Aspectos de Calidad en el Desarrollo de Software Basado en Componentes

Manuel F. Bertoa, José M. Troya y Antonio Vallecillo

Depto. Lenguajes y Ciencias de la Computación. Universidad de Málaga.
{bertoa,troya,av}@lcc.uma.es

Resumen

El desarrollo de software basado en componentes se ha convertido actualmente en uno de los mecanismos más efectivos para la construcción de grandes sistemas y aplicaciones de software. Una vez que la mayor parte de los aspectos funcionales de esta disciplina comienzan a estar bien definidos, la atención de la comunidad científica comienza a centrarse en los aspectos extra-funcionales y de calidad, como un paso hacia una verdadera ingeniería. En este artículo se discuten precisamente los aspectos de calidad relativos a los componentes software y a las aplicaciones que con ellos se construyen, con especial énfasis en los estándares internacionales que los definen y regulan, y en los problemas que se plantean en este tipo de entornos.

1 Introducción

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 avance de lo que se conoce como Desarrollo de Software Basado en Componentes (DSBC). Esta nueva disciplina se apoya en componentes software ya desarrollados, que son combinados adecuadamente para satisfacer los requisitos del sistema. Construir una aplicación se convierte por tanto en la búsqueda y ensamblaje 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 apropiados para construir las aplicaciones. En este sentido, la tecnología de componentes ofrece ya soluciones robustas para muchos de los problemas que se plantean en la construcción de grandes sistemas de software y, de hecho, vivimos inmersos en una creciente "componentización" del software.

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 acapara 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 componentes 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 por McCall [McCall, 1977], el modelo de calidad de Boehm [Boehm et al., 1976], los estándares internacionales ISO-9126 [ISO, 1991] e ISO-14598 [ISO, 1998; ISO, 1999], la lista de atributos para la valoración citada para el modelo COCOTS [Abts et al., 2000] basada en estándares de IEEE (en particular el IEEE/ANSI 830-1993), y otras varias [Bosch 2000, Szyperski 1998].

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

Añadido a todo esto, podemos encontrar una ausencia casi total de métricas que permitan dar una estimación más objetiva de estos atributos. Este hecho se ve afectado 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 [Azuma, 2001] es el encargado de tratar de hacerlas converger, 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 suele centrarse en temas de un gran espectro. Por tanto, no suelen estar pensadas para ofrecer soluciones en temas muy concretos.

Nuestro objetivo en este artículo se centra en ofrecer una visión general de los temas de calidad que afectan al desarrollo de software basado en componentes, el estado actual de la normativa internacional, así como las principales propuestas y líneas de investigación en los que se trabaja en este momento. Igualmente, abordaremos los retos más importantes a los que se enfrenta el tratamiento de los aspectos de calidad dentro del DSBC, con vistas a ofrecer una perspectiva general de estos temas, y las líneas de trabajo abiertas.

El documento está estructurado en cinco secciones. Tras esta introducción, la sección 2 hace una breve descripción del DSBC y los componentes COTS, introduciendo el concepto de componente, sus características diferenciadoras, y los principios y procesos que rigen esta disciplina. Después, la sección 3 aborda los aspectos de calidad específicos del DSBC, sus características y la clasificación de atributos de calidad. En la sección 4 se presentan los trabajos actualmente en curso, tanto desde el punto de vista del producto como del proceso. La sección 5 plantea los principales problemas abiertos en el DSBC y plantea algunas líneas de trabajo futuro. Finalmente, la sección 6 presenta las conclusiones de este trabajo.

2 Desarrollo de Software Basado en Componentes (DSBC)

Hoy en día podemos decir que existe un consenso al considerar el desarrollo basado en componentes como un avance significativo hacia la construcción de sistemas mediante el ensamblado de componentes prefabricados. El DSBC trata de sentar las bases para el diseño y desarrollo de aplicaciones distribuidas basadas en componentes software reutilizables. Dicha disciplina cuenta actualmente con un creciente interés, tanto desde el punto de vista

académico como desde la industria, en donde la demanda de mecanismos y herramientas de desarrollo basados en componentes es cada día mayor.

En general, el desarrollo de software basado en componentes puede verse como una extensión natural de la programación orienta a objetos dentro del ámbito de los sistemas abiertos y distribuidos. Este paradigma se basa en el uso de los componentes software como entidades básicas del modelo, entendiendo por componente "una unidad de composición de aplicaciones software que posee un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes, de forma independiente en tiempo y espacio" [Szyperski, 1998].

Numerosas son las características que aporta la programación orientada a componentes frente a la programación orientada a objetos tradicional. Entre ellas, el desarrollo de los componentes de forma independiente del contexto en donde serán ejecutados, la reutilización por composición (frente a herencia, al tratarse de entidades "binarias" o "cajas negras"), la introspección (facilidad para interrogar al componente sobre sus propiedades y métodos de forma dinámica, normalmente mediante el uso de reflexión), nuevas formas de comunicación (como los "eventos" y las comunicaciones asíncronas frente a los rudimentarios mecanismos de los objetos), el enlazado dinámico y composición tardía, la extensión de las IDLs tradicionales, etc. [Szyperski, 1998].

Una de las principales ventajas del desarrollo de software basado en componentes se basa en la "reutilización" de los mismos. De esta forma, los componentes se diseñan y desarrollan con el objetivo de poder ser reutilizados en otras aplicaciones, reduciendo el tiempo de desarrollo, mejorando la fiabilidad del producto final (al usar componentes ya probados previamente), y siendo más competitivos en costes. Aunque hasta ahora la reutilización suele suceder principalmente a nivel interno dentro de las organizaciones, el uso de los componentes comerciales comienza a extenderse. De esta forma se habla de componentes COTS (*Commercial-Off-The-Shelf*), que han sido definidos como una clase especial de componentes software, normalmente de grano grueso, que presentan las siguientes características [Bass et al., 1999]:

- Son vendidos o licenciados al público en general
- Los mantiene y actualiza el propio vendedor, quien conserva los derechos de la propiedad intelectual
- Están disponibles en forma de múltiples copias, todas idénticas entre sí
- Su código no puede ser modificado por el usuario

La cada vez mayor disponibilidad y uso de este tipo de componentes está impulsando notablemente la creación de un mercado global de componentes COTS, que está pasando de ser la utopía de hace unos años a una realidad cada vez más cercana. La tecnología básica de componentes comienza a estar los suficientemente madura (a través de plataformas de objetos distribuidos como EJB, CORBA o .NET) como para que numerosas empresas la adopten en sus nuevos desarrollos y sistemas de información. Incluso el gobierno y el ejército norteamericano han anunciado su uso, y han empezado a apostar por la utilización de componentes comerciales como única vía de mantener sus costes de desarrollo y mantenimiento de software bajo control [Sweeney et al., 2001]. Asimismo, están empezando a proliferar las empresas que venden con éxito componentes software al mercado general, como pueden ser *componentsource*, *flashline*, *wrldcomp*, etc. [Seppanen y Helander, 2001].

Por supuesto, el uso de este tipo de componentes precisa de ciertas modificaciones en los procesos tradicionales de desarrollo de aplicaciones. En particular, los conocidos enfoques *top-down* o *bottom-up* están dejando paso a procesos en espiral [Nuseibeh, 2001] en donde existe una retroalimentación continua entre los requisitos del cliente, el diseño arquitectónico de la aplicación, y los componentes COTS disponibles [Ncube y Maiden, 2001].

En el DSBC, pueden identificarse varias tareas específicas para la construcción de aplicaciones utilizando componentes COTS [Dean y Vigser, 1997]:

(1) La búsqueda (**trading**) de componentes que satisfagan los requisitos impuestos tanto por el cliente como por la arquitectura de la aplicación

(2) La **evaluación** de los componentes candidatos para seleccionar los más idóneos;

(3) La **adaptación** y/o extensión de los componentes seleccionados para que se ajusten a los requisitos anteriores; y

(4) La **integración**, configuración e interconexión de dichos componentes para construir la aplicación final.

Es importante señalar que un factor imprescindible en todas esas tareas es la documentación de los componentes, pues es preciso contar con especificaciones completas, concisas y precisas de los componentes para poder llevarlas a cabo. La mayoría de las propuestas existentes para documentar componentes se basan en el uso de interfaces, los cuales proporcionan un mecanismo para describir la funcionalidad de los componentes. Sin embargo, los lenguajes de descripción de interfaces (IDLs) existentes permiten documentar sólo los aspectos "sintácticos" (o "estáticos") de dicha funcionalidad, sin tratar otros aspectos de vital importancia como son los protocolos de acceso a los servicios, el comportamiento de los métodos, o la semántica de las operaciones de los componentes [Vallecillo et al., 1999]. Por supuesto, dichos IDLs tampoco permiten tratar los aspectos "extra-funcionales" de los componentes, como la fiabilidad, la seguridad, o las prestaciones [Chung et al., 2000; Rosa et al., 2001].

Respecto a los problemas de *trading*, lo que nos encontramos es que solo existen propuestas para la búsqueda de objetos dentro de ciertos modelos (como RM-ODP) o plataformas (CORBA). Sin embargo, son demasiado básicas y simplistas como para poder ser utilizadas a nivel de componentes COTS (una descripción detallada de los problemas que plantean las propuestas tradicionales puede encontrarse en [Iribarne et al., 2001a]). En este sentido, comienzan a aparecer *traders* específicos de componentes, como por ejemplo el denominado "COTStrader" (www.cotstrader.com). Dicho *trader* permite buscar y seleccionar componentes partir de la definición en plantillas XML de su funcionalidad y otras propiedades de calidad [Iribarne et al., 2001a; Iribarne et al., 2001b].

La segunda tarea se centra en los procesos y herramientas para la evaluación y selección de componentes COTS. Aparte de tener en cuenta los requisitos funcionales de la aplicación, es necesario considerar otros factores que también intervienen a la hora de seleccionar componentes. El problema es que este tipo de requisitos, denominados "extra-funcionales" son difíciles de evaluar, aunque todos somos conscientes de la importancia que representan. Este tipo de factores priman muchas veces incluso más que los funcionales, pues un diseñador hasta es capaz de adaptar la arquitectura de un sistema para poder incluir un componente que se desea que esté, o bien para evitar la presencia de un componente de un fabricante en el cual

no se confía. En la introducción ya mencionamos algunas de las causas que intervienen en esa omisión de la valoración en los requisitos extra-funcionales: la no existencia de una definición consensuada de las características o atributos de calidad que hay que medir, la situación de revisión actual de los estándares internacionales relativos a la calidad del producto software (fundamentalmente las normas ISO-9126 e ISO-14598), y la ausencia casi total de métricas que pudieran dar una estimación más objetiva de estos atributos [Sedigh et al., 2001]. Además, es importante señalar que los estándares internacionales proporcionan guías y modelos de calidad para temas muy generales, pero no suelen estar pensados para ofrecer soluciones en temas muy concretos. Igual les sucede a los procesos de selección de productos software, que suelen ser demasiado genéricos para tratar de forma efectiva con la problemática particular de los componentes COTS [Lawlis et al., 2001].

Una vez seleccionados los componentes candidatos, normalmente es preciso realizar una adaptación y/o extensión de los mismos a la hora de integrarlos. En general, al tratar de combinar los componentes suelen aparecer tres tipos de problemas típicos de los ambientes COTS: las lagunas (*gaps*), los solapamientos (*overlappings*), y la incompatibilidad de interfaces (*mismatches*). Los dos primeros aparecen al unir todos los componentes, pues podemos descubrir que alguno de los servicios requeridos por la arquitectura de la aplicación no está cubierto por ninguno de los componentes seleccionados, o bien que algunos componentes ofrecen servicios que se solapan parcialmente. El problema del emparejamiento es de naturaleza distinta, pues surge al tratar de emparejar o conectar componentes cuyas interfaces no son compatibles, y que por tanto deben ser adaptadas como paso previo a su conexión [Garlan et al., 1995].

A pesar de estos problemas, los beneficios que proporciona el DSBC está extendiendo su uso en entornos industriales de desarrollo de software, sobre todo en cuanto a reducción de esfuerzos y costes de desarrollo y mantenimiento, y a la mejora de la calidad final de los productos y sistemas construidos. Esta extensión de su uso está favoreciendo la aparición de numerosas metodologías y herramientas de desarrollo para realizar un efectivo DSBC. Sin embargo, esta disciplina dista aún de ser una verdadera "ingeniería", pues aún no se dispone ni de métricas adecuadas, ni de procesos precisos, ni de normativa internacional que la regule. En este sentido, la definición de métricas de calidad para componentes y aplicaciones software, especialmente las desarrolladas en base a componentes COTS, aparece como una necesidad imperiosa.

3 Calidad en el Desarrollo Software Basado en Componentes

La palabra calidad tiene varios significados, aunque dentro de la Ingeniería del Software podemos adoptar la definición de la norma ISO-8402, que luego se repite en otras (por ejemplo en ISO-14598): "La totalidad de aspectos y características de un producto o servicio que tienen que ver con su habilidad para satisfacer las necesidades declaradas o implícitas".

Aunque la calidad es un objetivo importante para cualquier producto, no debemos olvidar que los productos, y también los productos software, se construyen para ser utilizados. Por tanto, el principal objetivo de un producto es satisfacer una necesidad (o varias) de un usuario y, por consiguiente, ofrecer al usuario algún beneficio por su utilización. Es decir, la calidad no es el objetivo último del producto, sino satisfacer las necesidades de un cliente. También es importante señalar que esto implica que la calidad de un producto software no se puede referir únicamente a obtener un producto sin errores. La especificación de la calidad del

software debe ser más detallada y exacta, y el camino para ello es la formalización de la calidad mediante un *modelo de calidad* que, como veremos a continuación, define las características de un producto que influyen a la hora de medir su calidad.

3.1 Características de Calidad en 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, utilizaremos los estándares internacionales, fundamentalmente el ISO-9126. Aunque actualmente en proceso de revisión, ha sido el primero en definir y concretar este tipo de características.

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. Una característica se puede refinar en múltiples niveles de *sub-características*. Ejemplos de características son la funcionalidad, la fiabilidad o la facilidad de uso. A su vez, la característica funcionalidad se puede descomponer en sub-características como corrección, interoperatividad y seguridad, entre otras.

Llamaremos *atributo* a una propiedad de calidad a la que puede asignársele una *métrica*, entendiendo por métrica 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 son medibles (p.e. la "demostrabilidad").

Con todo esto, un *modelo de calidad* se define como "un conjunto de características y sub-características, junto con las relaciones que existen entre ellas". 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, válidos para grandes familias de productos. Sin embargo, su alto grado de generalización y abstracción hacen difícil su aplicación a temas concretos. También, es importante tratar de "refinar" esos modelos generales, particularizándolos a casos concretos. Por ello, resultaría de gran interés y utilidad poder contar con un modelo de calidad particularizado para componentes. Desde nuestro punto de vista, dicho modelo de calidad ha de ser simple y concreto si queremos que pueda ser utilizado con ciertas garantías de éxito.

Uno de los principales objetivos de tal modelo de calidad para componentes es el de detectar los atributos que 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.

3.2 Clasificación de los atributos de calidad

Para los componentes, y tendiendo en cuenta como posible objetivo la definición de un modelo de calidad específico, es fundamental primero realizar una taxonomía, tratando de clasificar las características de calidad de acuerdo a su naturaleza y a distintos parámetros que intervienen en su medida. Se pueden realizar 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).

3. Como se menciona en los estándares de ISO, es importante identificar los usuarios a los que se dirige el modelo. En el ámbito del DSBC, 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.

3.3 Otras características

Por último, es importante señalar 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.

Todas estas características técnicas y no técnicas deben estar documentadas en un formato aceptado por los participantes en el desarrollo de software. Una forma adecuada de implementar la documentación de componentes es utilizar el modelo de "propiedades" de ODP [RM-ODP 1997] donde cada atributo se describe mediante un par (nombre, valor), con un tipo asociado a cada nombre. Estas propiedades se pueden describir mediante plantillas XML, lo cual facilita la conexión con cualquier tipo de herramientas. Además, esta forma de documentar los componentes abre la posibilidad de implementar procesos de selección automática o semiautomática, facilitando el proceso de valoración de los componentes software.

Otro tema, muy relacionado con este, es el de la Calidad de Servicio (QoS). En su normativa, ISO define la QoS como "un conjunto de calidades relacionadas con el comportamiento colectivo de uno o más objetos". En este sentido es importante señalar que fundamentalmente este tipo de atributos van a coincidir con los atributos de calidad que

pueden ser medidos durante la ejecución del sistema o de los componentes. Sin embargo, suelen ser más al nivel de sistema que de componentes, pues la QoS se mide con respecto al comportamiento global del sistema y a los resultados que éste produce. Por tanto, no es suficiente con disponer de los valores de los atributos de calidad de los componentes de un sistema, sino también influye la forma en la que están conectados, el medio de transporte que utilizan para comunicarse, la complejidad de los protocolos de interacción entre ellos, etc. En general este es un tema complejo y en el cual se trabaja de forma muy activa [Pal et al., 2000; Cukier et al., 1998; Schmitdt et al., 1998; Zinky et al., 1997].

4 Propuestas de Calidad en el Desarrollo Software Basado en Componentes

En este apartado vamos a comentar las principales propuestas relativas a la calidad en el desarrollo de software basado en componentes en las que actualmente se trabaja. Las dividiremos en dos categorías: por un lado, las que presentan propuestas de calidad para el producto y, por otro, las que lo hacen respecto al proceso.

4.1 Calidad en el producto

La primera referencia a la calidad de un componente COTS, visto como un producto software, la tenemos que hacer a los estándares ISO-9126 e ISO-14598 ya comentados anteriormente. La importancia de estos estándares reside en que aportan la idea y necesidad de un *modelo de calidad*, que en nuestro caso, se debe particularizar para componentes. En concreto, la norma ISO-9126 define un modelo general de calidad basado en seis características principales (funcionalidad, fiabilidad, facilidad de uso, eficiencia, mantenibilidad y portabilidad), que a su vez se refinan en 21 sub-características [ISO, 1991]. Aunque este modelo de calidad es bastante completo, presenta dos problemas principales. El primero es la falta de precisión en la definición de tales características, mientras que el segundo se debe a que no todos esas características y sub-características son aplicables a componentes software [Hansen, 2001; Bertoa y Vallecillo, 2002]. Por tanto, pensamos que ese modelo se muestra idóneo como base de partida general, pero que es necesario refinarlo al caso particular de los componentes, al igual que se hace para otros desarrollos, como pueden ser las páginas Web u otros.

Por otro lado, desde el punto de vista de la arquitectura software y dado la gran relación que con ella tienen los componentes, es destacable el tratamiento que da Bass [Bass et al., 1999] a los atributos de calidad dentro de la arquitectura software. Su trabajo presenta una idea muy interesante y que puede ser aplicada a los componentes, al proponer una clasificación de los atributos de calidad en función de si tienen relación con la arquitectura software o no. Desde el punto de vista de los componentes podemos pensar si un atributo está relacionado con un componente como entidad independiente, o bien, está relacionado con la arquitectura software. Además, en su propuesta Bass divide los atributos de calidad en los que son aplicables al sistema en ejecución (p.e. Rendimiento, Usabilidad); los que son aplicables a las tareas de desarrollo o mantenimiento, es decir, durante el ciclo de vida (p.e. Mantenibilidad, Portabilidad); los que tienen relación con el entorno comercial del producto (p.e. Coste, Tiempo de vida previsto del sistema); y, los que se aplican a la arquitectura software directamente (p.e. Integridad Conceptual, Construibilidad).

Otro trabajo de gran interés en el ámbito de la arquitectura software es el de Jan Bosch [Bosch 2000]. Su propuesta incluye la valoración de los requisitos de calidad para una

arquitectura, y no sólo la valoración de los requisitos funcionales. Estos requisitos de calidad se deben valorar durante la fase de diseño de la arquitectura software.

Bosch muestra la dificultad de especificar con detalle los requisitos de calidad, pero sí encuentra que los requisitos más importantes en la mayoría de las propuestas existentes presentan alguna forma de lo que denomina perfil (*profile*). Un perfil es un conjunto de escenarios, generalmente con alguna relativa importancia relacionada con cada escenario. Por ejemplo, el *perfil de uso* es un conjunto de escenarios que describen la utilización típica del sistema software. Otros perfiles posibles son el perfil de cambios o el perfil de riesgos. Basándose en esta idea, propone perfiles de los atributos de calidad y selecciona cinco atributos de calidad como los más relevantes desde una perspectiva de ingeniería de sistemas software general. Estos atributos son: Rendimiento, Mantenibilidad, Fiabilidad, Seguridad Física y Seguridad de Acceso.

Por último, dentro de las propuestas relativas al producto, Preiss [Preiss et al., 2001] presenta también en su trabajo una clasificación de los atributos de calidad de los componentes. Por atributo de calidad entienden el grupo de propiedades que aparecen habitualmente en la literatura con el sufijo "-bilidad" ("ilities" o "nesses" en inglés). De nuevo, establece la distinción de que estos atributos se pueden clasificar en dos categorías: los que son observables durante tiempo de ejecución (p.e. seguridad) y los que son discernibles durante todo el ciclo de vida del componente (p.e. interoperatividad). Los atributos observables en tiempo de ejecución son referidos, en esta propuesta, como los atributos de Calidad de Servicio (QoS) de los componentes.

De esta forma, Preiss propone que la calidad de un sistema software se puede valorar mediante un conjunto de atributos de calidad. A la mayoría de estos atributos los podemos considerar "sistémicos", es decir, son aplicables a un sistema software completo o están extendidos por una parte de él. Además, es importante tener en cuenta que la decisión de qué conjunto de atributos de calidad son los más importantes va a depender del punto de vista desde donde miremos el sistema. No es lo mismo el punto de vista de un usuario que el de un desarrollador, o que el de la persona encargada de su administración y mantenimiento.

Basándose en la propuesta de Bass, Preiss et al. también tratan de clasificar los atributos de calidad entre los que son observables en tiempo de ejecución y los que son observables durante el ciclo de vida del producto. Además, se intenta definir una categoría principal y una subcategoría de atributos siguiendo la idea de la norma ISO-9126, para poder simplificar así las referencias a un conjunto de atributos relacionados.

Según estos autores, los atributos QoS se refieren a un conjunto de atributos de calidad extra-funcionales que son discernibles durante la ejecución del sistema y que pueden ser vinculados con un *caso de uso* particular. En ese contexto, representan calidades de comportamiento (*qualities of behavior*), lo cual es muy coherente con la definición que RM-ODP da para la QoS.

4.2 Calidad en el proceso

El proceso de construcción de un producto software basado en componentes adquiere unas especiales características como se ha mencionado anteriormente. Entre los trabajos que tratan de forma directa el proceso de desarrollo basado en componentes nos parecen de especial interés los que comentamos a continuación.

4.2.1 COCOTS

En primer lugar, Boehm y sus colaboradores han extendido al ámbito de los componentes COTS el modelo general COCOMO II [Boehm et al., 1995] de estimación del coste (y esfuerzo) de un proyecto software. Este nuevo modelo para componentes se ha denominado COCOTS (*Constructive COTS*) [Abts et al., 2000]. El modelo se compone de cuatro submodelos que recogen las cuatro principales fuentes de los costes de integración de componentes COTS:

- Valoración (*Assessment*) es el proceso mediante el cual los componentes COTS son seleccionados para ser integrados en el sistema que se está desarrollando.

- Adaptación (*Tailoring*) se refiere a las actividades que siempre hay que realizar para particularizar el componente, independientemente del producto donde se va a integrar. Por ejemplo, inicializar los valores de los parámetros, especificar pantallas de E/S o formato de los informes, configurar protocolos de seguridad, etc.

- Código de Integración (*Glue Code*) es el código nuevo que hay que desarrollar y probar, externo a los COTS, pero necesario para integrar los componentes COTS en un sistema.

- Finalmente, la volatilidad (*Volatility*) tiene en cuenta la frecuencia con la cual aparecen en el mercado nuevas versiones o actualizaciones de los componentes COTS que se están utilizando en el sistema durante su desarrollo y puesta en funcionamiento.

Durante la fase de Valoración es cuando se deben poner de manifiesto las características de calidad de los componente COTS, idealmente valoradas cuantitativamente de forma imparcial y de forma que permitan facilitar el proceso de selección de los componentes. Aquí es precisamente donde entran los atributos de calidad. Sin embargo, una de las principales lagunas de este modelo es que no se llega a definir ningún conjunto de atributos de calidad (es decir, un modelo de calidad), lo cual limita bastante su utilización práctica

4.2.2 QESTA

Otro trabajo de gran interés relativo a los temas de calidad en DSBC es el presentado por Hansen [Hansen 2001], en el cual plantea un proceso de evaluación para seleccionar el mejor candidato de los posibles componentes que se puedan utilizar. Este proceso se denomina QESTA: "Quantification, Examination, Specification, Transformation, Aggregation".

En primer lugar, y refiriéndose a la terminología utilizada habitualmente por distintos autores, plantea la necesidad de tener una terminología única. Palabras como, atributo, característica, condición, criterio, propiedad, calidad y otras se utilizan de forma intercambiable en las distintas propuestas existentes en la literatura, e incluso las definiciones que tiene ISO actualmente no están exentas de ambigüedades e inconsistencias. Por ejemplo, ISO se queda a medio camino al olvidar definir los términos más básicos en los que basa la mayoría de sus definiciones. Por todo ello, Hansen utiliza los términos *métricas* y *transformaciones*, por los que entiende funciones que producen un valor numérico de una clase específica.

El núcleo del proceso de evaluación QESTA se muestra en la siguiente tabla, y está basado en un conjunto de "operaciones" básicas, que son las que van a definir un proceso que

permita realizar una evaluación efectiva de componentes software. A su vez, cada operación viene determinada por sus datos de entrada, sus datos de salida, y una descripción de su objetivo y comportamiento básico:

Datos Entrada	*Operaciones*	*Datos Salida*
Context, Qualities	**Quantification**	Metrics
Entities, Metrics	**Examination**	Values
Metrics	**Specification**	Transforms
Transforms, Values	**Transformation**	Factors
Factors	**Aggregation**	Decision Data

– Cuantificación: Generalmente, las calidades que se desean para un sistema no son directamente medibles sino que provienen de afirmaciones vagas del tipo "pequeño tamaño", "rendimiento aceptable" o "alta disponibilidad". Este paso define para cada calidad requerida una o más *métricas*, donde una métrica es un procedimiento para examinar una entidad y producir un dato simple, un número o un símbolo.

– Examen: En este paso las entidades candidatas seleccionadas son medidas con las métricas seleccionadas en el paso anterior. El resultado es un conjunto de Valores, uno para cada combinación de entidad y métrica.

– Especificación: Una vez que están definidas las métricas, los que tienen que tomar las decisiones deben ser consultados para que interpreten juiciosamente los valores obtenidos. Por ejemplo, sobre si un "Sí " es un valor aceptable o inaceptable. O bien, si se trata de un valor necesario y su ausencia descalifica la entidad bajo medida. Tomando como base estos juicios de valor, es preciso definir una transformación (*transform*) para cada métrica de tal forma que cada valor obtenido sea convertido en un valor adecuado para el paso de agregación posterior.

– Transformación: En este paso, las transformaciones desarrolladas en el paso de especificación son aplicadas a los valores obtenidos en el paso de examen para producir factores (*factors*). Uno de los propósitos de este paso es alinear todos los valores en rangos comparables, por ejemplo entre –1 y +1. Sin embargo, las transformaciones más importantes son las que pueden rechazar una entidad por el hecho de no tener el valor requerido para alguna métrica.

– Agregación: El objetivo de este último paso es producir una decisión final. Existen muchos métodos de agregación, cada uno de ellos con una formalidad matemática distinta. Por ejemplo, un método común consiste en asignar pesos a los factores obtenidos en el paso anterior, aunque se ha visto que este método puede no ser el más apropiado en todos los casos, y por tanto sería preciso buscar métodos de agregación alternativos.

Esta propuesta tampoco concreta los atributos de calidad (o "métricas" en su terminología) que se deben medir dentro del proceso QESTA. Esto limita su aplicación

inmediata, aunque posiblemente esta sea la propuesta que presenta el proceso de evaluación mejor especificado.

4.2.3 COTS-Based Requirements Engineering (CRE)

Las propuestas anteriores inciden en la fase de valoración de los componentes candidatos, pero sin entrar en detalle de los atributos de calidad que pueden usarse de forma efectiva para medir un componente. Existen otras propuestas de interés en el ámbito de la selección de componentes que queremos comentar.

El método CRE propuesto por Alves [Alves y Castro, 2001] se desarrolla para facilitar el proceso de selección y evaluación basado en los requisitos, prestando una especial atención al análisis de los requisitos extra-funcionales. La selección de componentes se realiza por *rechazo*, es decir, los componentes candidatos que no cumplen alguno de los requisitos del cliente van siendo rechazados y retirados de la lista de candidatos. Conforme esta lista va decreciendo, es necesario aumentar el número y el detalle de los requisitos. El resultado es un proceso iterativo mediante el cual el proceso de adquisición de los requisitos permite la selección de productos y, adicionalmente, el propio proceso de selección genera información para la obtención de nuevos requisitos de calidad.

CRE es un método orientado a objetivos, es decir, cada fase se orienta hacia conseguir unos objetivos predefinidos. Para ello, cada fase tiene unas plantillas que incluyen guías y técnicas para la adquisición y/o modelado de requisitos y para la evaluación de productos.

El método tiene cuatro fases iterativas:

- Identificación. La primera tarea del proceso de identificación es la definición de objetivos. Esto debe estar basado en el análisis detallado de los factores que influyen.. Los criterios de evaluación deberían ser desarrollados en función de estos factores
- Descripción. Durante esta fase, los criterios de evaluación se elaboran con mayor detalle, tomando mayor importancia los requisitos extra-funcionales como discriminadores entre productos similares. Aunque se reconoce la dificultad de obtener, expresar, cuantificar y probar este tipo de requisitos.[Cysneiros y Leite, 1999]
- Evaluación. Esta fase usa las técnicas apropiadas para el análisis de los datos de evaluación con el fin de facilitar la toma de decisiones. La decisión de seleccionar un producto COTS determinado está basada en el coste estimado versus el análisis de beneficios para cada alternativa COTS. CRE sugiere el uso de un modelo de coste como el propuesto en COCOTS.
- Aceptación. Esta fase concierne a la negociación de los contratos legales con los suministradores de componentes para el producto final. En esta fase se deben resolver los aspectos legales pertinentes a la compra de los productos o de su licencia.

4.2.4 Off-The-Shelf Option (OTSO)

El método OTSO desarrollado por Kontio [Kontio et al, 1995] establece un proceso de selección de "paquetes" de software reutilizables, denominados por los autores como OTS (Off-The-Shelf) pues incluyen tanto componentes comerciales (COTS) como componentes desarrollados internamente por las propias organizaciones. El método OTSO facilita la búsqueda, evaluación y selección de software reutilizable. Además, proporciona técnicas específicas para la definición de criterios de evaluación, comparando los beneficios y costes

de cada una de las alternativas posibles, y consolidando los resultados de la evaluación que faciliten la toma de decisiones [Kontio et al.,1996].

Este método define varias actividades o fases –definición, búsqueda, filtrado, evaluación y análisis– donde se van refinando los criterios de evaluación conforme avanza el proceso de selección. La fase de *búsqueda* (*search*) tiene como meta identificar los potenciales candidatos que posteriormente serán examinados. La fase de *filtrado* (*screening*) selecciona los candidatos más prometedores que pasarán a la fase *evaluación* detallada. En la fase de *análisis*, se consolidan los resultados de las evaluaciones de los productos y se toma una decisión sobre la reutilización o no de algún producto software. La definición de criterios de evaluación es la actividad central del proceso que alimenta a las otras fases y, a su vez, se realimenta de los resultados de ellas.

OTSO se centra en aportar un conjunto de criterios de evaluación. Para ello, al ser un método dirigido por objetivos, propone una serie de relaciones entre los factores de influencia, los objetivos de reutilización y los criterios de evaluación. La primera tarea importante en una evaluación de software reutilizable es definir los *objetivos de reutilización*. Estos objetivos deben basarse en un análisis cuidadoso de los factores que influyen en la selección. En este sentido, una aportación interesante que realiza este método es la clasificación en cinco grupos de los factores de influencia en la selección de COTS:

- Requisitos de Usuarios
- Arquitectura de la aplicación
- Restricciones y objetivos del proyecto
- Disponibilidad de productos
- Infraestructura de la organización

A partir de estos factores, y debido a que cada contexto o situación en donde se desea reutilizar el software es distinto, el método propone que se deben definir unos objetivos de reutilización asociados a cada caso particular. Dichos objetivos pueden dividirse en: objetivos para el proceso de desarrollo, para el proceso de mantenimiento, y para las características del producto.

Aunque el método OTSO resalta que el problema clave en la selección de componentes COTS es la falta de atención a los requisitos de calidad, el método no proporciona ni sugiere una solución efectiva. Además, asume que los requisitos ya existen, puesto parte de la especificación de requisitos para la interpretación de los mismos.

4.3 Procurement-Oriented Requirements Engineering (PORE)

El método PORE [Ncube y Maiden, 1999] es una propuesta basada en plantillas (*templates*) para facilitar la captura de requisitos. Utiliza un proceso iterativo de captura de requisitos y de selección y evaluación de productos.

El ciclo de vida del modelo de proceso PORE tiene seis procesos genéricos. Estos procesos son definidos a tres niveles de acuerdo a la clasificación propuesta en [Humphrey,89]: Nivel Universal, Nivel Mundial y Nivel atómico.

Por ejemplo, los seis procesos genéricos que PORE define a nivel atómico (aunque no sea necesario realizarlos todos en cada adquisición (*procurement*) de producto), son los siguientes:

- Gestión del sistema: Los objetivos de este proceso son la planificación y el control del proceso de obtención, para poder satisfacer las necesidades del comprador en fecha y con un coste razonable. Este proceso se desarrolla durante todo el ciclo de vida del método, simultáneamente al resto de procesos.
- Adquisición de requisitos: Este proceso obtiene y valida los requisitos de calidad del usuario. Además, determina la arquitectura del sistema para que los nuevos componentes puedan integrarse en él.
- Selección de suministradores: Sus objetivos son establecer los criterios de selección de los suministradores, evaluarlos, ordenarlos de acuerdo a esos criterios de calidad y seleccionar el que mejor se ajuste a ellos.
- Selección de paquetes/componentes software: El objetivo de este proceso es identificar los paquetes (componentes) candidatos, establecer los criterios de selección usando los requisitos del cliente, evaluar los componentes identificados, ordenarlos de acuerdo a los criterios y seleccionar uno o más de entre los que mejor cumplan el núcleo de requisitos esenciales del cliente.
- Contrato de producción: Su objetivo es negociar los contratos legales con los suministradores de los componentes software y resolver los aspectos legales relativos a la compra y licencia de los mismos.
- Aceptación del paquete/componente: El objetivo de este proceso es comprobar que el paquete, componente o sistema comercializado cumple con el núcleo original de requisitos esenciales del cliente.

Otra característica de este método es que propone cuatro objetivos esenciales para la selección o el rechazo de los productos, los cuales se integran en una plantilla para facilitar la labor del evaluador. De esta forma, PORE rechaza un producto de acuerdo a la conformidad o no con los siguientes objetivos de calidad:

- Requisitos del cliente atómicos esenciales (Objetivo 1);
- Requisitos del cliente atómicos no esenciales (Objetivo 2);
- Requisitos del cliente no atómicos complejos (Objetivo 3);
- Requisitos de usuario del cliente (Objetivo 4);

La principal limitación de esta propuesta, que es también común a la mayoría de las que existen al nivel de proceso, es que se centra solamente en el proceso de evaluación de los componentes, pero sin llegar a detallar los atributos concretos que se han de medir, o las métricas a ser utilizadas.

5 Problemas y líneas de trabajo abiertas

Una vez que hemos repasado las principales propuestas de trabajo actualmente en marcha en los temas de calidad relativos al DSBC, en esta sección trataremos de discutir algunos de los principales retos y problemas abiertos de este área de trabajo.

En primer lugar, hemos destacado anteriormente la importancia que tiene para el desarrollo basado en componentes el poder contar con un modelo de calidad particular para componentes software (como por ejemplo, una particularización del modelo general de calidad de ISO), que tenga especial cuidado con los aspectos de calidad específicos de componentes COTS. Asimismo, se debería también identificar un conjunto de atributos de calidad que sean aplicables a este tipo de componentes, junto con una serie de métricas para evaluarlos. Es importante establecer una relación entre el modelo de calidad y los atributos que se definan, un tema que la mayoría de las propuestas y estándares dejan sin clarificar. Este sería por tanto, desde nuestro punto de vista, el primer objetivo a abordar, y que ya comienza a ser tratado en algunos trabajos [Bertoa y Vallecillo, 2002].

La definición de atributos y métricas para componentes, dentro de un modelo de calidad, tiene como objetivo la mejora de los procesos de evaluación y selección de componentes COTS en la construcción con ellos de aplicaciones comerciales. Si bien son claras las ventajas que reporta disponer de esas métricas, también somos conscientes de las dificultades que esto conlleva. 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–, y el problema es que dichas dificultades tienen una difícil solución por el marcado carácter comercial de sus motivaciones.

Otro de los aspectos a resolver es el tema de la documentación de los componentes. Como mencionamos anteriormente, es preciso contar con especificaciones completas, concisas y precisas de los componentes para poder llevar a cabo los procesos identificación, evaluación, y selección de los mismos, de acuerdo a los requisitos de los usuarios (tanto funcionales como de calidad). Quizá XML sea el lenguaje apropiado, pero XML sólo proporciona la sintaxis. Por tanto, es necesario también definir la semántica y las "ontologías" apropiadas para la documentación de componentes, así como desarrollar las herramientas que hagan uso de estas plantillas para la implementación de los procesos adecuados de búsqueda, selección y evaluación de componentes. Una de las primeras propuestas se describe en [Iribarne et al, 2001a], en donde se definen plantillas XML para la descripción de componentes, documentando los distintos aspectos que intervienen a la hora de evaluarlos: su funcionalidad (interfaces y nombres de métodos, semántica operacional y protocolos de accesos a sus servicios), sus requisitos de calidad y extra-funcionales, los aspectos técnicos de distribución y empaquetamiento (tipo de plataforma en la que corren, cómo instalarlos, requisitos arquitectónicos y dependencias del sistema, etc.), y finalmente los aspectos de marketing y comerciales (vendedor, precio de las licencias, acuerdos de uso, garantía, contratos de mantenimiento, etc.). Es importante señalar que ese trabajo proporciona los mecanismos y herramientas para la definición de tales parámetros y su posterior utilización para el proceso de selección (*trading*). Sin embargo, no se definen los parámetros concretos que deben de aparecer en esa documentación (p.e. un modelo de calidad concreto que defina los atributos que deben ser descritos), lo que queda pendiente para futuros trabajos.

Finalmente, estamos asistiendo al nacimiento de los WebServices como una forma de facilitar el acceso a las aplicaciones a través de Internet. Pero no sólo eso, sino que permiten llevar el adjetivo "COTS" a los servicios, más allá de los componentes. En general, los WebServices ofrecen importantes ventajas, como son el acceso a los servicios desde cualquier sitio en la red (*pervasiveness*), permiten simplificar el acceso y facilitar la interoperabilidad, y también posibilitan pasar de las típicas aplicaciones Web, que son *2-tier*, a aplicaciones más complejas, que requieren una arquitectura *n-tier*. Y por último, suponen un gran avance frente a los componentes desde un punto de vista del marketing, pues permiten realizar un alquiler de servicios externos frente al desarrollo o a la compra de componentes. Quizá el ámbito de aplicación actual de los WebServices sea el comercio electrónico, pero sin duda ocuparán un nivel similar a los componentes software COTS en un futuro muy cercano. De ahí que sea necesario también comenzar a trabajar en modelos de calidad para este tipo de servicios, además de para los componentes COTS.

6 Conclusiones

En este trabajo hemos analizado el estado actual de la principales propuestas que abordan los aspectos de calidad en el desarrollo de software basado en componentes, y discutido los principales retos y problemas abiertos en este campo.

Como hemos puesto de manifiesto, no existe actualmente un tratamiento lo suficientemente exhaustivo y consolidado de estos temas de calidad, a pesar de su gran importancia. Probablemente esto sea debido tanto a la reciente implantación de la tecnología de componentes software y al incipiente uso industrial de los componentes COTS, como a la fuertes implicaciones comerciales y de marketing que conllevan los aspectos de calidad en los componentes software.

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 ideas 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. Y en segundo lugar, quizás 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 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

[Abts et al., 2000] C. Abts, B.W. Boehm and E.B. Clark. "COCOTS: A COTS Software Integration Lifecycle Cost Model - Model Overview and Preliminary Data Collection Findings", Informe Técnico de la Universidad de Carolina del Sur, 2000. http://sunset.usc.edu/publications/TECHRPTS/2000/usccse2000-501/usccse2000-501.pdf

[Alves y Castro, 2001] C. Alves and J. Castro. "Um Método Baseado em Requisitos para Seleção de COTS". 4th Iberoamerican Workshop on Software Engineering and Software Environment (IDEAS'01) San Jose, Costa Rica, April 2001.

[Azuma, 2001] M. Azuma "SquaRE: the next generation of the ISO/IEC 9126 and 14598 international standards series on software product quality". In ESCOM (European Software Control and Metrics conference), April 2001.

[Bass et al., 1999] L. Bass and P. Clements y R. Kazman. *Software Architecture in Practice*. 6ª edición. Addison-Wesley, 1999.

[Bertoa y Vallecillo, 2002] M. F. Bertoa and A. Vallecillo. "Atributos de Calidad para Componentes COTS". In Proc. of IDEAS 2002, pp. 352-363. La Habana, Cuba, April 2002.

[Boehm et al., 1976] B.W. Boehm et al. "Qualitative evaluation of software quality". In Proc. of ICSE'76, pp. 592-605, 1976.

[Boehm et al., 1995] B.W. Boehm B. Clark, E. Horowitz, R. Madachy, R. Shelby y C. Westland. "Cost Models for Future Software Lifecycle Processes: COCOMO 2.0". Annals of Software Engineering, 1995.

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

[Chung et al., 2000] Chung, L., Nixon, B., Yu, E., and Mylopoulos, J., "Non-Functional Requirements in Software Engineering". Kluwer Academic Publisher, 2000.

[Cukier et al., 1998] M. Cukier, J. Ren, C. Sabnis, D. Henke, J. Pistole, W. Sanders, D. Bakken, M. Berman, D. Karr, and R.E. Schantz. "Aqua: An adative architecture that provides dependable distributed objects". In Proc. of the 17th IEEE Symposium on Reliable Distributed Systems, pages 245-253, October, 1998.

[Cysneiros y Leite, 1999] L. Cysneiros, and J.C.S.P. Leite. "Integrating Non-Functional Requirements into Data Modeling". In Proc. of the 4th IEEE Requirements Engineering Symposium, Ireland, June 1999.

[Dean y Vigser, 1997] J. Dean and M.R. Vigser. "System Implementation Using Off-the-shelf Software". In Proceedings of the 9th Annual Software Technology Conference, April 1997.

[Garlan et al., 1995] D. Garlan, R. Allen and J. Ockerbloom. "Architectural Mismatch: Why Reuse is So Hard". IEEE Software. 12(6):17-26, November, 1995.

[Hansen 2001] W. Hansen. "A Generic Process and Terminology for Evaluating COTS Software". Disponible en www.sei.cmu.edu/staff/wjh/Qesta.html, 2001.

[Humphrey, 1989] W.S. Humphrey. "Managing the Software Process". Addison-Wesley, 1989.

[Iribarne et al., 2001a] L. Iribarne, J.M. Troya and A. Vallecillo. "Trading for COTS components in Open Environments". In Proc. of the 27th Euromicro Conference, pp. 30-37, Warsaw, Polonia, September 2001. IEEE Press.

[Iribarne et al., 2001b] L. Iribarne, A. Vallecillo, C. Alves and J. Castro. "A non-functional approach for COTS-components trading". In Proc. of WER 2001, pages 124-138. Buenos Aires, Argentina, November 2001.

[ISO, 1991] ISO/IEC-9126:1991. "Information Technology – Software Product Evaluation – Quality Characteristics and Guideline for their Use" December 1991.

[ISO, 1998] ISO/IEC-14598-5:1998. "Information Technology – Software Product Evaluation – Part 5 Process for evaluators". July 1998.

[ISO, 1999] ISO/IEC-14598-1:1999. "Information Technology – Software Product Evaluation – Part 1 General Overview". April 1999.

[Kontio et al., 1995] J. Kontio, S. Chen, K. Limperos, R. Tesoreiro, G. Caldeira, and M.S. Deutsch, "A COTS Selection Method and Experience of Its Use". In Proceeding of the 20th annual Software Engineering Workshop. NASA. Greenbelt, Maryland. November 1995.

[Kontio et al., 1996] J. Kontio, G. Caldeira, and V.R. Basili, "Defining Factors, Goals and Criteria for Reusable Component Evaluation". CASCON'96 conference, Toronto, Canada. November 1996.

[Lawlis et al., 2001] P.K. Lawlis, K.E. Mark, D.A. Thomas and T. Courtheyn. "A Formal Process for Evaluating COTS Software Products". IEEE Computer, 34(5):58-63, May 2001.

[McCall et al., 1977] J.A. McCall, P.K. Richards and G.F. Walters. "Factors in software quality, volume III: Preliminary handbook on software quality for an acquisition manager". Informe técnico RADC-TR-77-369, vol. III, Hanscom AFB, MA 01731, 1977.

[Ncube y Maiden, 1999] C. Ncube and N. Maiden. "M PORE: Procurement-Oriented Requirements Engineering Method for the Component-Based Systems Engineering Development Paradigm". International Workshop on Component-Based Software Engineering, May 1999.

[Ncube y Maiden, 2000] C. Ncube and N. Maiden "COTS Software Selection: The Need to Make Tradeoffs between System Requirements, Architectures and COTS Components". In COTS workshop: Continuing Collaborations for Successful COTS Development, 2000.

[Nuseibeh, 2001] B. Nuseibeh. "Weaving Together Requirements and Architectures". IEEE Computer, 34(3):115-117, March 2001.

[Pal et al., 2000] P. Pal, J. Loyall, R. Schantz, J. Zinky, R. Shapiro, and J. Megquier. "Using QDL to Specify QoS Aware Distributed (QuO) Application Configuration", In Proc. of the 3rd International Symposium on OO R-Time Distr. Computing (ISORC'00). March, 2000.

[Preiss et al., 2001] O. Preiss, A. Wegmann, y J. Wong. "On Quality Attribute Based Software Engineering". In Proc. of the 27th Euromicro Conference, Warsaw, Poland, September 2001. IEEE CS Press.

[RM-ODP 1997] 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.

[Rosa et al, 2001] N. Rosa, C. Alves, P. Cunha, J. Castro and G. Justo. "Using Non-Functional Requirements to Select Components: A Formal Approach." In Proceedings of IDEAS'01, San José, Costa Rica. April 2001.

[Sedigh et al., 2001] S. Sedigh-Ali, A. Ghafoor and R.A. Paul. "Software Engineering Metrics for COTS-based Systems". IEEE Computer, 34(5):44-50, May 2001.

[Seppanen y Helander, 2001] V. Seppanen and N. Helander. "Original Software Component Manufacturing: Survey of the State-of-the-Practice". 27th Euromicro Conference, pp. 138-145. Varsovia, Polonia, September 2001.

[Schmidt et al., 1998] "The design of the TAO real-time object request broker". Computer Communications Special Issue on Building Quality of Service into Distributed Systems, 21(4), April 1998.

[Sweeney et al., 2001] L.E. Sweeney, E.V. Kortright and R.J. Buckley. "Developing an RM-ODP-based architecture for the defense integrated military human resources system". J. Cordeiro and H. Kilov (eds.) Proceedings of WOODPECKER'01, pp. 110-124. Setúbal, Portugal, July 2001.

[Szyperski, 1998] C. Szyperski. Component Software. Beyond Object-Oriented Programming. Addison-Wesley Longman, 1998.

[Vallecillo et al.,1999] A. Vallecillo, J. Hernández, and J.M. Troya. "Object Interoperability." In ECOOP'99 Workshop Reader, LNCS 1743, pp. 1-21. Springer-Verlag, 1999.

[Zinky et al., 1997] J.A. Zinky, D.E. Bakken, and R.E. Schantz. "Architectural support for quality of service for CORBA objects". Theory and Practice of Objects Systems, 1(3):55-73, April 1997.