de resultados obtenidos con precisión insuficiente respecto a los requisitos del usuario.

  Aparte de medir la precisión computacional de las operaciones del componente, este atributo también va a permitir medir, por ejemplo, el nivel de "actualización" (*freshness*) de la información que devuelven las operaciones invocadas. Este es el caso de un componente que devuelve datos que mantiene en la cache en aras de incrementar la eficiencia, en detrimento del grado de "actualización"de la información que proporciona.

  Este atributo se mide con una variable de tipo *Ratio*, calculada dividiendo el número de resultados imprecisos obtenidos entre el número total de resultados que devuelven las operaciones del componente al ser invocadas un determinado número de veces.

- **Exactitud Computacional (Computational Accuracy)**

  Este atributo evalúa el número de resultados incorrectos que devuelven las operaciones del componente, respecto a sus especificaciones.

  Se mide mediante una variable de tipo *Ratio*, calculada dividiendo el número de resultados incorrectos encontrados entre el número total de operaciones invocadas.

4.1.2. Atributos asociados a "Seguridad (Security)"

- **Cifrado de datos (Data Encryption)**

  El atributo señala si el componente es capaz de manejar los datos de forma cifrada, y si es así, que método utiliza.

  Utilizaremos por tanto una métrica de tipo *Presencial*.

- **Capacidad de Control (Controllability)**

  Indica si el componente permite realizar un control sobre el acceso a los servicios que proporciona.

  Ejemplos son componentes que proporcionen interfaces con funciones para realizar identificación o autenticación de los usuarios.

  Utilizaremos una métrica de tipo *Presencial*, indicando el mecanismo de control implementado.

- **Capacidad para Auditar (Auditability)**

  Este atributo indica si el componente tiene la posibilidad de registrar las operaciones que lleva a cabo y qué usuario las realiza.

  Por ejemplo, el componente puede proporcionar funciones que permitan abrir, escribir y cerrar un archivo de registro (*log file*), anotando el usuario, el tipo de operación, los datos implicados o el instante en el cual se realiza cada operación.

  Utilizaremos una métrica de tipo *Presencial*, que indique como realiza el registro de operaciones y como recuperarlo.

4.1.3. Atributos asociados a "Recuperabilidad"

- **Secuencializable (Serializable)**

  Indica si el componente es capaz de secuenciar su código y su estado, para transferirlo entre diversos agentes y recuperarlos con posterioridad.

  Utilizaremos una métrica de tipo *Presencial*.

- **Persistente (Persistent)**

  Este atributo señalará si el componente puede almacenar su estado de forma persistente.

  Utilizaremos una métrica de tipo *Presencial*.

- **Transaccional (Transactional)**

  Este atributo debe indicar si el componente suministra alguna interfaz que permita que sus operaciones estén sujetas a transacciones (p.e. un componente CORBA que implemente el interfaz "`Resource`").

  Utilizaremos una métrica de tipo *Presencial*.

- **Tratamiento de Errores (Error Handling)**

  Este atributo debe indicar si se realiza algún tipo de tratamiento de errores y en su caso el tipo de tratamiento de errores que lleva a cabo el componente.

  Se utiliza un a variable *Presencial* que deberá indicar el nivel de tratamiento de los errores, p.e. siguiendo un modelo clásico como el siguiente:

  Detectar Los detecta pero no realiza ninguna acción

Detectar y avisar Los detecta y avisa de ello

Tratar Los detecta e implementa un mecanismo de excepciones

4.1.4. Atributos asociados al "Comportamiento Temporal (Time Behavior)"

En estos atributos distinguiremos entre los que son apropiados para valores discretos y los que se pueden asociar a un flujo continuo de datos (*data streaming*).

En primer lugar, indicamos los atributos relacionados con el tratamiento de datos discretos.

- **Tiempo de Respuesta (Response Time)**

  Este atributo se asocia a los métodos que implementa un componente en cualquiera de sus interfaces, y evalúa el tiempo transcurrido desde que se recibe la petición, hasta que se emite la respuesta.

  Como este tiempo normalmente dependerá de los parámetros de entrada, será preciso especificar si se trata del tiempo mejor, peor, o medio.

  Este atributo puede asociarse también a interfaces, entendiendo entonces el tiempo peor (resp. mejor, o medio) como el peor (mejor, o medio) de los tiempos de las operaciones que forman parte de la interfaz.

  Se mide con una variable de tipo *Tiempo*.

En segundo lugar, describimos los atributos relacionados con el tratamiento de un flujo continuo de datos:

- **Capacidad en Emisión (Throughput)**

  Este atributo mide la cantidad de información que es capaz de emitir el componente en un intervalo de observación dado.

  Se mide mediante un /emphEntero que indique el número de kilobytes de información por unidad de tiempo.

- **Capacidad en Recepción (Capacity)**

  Este atributo mide la cantidad de información que es capaz de admitir y procesar el componente en un intervalo de observación dado.

  Se mide mediante un /emphEntero que indique el número de kilobytes de información por unidad de tiempo.

4.1.5. Atributos asociados a "Utilización de Recursos (Resource utilization)"

- **Requisitos de Memoria (Memory utilization)**

  La cantidad de memoria que necesita un componente para ejecutarse puede determinar si el componente es adecuado para los productos software donde un requisito importante sea la limitación de memoria.

  Este atributo mide el consumo de recursos respecto a las necesidades de memoria (nº de kilobytes) que necesita el componente para su ejecución. Se puede indicar un tamaño mínimo, máximo o medio (o recomendable, en el sentido de un rendimiento óptimo).

  Se utiliza una variable de tipo *Entero* que indique el número de kilobytes necesarios.

- **Utilización de Disco (Disk utilization)**

  En algún caso específico de diseño de un producto software puede ser necesario limitar el espacio que se utiliza en disco. Por tanto, se mide el tamaño de disco que ocupa el componente de forma estática, y se debe añadir el que ocupa de forma dinámica cuando se invocan sus operaciones.

  Este atributo mide el espacio en disco (nº de kilobytes) que necesita el componente tanto para almacenarse y como para ser utilizadas sus operaciones o servicios.

  Se utiliza una variable de tipo *Entero* que indique el número de kilobytes necesarios.

4.2. Atributos medibles durante el ciclo de vida

Los atributos de calidad para componentes COTS que son observables durante el ciclo de vida del componente se detallan en los siguientes apartados y se muestran resumidos en la tabla 4, agrupados según la sub-característica que miden e indicando el tipo de métrica que utilizan.

4.2.1. Atributos asociados a "Idoneidad (Suitability)"

Un componente será más idóneo cuanto más se ajuste a los requisitos funcionales que necesita o especifica el comprador. Por tanto, se deben medir el número de interfaces que se utilizan frente a las que se ofrecen, es decir, que porcentaje se aprovechan; y cuantas interfaces de las que se necesitan no las suministra el componente.

La dificultad en recoger este tipo de atributo está en que no puede ser medido inicialmente por el proveedor del componente, sino en función de los requisitos de sus potenciales compradores.

- **Cobertura (Coverage)**

  Relaciona el número de interfaces que necesita el usuario del componente con el número de interfaces que ofrece.

Sub-característica	Atributo	Tipo
Corrección	1. Precisión	Ratio
	2. Exactitud computacional	Ratio
Seguridad	3. Cifrado de Datos	Presencial
	4. Capacidad de Control	Presencial
	5. Capacidad para Auditar	Presencial
Recuperabilidad	6. Secuencializable	Presencial
	7. Persistente	Presencial
	8. Transaccional	Presencial
	9. Tratamiento de Errores	Presencial
Comportamiento Temporal	10. Tiempo de Respuesta	Tiempo
	11. Capacidad en Emisión	Entero
	12. Capacidad en Recepción	Entero
Utilización de Recursos	13. Requisitos de Memoria	Entero
	14. Utilización de Disco	Entero

Cuadro 3: Atributos de calidad para componentes COTS medibles durante tiempo de ejecución

Se puede expresar con una métrica de tipo *Ratio* según la siguiente fórmula:

$$100 * \frac{Interfaces\ Necesarias \cap Interfaces\ Ofrecidas}{Interfaces\ Necesarias}$$

– **Exceso (Excess)**

Relaciona el número de interfaces del componente que se van a utilizar en la aplicación con las que este ofrece.

Se puede expresar mediante una métrica de tipo *Ratio* según la siguiente fórmula:

$$100 * \frac{Interfaces\ Utilizadas \cap Interface\ Ofrecidas}{Interfaces\ Ofrecidas}$$

– **Cobertura de la Implementación (Service implementation coverage)**

Es posible que algunas implementaciones de componentes que proveen servicios no cubran el total de los servicios descritos en la especificación. Este atributo pretende medir el número de operaciones que se han implementado respecto al número total de operaciones que se indican en la especificación.

Por ejemplo, muchos proveedores de servicios para plataformas como EJB o CORBA no implementan toda la funcionalidad descrita en la especificación del servicio. Este atributo mide el grado de cobertura de la implementación frente a la especificación.

Se mide con un métrica de tipo *Ratio* según la siguiente fórmula:

$$100 * \frac{Operaciones\ Implementadas}{Operaciones\ Especificadas}$$

4.2.2. Atributos asociados a "Interoperatividad (Interoperability)"

– **Compatibilidad de los datos (Data compatibility)**

Este atributo señala si la representación de los datos de entrada y salida es conforme a algún estándar de facto o internacional. Por ejemplo, si los resultados los produce en un formato propietario, o bien en XML o similar.

Se utilizará una métrica de tipo *Presencial* para indicar si maneja los datos en un formato propietario o estándar y, en su caso, cual.

4.2.3. Atributos asociados a "Conformidad (Compliance)"

– **Conformidad con estándares (Standardization)**

Este atributo indica si el componente cumple algún estándar internacional.

Utilizaremos una métrica de tipo *Presencial*, para indicar si cumple algún estándar y, de ser cierto, qué estándares cumple el componente.

– **Certificaciones (Certification)**

De forma similar al anterior atributo, si la componente puede acreditar algún tipo de certificación, tanto por organizaciones externas o de forma interna, puede indicarlo con este atributo.

Se utiliza una métrica de tipo *Presencial* para indicar si tiene algún tipo de certificación.

4.2.4. Atributos asociados a "*Compatibilidad (Compatibility)*"

– **Compatibilidad hacia atrás (Backwards Compatibilidad)**

Este atributo indica si existe compatibilidad de la versión actual del componente con las versiones anteriores del mismo. Es decir, una nueva versión del componente qué versión anterior puede sustituir sin necesidad de ningún tipo de adaptación.

Se utiliza una métrica de tipo *Presencial*, que permita indicar con qué versión anterior existe compatibilidad.

4.2.5. Atributos asociados a "Madurez (Maturity)"

La madurez de componente se mide en función del número de versiones comerciales que han salido al mercado y en función del tiempo que una versión permanece en activo, es decir, vamos a medir un tiempo medio entre versiones comerciales del componente. A esta atributo lo hemos denominado volatilidad porque da un indicación del tiempo que dura la version en el mercado.

- **Volatilidad (Volatility)**

  Este atributo mide el promedio de tiempo entre cada nueva version comercializada y su versión predecesora. Nos referiremos a el como "Tiempo medio entre versiones".

  Se utiliza una variable de tipo *Tiempo*, que indique el tiempo medio entre versiones.

- **Evolucionabilidad (Evolvability)**

  El número de versiones puede dar una indicación de la madurez de un producto, y de cómo ha ido evolucionando desde que estaba disponible su primera versión. Puede ser interesante en conjunción con la volatilidad para indicarnos si debemos esperar posibles cambios en un periodo corto de tiempo.

  Este atributo se mide mediante un /emphEntero el número absoluto de versiones del componente que han sido distribuidas comercialmente.

- **Fallos Eliminados (Failure removal)**

  Una medida de madurez es el número de errores corregidos en cada versión. Aunque en principio un alto número de errores corregidos puede parecer que estabiliza la nueva versión, la experiencia de los autores en el desarrollo de productos software nos lleva a afirmar que el número de errores corregidos es muchas veces un buen indicador del número de errores que quedan por descubrir.

  Se utiliza una métrica de tipo *Entero* para indicar el número medio de errores eliminados en las últimas versiones.

4.2.6. Atributos asociados a "Facilidad de Aprendizaje (Learnability)"

En relación con la capacidad de ser aprendido, hay un conjunto de atributos relativos al tiempo necesario para poder llevar a cabo una determina función —usar, configurar, etc.— de forma correcta. Estos tiempos, asignados inicialmente por el vendedor o estimados por terceros, dan una indicación de la facilidad de aprendizaje que presenta un componente. Estos tiempos se entenderán como tiempos medios para una persona técnica con un cierto grado de experiencia y preparación, sin ser un experto ni un novato.

- **Periodo para usar correctamente (time to use)**

  Este atributo mide el tiempo medio que necesitará un desarrollador de aplicaciones para poder utilizar de forma correcta el componente.

  Se mide utilizando una métrica de tipo *Tiempo*.

- **Periodo para configurar correctamente (time to configure)**

  Mide el tiempo medio que necesitará un desarrollador para poder configurar de forma adecuada los parámetros que el componente permite particularizar, comprendiendo el significado correcto de los valores que se pueden utilizar.

  Se mide utilizando una métrica de tipo *Tiempo*.

- **Periodo para administrar correctamente (time to admin)**

  Se relaciona con el tiempo medio para poder realizar las operaciones de administración que requiera el componente de forma correcta.

  Se mide utilizando una métrica de tipo *Tiempo*.

- **Periodo para dominar (time to expertise)**

  Se relaciona con el tiempo medio para llegar a dominar de forma precisa la gran mayoría de las posibilidades que ofrece el componente.

  Se mide utilizando una métrica de tipo *Tiempo*.

4.2.7. Atributos asociados a "Facilidad de Comprensión (Understandability)"

Este tipo de atributos se relacionan con las descripciones del componente, las demostraciones o tutorials disponibles y con la comprensión directa de los servicios e interfaces que el componente ofrece.

Es importante señalar que esta característica esta muy relacionada con la "Facilidad de Aprendizaje", ya que, para que un componente puede ser aprendido es necesario que antes sea comprendido por sus posibles usuarios. Por tanto, incluimos aquí los atributos que permiten calificar la facilidad de comprensión del componente que influirán en la facilidad de aprendizaje del mismo por parte del usuario.

– **Documentación de Usuario (User Documentation)**

Este atributo indica la calidad de la documentación suministrado junto con el componente. Al ser una medida subjetiva utilizaremos una métrica con varias categorías.

Se utiliza una métrica de tipo *Nivel* que indique la calidad de la documentación.

– **Sistema de Ayuda (Help System)**

Este atributo indica la calidad del sistema de ayuda asociado con el componente y disponibles por el usuario para localizar información acerca de los servicios e interfaces del componente. Al igual que en el caso de la documentación, se mide asignándole un grado de calidad.

Se utiliza una métrica de tipo *Nivel* que califique la calidad del sistema de ayuda.

– **Documentación computacional (Computer Documentation)**

Este atributo indica la existencia de algún tipo de documentación suministrada con el componente que permita a las herramientas y plataformas de componentes con facilidades reflexivas, la identificación del significado de sus servicios y su posterior invocación de forma dinámica.

Ejemplos de este tipo de documentación serían las descripciones usando UML o MOF (Meta-Object Facilities) del meta-modelo del componente, de sus servicios y de su contexto.

Se utiliza una métrica de tipo *Presencial* que indique si están disponibles este tipo de documentación y, en su caso, en que formato.

– **Formación (Training)**

Si existe alguna forma de realizar un aprendizaje o cursos de formación sobre el uso y configuración del componente que sea ofrecida por el proveedor del mismo, se indicará con este atributo.

Se utiliza una métrica de tipo *Presencial*.

– **Cobertura de la Demostración (Demonstration Coverage)**

Este atributo mide el porcentaje de interfaces (servicios) que están disponibles en una demostración del componente respecto al total de interfaces (servicios) que se suministran.

Se mide con una métrica de tipo *Ratio*, según la siguiente fórmula:

$$100 * \frac{Interfaces\ Demostradas}{Interfaces\ Suministradas}$$

4.2.8. Atributos asociados a "Operatividad (Operability)"

– **Esfuerzo para operar (Effort for operating)**

Este atributo indicará el grado de esfuerzo que se considera necesario para operar con el componente.

Se puede medir utilizando una variable de tipo *Nivel*.

– **Esfuerzo para configurar (Tailorability)**

Este atributo indicará el grado de esfuerzo que se considera necesario para configurar adecuadamente el componente, mediante la particularización de sus parámetros.

Un componente puede tener pocos o muchos parámetros que se puedan modificar —se mide con el atributo "Modificabilidad– pero también se debe tener en cuenta la cantidad de valores diferentes que estos parámetros pueden tomar, así como el significado y las implicaciones que esos valores pueden tener en el comportamiento del componente. Por tanto, se debe dar una idea de la dificultad que esto representa independientemente del número absoluto de parámetros.

Se puede medir utilizando una variable de tipo *Nivel* que indique el esfuerzo necesario para configurar el componente.

– **Esfuerzo para administrar (Administrability)**

Este atributo indicará el grado de esfuerzo que se considera necesario para realizar las operaciones de administración del componente.

Se puede medir utilizando una variable de tipo *Nivel*.

4.2.9. Atributos asociados a "Complejidad (Complexity)"

El objetivo de esta sub-características es dar una medida de la complejidad del uso de un componente y de su integración en un producto o sistema software. Medimos el número total de interfaces que tiene y el número medio de operaciones por interfaz.

– **Interfaces Ofrecidas (Provided Interfaces)**

Medida de la cantidad de interfaces que ofrece el componente como medida indirecta de la complejidad. Cuanto mayor sea este número mayor será la complejidad de uso, y posiblemente, también su complejidad funcional.

Se mide con una variable de tipo /emphEntero.

– **Interfaces Externas Utilizadas (Required Interfaces)**

  Medida de la cantidad de interfaces de otros componentes que necesita el componente para operar. Este atributo da una indicación de la complejidad de la integración de este componente en un sistema, así como el nivel de dependencia con respecto a otros componentes.

  Se mide con una variable de tipo /emphEntero.

– **Índice de Complejidad (Complexity Ratio)**

  Este atributo mide el número medio de operaciones por interfaz ofrecida que presenta el componente.

  Se utiliza una métrica de tipo *Indice* según la fórmula siguiente:

$$\frac{Operaciones\ en\ todas\ las\ Interfaces\ Ofrecidas}{Interfaces\ Ofrecidas}$$

4.2.10. Atributos asociados a "Cambiabilidad (Changeability)"

– **Modificabilidad (Customizability)**

  Este atributo mide el número de parámetros que son modificables por el usuario (diseñador) del componente.

  Se utiliza una métrica con una variable de tipo *Entero* que indique el número de parámetros que ofrece el componente.

– **Índice de Modificabilidad (Customizability Ratio)**

  Este atributo relaciona el número de parámetros que tiene un componente con el número de interfaces ofrecidas del mismo. Esta medida nos proporciona un indicador de la capacidad de modificación del componente. Así, un componente que ofrece pocas interfaces y muchos parámetros normalmente será muy modificable, aunque difícil de manejar, mientras que uno con muchas interfaces y pocos parámetros es poco modificable.

  Se utiliza una variable de tipo de *Índice*, según la siguiente fórmula:

$$\frac{Numero\ de\ Parametros}{Numero\ de\ Interfaces\ Ofrecidas}$$

– **Capacidad de Control del Cambio (Change Control Capability)**

  Este atributo indica la capacidad del usuario para identificar fácilmente la versión utilizada de cada componente.

  Se puede medir utilizando una variable de tipo *Nivel*.

4.2.11. Atributos asociados a "Facilidad de Prueba (Testability)"

Estos atributos indican si el componente suministra algún tipo de prueba que se le pueda realizar de forma independiente y sin tener que integrarlo previamente en una aplicación.

– **Auto-test de Arranque (Start-up self-test)**

  Este atributo se relaciona con la capacidad que puede tener el componente para comprobar su propio estado y del entorno que necesita para ejecutarse correctamente.

  Se utiliza una métrica de tipo *Presencial* que indique si se suministran auto-test de arranque y, en su caso, de que tipo.

– **Batería de pruebas(Tests suite provided)**

  Este atributo indica si el usuario puede disponer de algún tipo de pruebas que se le puedan pasar al componente de forma aislada para comprobar su funcionamiento o realizar medidas, sin tener que integrarlo en la aplicación final.

  Se puede utilizar una variable de tipo *Presencial* que indique si se suministran pruebas para el componente y, en su caso, cuáles y de qué tipo.

5. Documentación de los componentes

Una vez disponemos de un conjunto de atributos de calidad que permiten evaluar componentes COTS, nos planteamos en esta sección los mecanismos más adecuados para documentarlos.

Entre las posibles opciones nos hemos decantado por el modelo de "propiedades" de ODP [9], en el que cada atributo se identifica mediante un par (nombre,valor), y cada nombre de atributo tiene asociado un tipo. Luego, estas propiedades pueden describirse fácilmente mediante plantillas XML, puesto que se trata de un lenguaje neutro e intermedio, y que actualmente es de fácil conexión con todo tipo de herramientas. Más aún, cuando nuestro objetivo es poder utilizar esta información para implementar procesos de selección automática o semi-automática de componentes.

El siguiente ejemplo muestra una descripción del valor de dos atributos en nuestra notación.

```
<property name="Computational Accuracy">
  <type>xsd:ratio</type>
  <value>100</value>
</property>
<property name="Help Documentation">
  <type>xsd:level</type>
  <value>3</value>
</property>
```

Sub-característica	Atributo	Tipo
Idoneidad	1. Cobertura	Ratio
	2. Exceso	Ratio
	3. Cobertura de la Implementación	Ratio
Interoperatividad	4. Compatibilidad de los Datos	Presencial
Conformidad	5. Conformidad con Estándares	Presencial
	6. Certificaciones	Presencial
Compatibilidad	7. Compatibilidad hacia atrás	Presencial
Madurez	8. Volatilidad	Tiempo
	9. Evulocionabilidad	Entero
	10. Fallos Eliminados	Entero
Facilidad de Aprendizaje	11. Periodo para Usar Correctamente	Tiempo
	12. Periodo para Configurar Correctamente	Tiempo
	13. Periodo para Administrar Correctamente	Tiempo
	14. Periodo para Dominar	Tiempo
Facilidad de Comprensión	15. Documentación de Usuario	Nivel
	16. Sistema de Ayuda	Nivel
	17. Documentación Computacional	Presencial
	18. Formación	Presencial
	19. Cobertura de la Demostración	Ratio
Operatividad	20. Esfuerzo para Operar	Nivel
	21. Esfuerzo para Configurar	Nivel
	22. Esfuerzo para Administrar	Nivel
Complejidad	23. Interfaces Ofrecidas	Entero
	24. Interfaces Externas Utilizadas	Entero
	25. Índice de Complejidad	Índice
Cambiabilidad	26. Modificabilidad	Entero
	27. Índice de Modificabilidad	Índice
	28. Capacidad de Control de Cambio	Nivel
Facilidad de Prueba	29. Auto-test de Arranque	Presencial
	30. Batería de Pruebas	Presencial

Cuadro 4: Atributos de calidad para componentes COTS medibles durante el ciclo de vida

Esta notación es además consistente con alguno de los *traders* existentes para componentes COTS [7, 8].

6. Trabajos Relacionados

Hay dos tipos de propuestas relacionadas con el presente trabajo. En primer lugar se encuentran aquellos trabajos que realizan una clasificación de las características de calidad de los productos software, entre los que mencionaremos las normas ISO 9126 [11] e IEEE/ANSI 830-1993 [6], y diferentes propuestas [3, 4, 12, 13] . Quizá la principal diferencia con nuestro trabajo es el ámbito de aplicación, mucho más general en todos ellos, lo que dificulta su utilización de forma efectiva para componentes (más aún en entornos comerciales). En ese sentido nuestra propuesta se restringe a un ámbito muy particular, ciñéndose a las características particulares de los componentes COTS.

Otros trabajos y estándares [1, 5, 10] se centran en el proceso de evaluación de los componentes COTS y de los sistemas desarrollados en base a ellos, pero sin profundizar hasta el nivel de métricas, dejando por tanto sin detallar los atributos que se necesitan medir. En este sentido, nuestro trabajo realiza una aportación intentando concretar dichas métricas.

Parte del trabajo futuro que pretendemos acometer es la fusión de ambos aspectos, los de proceso y los de producto, en torno al modelo de calidad aquí definido. Para ello pretendemos hacer uso de XML, lo que nos va a permitir integrar las herramientas existentes de búsqueda y evaluación de componentes COTS de forma que la mayor parte del proceso se pueda automatizar.

7. Discusión y Conclusiones

En este artículo hemos presentado una particularización del modelo general de calidad de ISO, ajustándolo a los aspectos de calidad específicos de componentes COTS. Asimismo, hemos identificado un conjunto de atributos de calidad que son aplicables a este tipo de componentes, junto con una serie de métricas para evaluarlos. Es importante la relación

que establecemos entre el modelo de calidad y los atributos que definimos, un tema que la mayoría de las propuestas y estándares dejan sin clarificar.

Nuestra motivación principal es la mejora de los procesos de evaluación y selección de componentes COTS para construir con ellos aplicaciones comerciales. Si bien son claras las ventajas que reporta disponer de esas métricas, también somos conscientes de las dificultades que conllevan. En primer lugar, no pensamos que los fabricantes de componentes alcancen fácilmente un consenso sobre los parámetros de calidad que hay que medir en ellos, y menos que estén dispuestos a facilitar toda esa información. Razones comerciales, de marketing y de imagen van a dificultar que parámetros negativos (como el tiempo de fallos, o la ausencia de alguna cualidad importante) sean publicados por los fabricantes. Por otro lado, cada empresa está más interesada en resaltar aquellos atributos en los que sus componentes sean competitivos, restándole importancia a aquellos que no se implementen o para los que no se alcancen buenos valores. Personalmente creemos que esos factores son principalmente los que ocasionan gran parte de las dificultades con las que se está encontrando ISO a la hora de aprobar las normas sobre métricas —aparte de porque sean demasiado genéricas, por supuesto—.

Sin embargo, también pensamos que sin métricas para evaluar componentes no se puede conseguir una verdadera ingeniería del software. Para resolver este conflicto, una posible solución a largo plazo puede basarse en dos factores principales. En primer lugar, no puede haber consenso sobre temas demasiado genéricos. Por ello, es preciso definir y consensuar modelos de calidad para productos muy específicos. Nuestro trabajo pretende ser una primer aportación a este respecto. Y en segundo lugar, quizá la evaluación de la calidad de los componentes (es decir, las medidas que se realizan sobre sus atributos de calidad) debiera hacerse por entidades independientes, al menos hasta que los fabricantes de componentes software adquieran la madurez de los de hardware. Para estos últimos existen catálogos (*data sheet*) que publican los propios fabricantes con los valores de sus atributos de calidad. Hoy en día es todavía una utopía disponer de catálogos así para los componentes software, pero pensamos que es algo hacia lo que habría que tender si realmente queremos hacer una "ingeniería" del software basada en componentes.

Referencias

[1] Chris Abts, Barry W. Boehm, and Elizaberth Bailey Clark. Cocots: A cots software integration lifecycle cost model - model overview and preliminary data collection findings. Available at `sunset.usc.edu/publications/TECHRPTS/2000/usccse2000-501/usccse2000-501%.pdf`, 2000.

[2] Motoei Azuma. Square the next generation of the ISO/IEC 9126 and 14598 international standards series on software product quality. *ESCOM (European Software Control and Metrics conference)*, April 2001. Available at `http://www.escom.co.uk/conference2001/papers/azuma.pdf`.

[3] B.W. Boehm and al. Qualitative evaluation of software quality. In *Proc. 2nd ICSE*, pages 592–605, 1976.

[4] Jan Bosch. *Design & Use of Software Architectures.* Addison Wesley, 2000.

[5] Wilfred J. Hansen. A generic process and terminology for evaluating COTS software. Available at `http://www.sei.cmu.edu/staff/wjh/Qesta.html`, August 2001.

[6] IEEE/ANSI. Recommended practice for software requirements specifications. International Standard 830-1993, IEEE, 1993.

[7] Luis Iribarne, Carina Alves, Jaelson Castro, and Antonio Vallecillo. A non-functional approach for COTS-components trading. In *Proc. of WER 2001*, Buenos Aires, Argentina, November 2001.

[8] Luis Iribarne, José M. Troya, and Antonio Vallecillo. Trading for COTS components in open environments. In *Proc. of the 27th Euromicro Conference*, pages 30–37, Warsaw, Poland, September 2001. IEEE CS Press.

[9] ISO/IEC. RM-ODP. Reference Model for Open Distributed Processing. Rec. ISO/IEC 10746-1 to 10746-4, ITU-T X.901 to X.904, ISO/ITU-T, 1997.

[10] ISO/IEC-14598. Software Engineering - Product evaluation. International Standard ISO/IEC 14598, ISO.

[11] ISO/IEC-9126. Information technology - Software product evaluation - Quality characteristics and guidelines for their use. International Standard ISO/IEC 9126, ISO, December 1991.

[12] James A. McCall, Paul K. Richards, and Gene F. Walters. Factors in software quality, volume III: Preliminary handbook on software quality for an acquisition manager. Technical Report RADC-TR-77-369, vol. III, Hanscom AFB, MA 01731, 1977.

[13] Otto Preiss, Alain Wegmann, and Jason Wong. On quality attribute based software engineering. In *Proc. of the 27th Euromicro Conference*, pages 114–120, Warsaw, Poland, September 2001. IEEE CS Press.

[14] Clemens Szyperski. *Component Software Beyond Object-Oriented Programming.* Addison-Wesley and ACM Press, Boston, 1998.