ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE INGENIERÍA

USO DE COMPONENTES COMERCIALES COTS EN EL DESARROLLO DE APLICACIONES DE SOFTWARE

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

MARÍA GABRIELA URQUÍA SIGCHA

DIRECTOR: Msc. RAÚL CÓRDOVA

Quito, Diciembre 2006

DECLARACIÓN

Yo, María Gabriela Urquía Sigcha, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado por ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la siguiente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por su normativa institucional vigente.

María Gabriela Urquía Sigcha

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por María Gabriela Urquía Sigcha, bajo mi supervisión.

Msc. Raúl Córdova

DIRECTOR DE PROYECTO

DEDICATORIA

A mi abuelita Hortencia, que aunque ausente físicamente, ha estado conmigo siempre.

A mi tía Olguita, mi segunda madre, por todo su amor y dedicación con mi familia.

A mis padres, Manuel y Piedad, por darme la vida, pero sobretodo por su amor y apoyo infinito.

A mi hermana Salomé, por ser un ejemplo a seguir y por apoyarme incondicionalmente en todo.

A mi hermana Fátima, por su amor, su alegría y su apoyo incondicional.

Este logro está dedicado a ustedes con todo mi amor!

AGRADECIMIENTOS

A quien ha estado conmigo todos los días de mi vida, siempre poniéndome pruebas para superarlas pero sobretodo regalándome infinitas bendiciones, Gracias Diosito por todo lo que me has obsequiado.

Al regalo más grande que me dio Dios, mi familia, gracias por sus enseñanzas, su sacrificio, su confianza y apoyo… Los amo!

Al Ingeniero Raúl Córdova, porque más que un maestro, ha sido un gran amigo. Gracias por toda su ayuda.

Al Ingeniero Ángel Porras, por todo su apoyo, su confianza y su enseñanza.

A mis amigas de toda la vida: Eliana, Gaby G., Vanesa y Gaby J., que han estado siempre conmigo. Gracias por su compañía, sus consejos, y su apoyo.

A todos mis amigos y compañeros de aula, de trabajo y de la vida, porque de todos ustedes he aprendido lecciones inolvidables… Gracias!
.

Gaby

INDICE

CAPITULO 1

INDICE DE FIGURAS

CAPITULO 1

CAPITULO 2

CAPITULO 3

INDICE DE TABLAS

CAPITULO 1

CAPITULO 2

CAPITULO 3

INTRODUCCIÓN

Los continuos avances en la Informática y las Telecomunicaciones están provocando cambios en la forma en la que se desarrollan actualmente las aplicaciones de software. En particular, el incesante aumento de la potencia de los computadores, el abaratamiento de los costos del hardware y las comunicaciones y la aparición de redes de datos de cobertura global han disparado el uso de los sistemas abiertos y distribuidos. Esto ha provocado, entre otras cosas, que los modelos de programación existentes se vean desbordados, siendo incapaces de manejar de forma natural la complejidad de los requisitos que se les exigen para este tipo de sistemas.

En muchas ocasiones las tecnologías tradicionales como la Orientada a Objetos, no son suficientes para solucionar los problemas planteados por el desarrollo de sistemas distribuidos y abiertos. Se hace pues necesaria la adopción de nuevas tecnologías de desarrollo de software. El Desarrollo de Software basado en Componentes (DSBC), proporciona una nueva forma de crear aplicaciones mediante la composición de componentes de software independientes. Esta tecnología promueve la utilización de los componentes como elementos básicos en la construcción de las aplicaciones y su reutilización en diferentes contextos

En el ámbito de la Ingeniería de Software Basada en Componentes, se está presenciando un mayor interés por el uso de los componentes comerciales, también conocidos como componentes “off--the-shelf" o componentes COTS. El uso de esta clase de componentes puede tener significativas ventajas, como reducir sus costos, esfuerzos y tiempo de construcción, a la vez que aumenta la fiabilidad y flexibilidad del producto final.

No obstante, para sacar provecho de estas ventajas también es necesario que los componentes comerciales cumplan ciertas garantías de fiabilidad relacionadas con aspectos como la calidad de servicio que ofrece el componente, o la calidad de la información funcional y no funcional del componente, que luego los desarrolladores del software pueden consultar para comprobar si dicho componente cumple con las exigencias de los componentes especificados en la arquitectura de software de la aplicación.

Por estas razones y debido en parte al rápido crecimiento de las técnicas y tecnologías relacionadas con los componentes comerciales, se hace necesaria la existencia de pautas o mecanismos que ayuden a los ingenieros en las tareas de construcción de aplicaciones de software a partir de componentes COTS. En el presente trabajo se muestran los mecanismos para realizar las tareas de definición, búsqueda, selección e integración de componentes comerciales en el desarrollo de aplicaciones de software.

CAPITULO 1 INGENIERÍA DEL SOFTWARE BASADA EN COMPONENTES COTS

El desarrollo de software basado en componentes (DSBC o CBD[1]), es una aproximación del desarrollo de software que describe, construye y utiliza técnicas de software para la elaboración de sistemas abiertos y distribuidos mediante el ensamblaje de partes de software reutilizables. El DSBC es utilizado para reducir los costos, tiempos y esfuerzos en el desarrollo del software, a la vez que ayuda a mejorar la fiabilidad, flexibilidad y la reutilización de la aplicación final.

Durante algunos años, el DSBC fue referido como una filosofía conocida como “compre, y no construya" promulgada por Fred Brooks en 1987[2] y que abogaba por la utilización de componentes prefabricados sin tener que desarrollarlos de nuevo. En la actualidad, muchos autores que trabajan en el campo de DSBC, como Heineman y Councill o Wallnau, entre otros, empiezan a reconocer y a aceptar el uso de estándares, guías, procesos y prácticas de ingeniería, sin los cuales el desarrollo de software basado en componentes sería solamente una mezcla entre competitivos y confusos lenguajes, metodologías y procesos. Estos estándares, guías, procesos y prácticas han propiciado que se empiece a hablar del término de “Ingeniería del Software Basada en Componentes" (ISBC o CBSE[3]), como una subdisciplina de la “Ingeniería del Software".

1.1 COMPONENTES

1.1.1 HISTORIA

El auge de los “sistemas distribuidos" coincide justo con la época del “boom de Internet", y con ello, un cambio radical en las metodologías de desarrollo de los sistemas. En pocos años se pasa de una mentalidad centralizada, donde prevalecía la confidencialidad y

[1] En inglés se conoce como CBD (Component-Based Development).
[2] Brooks, F. No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer. Pag:10-14.
[3] En inglés se conoce como CBSE (Component-Based Software Engineering).

sistemas basados en Intranet, a una mentalidad totalmente opuesta, descentralizada y basada en Internet.

A mediados de los años 80 empiezan a converger diversos factores en el mundo de la informática, y que serían el detonante de un cambio en el proceso de ingeniería de los sistemas informáticos. La masiva presencia de equipos de bajo costo (PCs), permitió a los ingenieros desarrollar grandes aplicaciones desglosadas en módulos de software que podían estar ubicados en distintos ordenadores, propiciando un nuevo enfoque en el desarrollo de los sistemas.

Inicialmente, estos bloques de software funcionaban como elementos de cómputo independientes dentro del sistema, pero pronto, los ingenieros vieron la necesidad de disponer de nuevas técnicas para la comunicación y transferencia de los datos entre estos elementos de cómputo. Precisamente por esta fecha, y ajeno a estas necesidades, empezaban a consolidarse fuertes líneas de investigación en computación paralela y programación concurrente que junto con las necesidades de comunicación de procesos en ambientes de cómputo independientes, dieron lugar a los primeros esfuerzos en la elaboración de una nueva tecnología para la programación distribuida de aplicaciones. Precisamente, uno de los primeros resultados fue el desarrollo de la técnica RPC (Remote Procedure Call). Esta técnica permite que los desarrolladores de software puedan diseñar sus aplicaciones mediante módulos de software comunicantes, como si fuesen un conjunto de procesos cooperativos independientes.

Debido a la rápida utilización de la técnica RPC, se empezó a dar forma a todo un entorno de computación distribuida sin la elaboración de un marco teórico que lo sustentase. Esto propició la aparición del primer modelo de distribución en 1994, conocido con el nombre de DCE (Distributed Computation Environment)[4]. Este modelo fue desarrollado por OSF (Open Systems Foundation), una organización formada por IBM, DEC y Hewlett-Packard. El modelo establecía las pautas y normas que los ingenieros debían seguir para desarrollar sus sistemas.

[4] OSF DCE Application Development Guide. Cambridge, MA. http://www.opengroup.org.

Esta nueva mentalidad de construir aplicaciones divididas en partes comunicantes y residentes en distintos ambientes de cómputo fue un gran paso en el campo programación distribuida. Esto dio paso a la aparición de nuevos conceptos, como los “sistemas heredados" (legacy systems), que hace referencia a la integración de partes de software existentes con las del sistema actual. Otro concepto es el de “envolvente" (wrapper), que son porciones de código especialmente diseñados para encapsular y dar funcionalidad a otras partes del sistema ya existentes. O el concepto de “código de enlace" (glue), que son porciones de código cuyo efecto es similar al de un “pegamento" y que sirve para unir distintas partes funcionando con “envolventes".

Pero el concepto más importante que ha cambiado y sigue cambiando los procesos de ingeniería y reingeniería, es el concepto de “componente". Con el término componente se empiezan a diferenciar dos estilos de desarrollo de software. Por un lado está el estilo de desarrollo de software basado en reutilización, donde las aplicaciones se construyen a partir de otras partes de software ya existentes y accesibles en repositorios conocidos. Por otro lado está el desarrollo de software de reutilización, donde se ponen en práctica procesos de ingeniería para la elaboración de partes eficientes de software que luego pueden ser utilizadas para la construcción de aplicaciones (en el otro estilo de desarrollo de software). A estas partes de software se las conoce como componentes de software, y han dado lugar a los paradigmas de programación de componentes top-down o descendente (para reutilizar) y bottom-up o ascendente (basado en reutilización).

Pero el uso generalizado de los componentes en procesos de ingeniería de software realmente empieza a tomar presencia y sentido con la aparición de nuevos modelos de distribución, como CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model) o EJB (Enterprise Java Bean), modelos que actualmente se están utilizando para el desarrollo de aplicaciones distribuidas. Su predecesor, el modelo DCE, empieza a ser visto por los ingenieros de sistemas como un modelo difícil y costoso de llevar a la práctica.

1.1.2 DEFINICIÓN DE COMPONENTE

Desde el punto de vista de la ingeniería del software, el término “componente" procede de las “técnicas orientadas a objetos", de los problemas de descomposición usados en “técnicas de descomposición de problemas", y de su necesidad para desarrollar sistemas abiertos. Con la aparición de los modelos de componentes COM (Component Object Model), EJB y CCM (CORBA Component Model), en pocos años ha ido emergiendo una práctica de desarrollo basada en componentes. Sin embargo, su expansión se ha visto retrasada por la falta de acuerdo entre los especialistas, a la hora de dar una definición concreta sobre lo que es y no es un componente de software.

Existe una gran diversidad de opiniones sobre lo que debe ser un componente de software. Esto se pone de manifiesto, por ejemplo, en el artículo “What characterizes a (software) component?"[5], donde se hace una variada recopilación de definiciones de componente y ofrece una discusión sobre lo que caracteriza a un componente de software. Uno de los factores que impide una definición concreta del término se debe a la falta de un acuerdo común sobre cuales son las características y propiedades que lo diferencian de un objeto. En algunos casos, incluso, se llega a utilizar de forma indistinta los términos componente y objeto.

Una de las definiciones de componente software más extendidas es la de Clemens Szyperski, y que dice lo siguiente:

“Un componente es una unidad binaria de composición de aplicaciones software, que posee un conjunto de interfaces y 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.” [6]

[5] Broy, M., Deimel, A., Henn, J., Koskimies, K., Plásil, F., Pomberger, G., Pree, W., Stal, M., y Szyperski, C. What characterizes a (software) component? Software - Concepts and Tools. Pags: 49-56.
[6] Szyperski, C. Component Software. Beyond Object-Oriented Programming. Pag: 33.

Según Clemens Szyperski, las nociones de “instanciación”, “identidad” y “encapsulación” son más propias de los objetos que de los componentes, y define un objeto como: “Una unidad de instanciación que tiene una única identidad, un estado que puede ser persistente, y que encapsula su estado y comportamiento”. Sin embargo, un componente puede contener múltiples objetos, clases y otros componentes.

La noción de componente puede variar dependiendo del nivel de detalle desde donde se mire, conocido como “granularidad de un componente”. Un componente software puede ser desde una subrutina de una librería matemática, hasta una clase en Java, un paquete en Ada, un objeto COM, un JavaBeans, o incluso una aplicación que pueda ser usada por otra aplicación por medio de una interfaz especificada. Un componente con granularidad gruesa se refiere a que puede estar compuesto por un conjunto de componentes, o ser una aplicación para construir otras aplicaciones o sistemas a gran escala, generalmente abiertos y distribuidos. A medida que se desciende en el nivel de detalle, se dice que un componente es de granularidad fina. Un ejemplo de componentes de granularidad gruesa son los componentes Advanced Components de la arquitectura WSBC (WebSphere Business Components) de IBM. En WSBC se define un componente de la siguiente manera:

“Una implementación que (a) realiza un conjunto de funciones relacionadas, (b) puede ser independientemente desarrollado, entregado e instalado, (c) tiene un conjunto de interfaces para los servicios proporcionados y otro para los servicios requeridos, (d) permite tener acceso a los datos y al comportamiento sólo a través de sus interfaces, (e) opcionalmente admite una configuración controlada.”[7]

Otra definición de componente es la que adopta el SEI (Software Engineering Institute), y que dice lo siguiente:
“Un componente software es un fragmento de un sistema software que puede ser ensamblado con otros fragmentos para formar piezas más grandes o aplicaciones completas.”[8]

[7] IBM-WebSphere Large Grained Components. http://www.ibm.com/software/components.
[8] Brown, A. Constructing Superior Software, capítulo Building Systems from Pieces: Principles and Practice of Component-based Software Engineering. Pag: 102

Esta definición se basa en tres perspectivas de un componente: (a) la perspectiva de empaquetamiento (packaging perspective) que considera un componente como una unidad de empaquetamiento, distribución o de entrega. Algunos ejemplos de componente de esta perspectiva son los archivos, documentos, directorios, librerías de clases, ejecutables, o archivos DLL, entre otros; (b) la perspectiva de servicio (service perspective) que considera un componente como un proveedor de servicios. Ejemplos son los servicios de bases de datos, las librerías de funciones, o clases COM, entre otros; (c) la perspectiva de integridad (integrity perspective) que considera un componente como un elemento encapsulado, como por ejemplo una base de datos, un sistema operativo, un control ActiveX, una applet de Java, o cualquier aplicación en general.

Los autores Cheesman y Daniels, identifican diferentes “visiones” en las que puede aparecer el término componente en las etapas de desarrollo de un sistema software. Concretamente se identifican hasta cinco formas de componente, mostrados en la Figura 1.1.

En primer lugar está la “especificación de componente", que representa la especificación de una unidad software que describe el comportamiento de un conjunto de “objetos componente" y define una unidad de implementación. El comportamiento se define por un conjunto de interfaces. Una especificación de componente es “realizada” como una “implementación de componente”.

En segundo lugar está la “interfaz de componente", que es una definición de un conjunto de comportamientos (normalmente operaciones) que pueden ser ofrecidos por un objeto componente.

En tercer lugar está la “implementación de componente", que es una realización de una especificación de componente que puede ser implantada, instalada y reemplazada de forma independiente en uno o más archivos y puede depender de otros componentes.

En cuarto lugar está el “componente instalado”, que es una copia instalada de una implementación de componente.

Y en quinto y último lugar está el “objeto componente”, que es una instancia de un “componente instalado”. Es un objeto con su propio estado e identidad única y que lleva a cabo el comportamiento implementado. Un “componente instalado" puede tener múltiples “objetos componente” o uno solo.

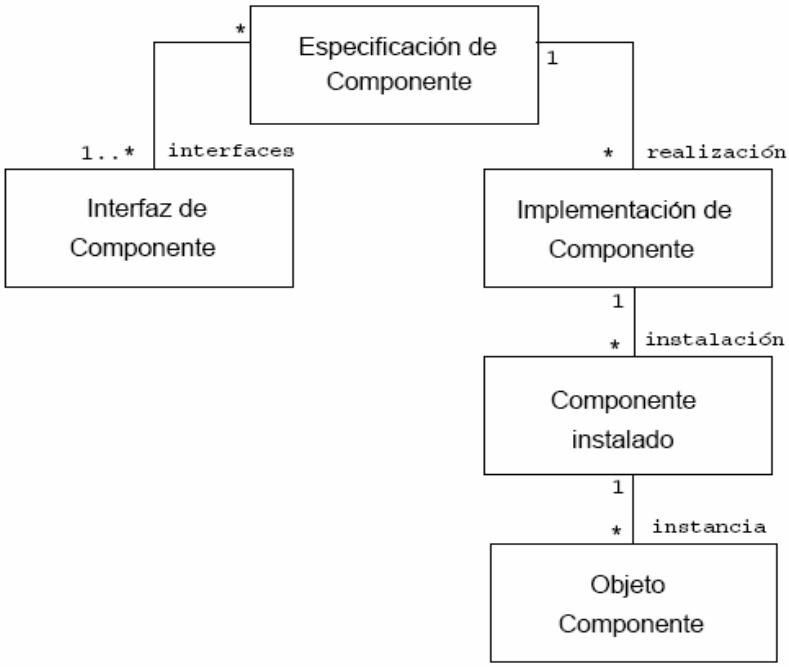

Figura 1 1 Formas en las que puede presentarse el término componente

Para finalizar está la visión de componente EDOC (Enterprise Distributed Object Computing) del OMG (Object Management Group). EDOC es una especificación para la computación de objetos distribuidos que se ajusta al modelo de la arquitectura de objetos de OMG denominado OMA (Object Management Architecture)[9]. La definición de componente que se ofrece en este ambiente dice simplemente lo siguiente:

“Un componente es algo que se puede componer junto con otras partes para formar una composición o ensamblaje.”

Como conclusión de estas definiciones se pueden hacer las siguientes valoraciones. Los componentes son partes de software que se pueden combinar con otros componentes para generar un conjunto aún mayor (por ejemplo otro componente, subsistema o sistema). Un

[9] OMG: http://www.omg.org

componente juega el papel de una unidad software reutilizable que puede interoperar con otros módulos de software mediante sus interfaces. Un componente define una o más interfaces desde donde se puede tener acceso a los servicios que éste ofrece a los demás componentes.

Un componente puede presentarse en forma de código fuente o código objeto; puede estar escrito en un lenguaje funcional, procedural o en un lenguaje orientado a objetos; y puede ser tan simple como un botón GUI o tan complejo como un subsistema.

1.2 COMPONENTES COMERCIALES COTS

El término COTS, como sucede con muchos otros términos en el campo de la informática (como por ejemplo Internet), surge desde el Ministerio de Defensa de los Estados Unidos. Históricamente hablando, el término COTS se remonta al primer lustro de los años 90, cuando en junio de 1994 el Secretario de Defensa americano, William Perry, ordenó hacer el máximo uso posible de especificaciones y estándares comerciales en la adquisición de productos (hardware y software) para el Ministerio de Defensa. En Noviembre de 1994, el Vicesecretario de Defensa para la Adquisición y Tecnología, Paul Kaminski, ordenó utilizar estándares y especificaciones de sistemas abiertos como una norma extendida para la adquisición de sistemas electrónicos de defensa. A partir de entonces, los términos "comercial", "sistemas abiertos", "estándar" y "especificación" han estado muy ligados entre sí (aunque un término no implica los otros), estando muy presentes en estos últimos años en ISBC.

El término "componente comercial" puede ser referido de muy diversas formas, como por ejemplo, software *Commercial Off-The-Shelf (COTS)*, o *Non-Developmental Item (NDI)*, o incluso *Modifiable Off-The-Shelf (MOTS)*. En realidad existen unas pequeñas diferencias entre ellos, diferencias que se reflejan en la Tabla 1.1. En cualquier caso, se hará referencia a las tres categorías como componentes COTS:

CATEGORÍA	DESCRIPCIÓN
COTS	Software que (a) existe a priori, posiblemente en repositorios; (b) está

	disponible al público en general; y (c) puede ser comprado o alquilado.
NDI	Software desarrollado (inicialmente sin un interés comercial) por organizaciones para cubrir ciertas necesidades internas, y que puede ser requerido por otras organizaciones. Por tanto es un software que (a) existe también a priori, aunque no necesariamente en repositorios conocidos; (b) está disponible, aunque no necesariamente al público en general; y (c) puede ser adquirido, aunque más bien por contrato.
MOTS	Un tipo de software Off-The-Shelf donde se permite tener acceso a una parte del código del componente, a diferencia del componente COTS, cuya naturaleza es de caja negra, adquirido en formato binario, y sin tener posibilidad de acceder al código fuente.

Tabla 1. 1 Tipos de software comercial

Como sucede para el caso de los componentes de software, tampoco existe una definición concreta y comúnmente usada para el término COTS. Una definición híbrida del término "componente COTS" se puede encontrar utilizando, por un lado, la definición de "componente" de Szyperski, y por otro lado la definición de elemento "COTS" del SEI, que dice lo siguiente:

"Un elemento COTS se refiere a un tipo particular de componente software, probado y validado, caracterizado por ser una entidad comercial, normalmente de granularidad gruesa y que reside en repositorios de software, y que es adquirido mediante compra o alquiler con licencia, para ser probado, validado e integrado por usuarios de sistemas."[10]

Existen otros autores, que también consideran que un componente comercial no tiene necesariamente que ser adquirido mediante compra o licencia, sino que también puede ser adquirido como software de dominio público o desarrollado fuera de la organización. Según estas perspectivas, se entiende por componente COTS lo siguiente:

Un componente COTS es una unidad de elemento de software en formato binario, utilizada para la composición de sistemas de software basados en componentes, que generalmente es de granularidad gruesa, que puede ser adquirido mediante compra, licencia, o ser un

[10] Brown, A. Constructing Superior Software, capítulo Building Systems from Pieces: Principles and Practice of Component-based Software Engineering. Pag: 137

software de dominio público, y con una especificación bien definida que reside en repositorios conocidos.

1.2.1 CARÁCTERÍSTICAS DE UN COMPONENTE COMERCIAL

Por regla general, existe una gran diversidad de parámetros que caracterizan a un componente COTS, pero sin embargo, dos son los más comunes. En primer lugar, un componente COTS suele ser de granularidad gruesa y de naturaleza de “caja negra" sin posibilidad de ser modificado o tener acceso al código fuente. Una de las ventajas de un software comercial es precisamente que se desarrolla con la idea de que va a ser aceptado como es, sin permitir modificaciones. Hay algunos desarrolladores de componentes que permiten la posibilidad de soportar técnicas de personalización que no requieren una modificación del código fuente, por ejemplo mediante el uso de plug-ins y scripts. Y en segundo lugar, un componente COTS puede ser instalado en distintos lugares y por distintas organizaciones, sin que ninguna de ellas tenga el completo control sobre la evolución del componente software. Es sólo el vendedor de componentes COTS quien decide su evolución y venta.

Son muy numerosas las ventajas - aunque también lo son los inconvenientes – de utilizar componentes COTS en lugar de componentes de “fabricación propia”.

Una de las ventajas más claras es el factor económico, relacionado con el costo de desarrollo. Puede ser mucho más barato comprar un producto comercial, donde el costo de desarrollo ha sido amortizado por muchos clientes, que intentar desarrollar una nueva “pieza” de software. Por otro lado, el hecho de que un componente COTS haya sido probado y validado por el vendedor y por otros usuarios del componente en el mercado, suele hacer que sea aceptado como un producto mejor diseñado y fiable que los componentes construidos por uno mismo.

Otra ventaja es que el uso de un producto comercial permite integrar nuevas tecnologías y nuevos estándares más fácilmente y rápidamente que si se construye por la propia organización.

En cuanto a las desventajas, principalmente se destacan dos, aunque estas derivan en otras más. En primer lugar, los desarrolladores que han adquirido un componente comercial no tienen posibilidad de acceso al código fuente para modificar la funcionalidad del componente. Esto significa que en las fases de análisis, diseño, implementación y pruebas, el componente es tratado como un componente de caja negra, y esto puede acarrear ciertos inconvenientes para el desarrollador, como por ejemplo no saber cómo detectar y proceder en caso de fallos; o que el sistema requiera un nivel de seguridad no disponible en el componente, entre otros problemas. Además, los productos comerciales están en continua evolución, incorporando el fabricante nuevas mejoras al producto y ofreciéndoselo a sus clientes (por contrato, licencia o libre distribución). Sin embargo, de cara al cliente desarrollador, reemplazar un componente por uno actualizado puede ser una tarea laboriosa e intensiva: el componente y el sistema deben pasar de nuevo unas pruebas (en el lado cliente).

En segundo lugar, otra gran desventaja es que, por regla general, los componentes COTS no suelen tener asociados ninguna especificación de sus interfaces, ni de comportamiento, de los protocolos de interacción con otros componentes, de los atributos de calidad de servicio, y otras características que lo identifiquen. En algunos casos, las especificaciones que ofrece el fabricante de componentes COTS puede que no sean siempre correctas, o que sean incompletas, o que no sigan una forma estándar para escribirlas.

Otras veces, aunque el vendedor de componentes COTS proporcione una descripción funcional del componente, puede que ésta no satisfaga las necesidades del integrador, y que necesite conocer más detalles de la especificación del comportamiento y de los requisitos del componente. Además, la falta de una información de especificación puede acarrear ciertos problemas al desarrollador que utiliza el componente COTS, como por ejemplo la imposibilidad de estudiar la compatibilidad, la interoperabilidad o la trazabilidad de los componentes durante el desarrollo del sistema basado en componentes.

En la actualidad son muchos los autores que proclaman la necesidad de un modelo para la especificación de componentes COTS utilizando diferentes notaciones y estrategias. Estas propuestas estudian el tipo de información básica que debe ser capturada en una plantilla de especificación de componente COTS; pero son muy pocas las propuestas existentes que están soportadas por herramientas, y probablemente ninguna de ellas es ampliamente aceptada por la industria para documentar componentes de software comerciales.

1.3 SISTEMAS BASADOS EN COMPONENTES COTS

Desde hace tiempo, la reutilización de software ha venido siendo una práctica común para la construcción de productos software. La reducción de los costos, tiempos y esfuerzos en los procesos de elaboración han sido algunos de los motivos que han llevado a los ingenieros de software a considerar técnicas para la reutilización de partes software en prácticamente cualquier fase del ciclo de vida del producto (análisis, diseño e implementación). Estas partes de software, generalmente, se corresponden con fragmentos de código, procedimientos, librerías y programas desarrollados en otros proyectos dentro de la organización, y que pueden ser utilizados de nuevo para ser incorporados en ciertas partes del nuevo producto que hay que desarrollar.

Además, en estos últimos años se ha comprobado un aumento en el uso de componentes comerciales en prácticas de reutilización de software. Concretamente, estos componentes comerciales, están siendo considerados con mayor frecuencia para la construcción de sistemas complejos, distribuidos y abiertos. Para la elaboración de estos sistemas, los ingenieros utilizan metodologías, procesos y técnicas de desarrollo basados en componentes (DSBC). El sistema a desarrollar estaría compuesto por una o más aplicaciones de software, que pueden ser consideradas o no como componentes. Incluso puede que algunas de estas aplicaciones software hayan sido construidas mediante la composición de otras partes de software (componentes) durante el desarrollo del sistema. Estas partes software ensambladas pueden ser obtenidas de muy diversas formas:

a) Desarrolladas: En este caso los componentes son construidos directamente por la organización dentro del proyecto de desarrollo del sistema, y todavía sigue siendo (aunque cada vez menos) la práctica habitual y tradicional en la elaboración de un producto de software.

b) Reutilizadas: Otra posibilidad es que los componentes hayan sido reutilizados a partir de repositorios, propios a la organización, que contienen código (fuente y ejecutable) bien definido, desarrollado en otros proyectos de la organización, y que pueden ser utilizados en nuevos proyectos. Sin embargo, esta no es una práctica muy

habitual dentro de las organizaciones, ya que por regla general no se disponen de estos repositorios internos con partes software (componentes) con especificaciones bien definidas, y en ocasiones hay que aplicar prácticas de reingeniería inversa, donde a partir del código y de la documentación comentada dentro del código (si lo hay) se intenta extraer sus características de funcionamiento para comprender su comportamiento.

c) Adquiridas: Una última posibilidad consiste en adquirir el componente software a través de terceras partes, en lugar de construirlo, reduciendo con ello el tiempo, costo y esfuerzo que conlleva el desarrollo del componente. No obstante, esto supone también disponer de repositorios y catálogos de componentes comerciales conocidos, con especificaciones bien definidas y estandarizadas, con adecuadas técnicas y procesos de búsqueda, selección y evaluación de componentes: en definitiva, disponer de un mercado de componentes comerciales consolidado. Desafortunadamente, todo esto no sucede en la realidad, aunque es cierto que en el área de ISBC existen grandes esfuerzos de investigación en estos últimos años para llegar a conseguir estos objetivos.

1.3.1 PROPIEDADES DE UN SISTEMA BASADO EN COMPONENTES COTS

En base a las actividades de desarrollo mostradas en la Tabla 2, un sistema de software basado en componentes COTS puede tener las siguientes propiedades deseables:

1.3.1.1 Adaptable

Las actualizaciones de la configuración de un componente es una tarea frecuente. Puesto que algunos fabricantes de componentes COTS generan nuevas versiones a lo largo de un año, el proceso de reemplazar componentes puede tener lugar varias veces al año para cada componente COTS del sistema. Para reducir este esfuerzo, un sistema debería tener una configuración de componentes adaptable que permita a los componentes una fácil incorporación, eliminación o sean reemplazados.

1.3.1.2 Auditable

Un sistema es auditable si los integradores y gestores son capaces de ver y monitorizar su comportamiento interno. La auditoría es crítica en tareas de prueba, monitorización y detección de errores del sistema. El software COTS puede o no ofrecer visibilidad de su comportamiento interno. Ya que el código fuente no está disponible, esta visibilidad puede ofrecerse de diferentes formas, por ejemplo a través de interfaces especiales, archivos “log" o estructuras de datos visibles. Además de proporcionar visibilidad a nivel de componente, el sistema y la arquitectura pueden ofrecer visibilidad de aspectos de comportamiento. Por ejemplo, los protocolos de comunicación pueden ser monitorizados con rastreadores (sniffers), o el sistema operativo puede proporcionar información acerca de los recursos de uso, procesos en curso, etc.

1.3.1.3 Abierto

Un sistema abierto es aquel que ha sido construido acorde a unos estándares y tecnologías abiertas y disponibles. La apertura de un sistema permite que éste sea extendido e integrado.

1.3.1.4 Seguro

La seguridad es una propiedad que debe ser considerada en un sistema a nivel arquitectónico. Los componentes individuales pueden o no tener seguridad, pero es la arquitectura del sistema de software la que debe implementar las políticas de seguridad apropiadas.

1.3.2 LIMITACIONES DEL DESARROLLO DE SOFTWARE CON COMPONENTES COTS

El uso que se hace de la definición de componente COTS conlleva una serie de limitaciones. En primer lugar, aunque es cierto que desde 1994 se están utilizando prácticas para la utilización de componentes comerciales en procesos de desarrollo, la

realidad es que muchas organizaciones encuentran que el uso de componentes COTS conlleva un alto riesgo y esfuerzo de desarrollo, y para controlar su evolución y mantenimiento dentro del sistema. Estos problemas se deben en cierta medida, a que las organizaciones utilizan procesos y técnicas tradicionales para el desarrollo basado en componentes, pero no para componentes comerciales.

Otro inconveniente, es que los fabricantes de componentes COTS tampoco documentan de forma adecuada sus productos para que puedan ser consultados por usuarios desarrolladores que necesitan conocer detalles de especificación del componente, como información acerca de sus interfaces, protocolos de comunicación, características de implantación (tipos de sistemas operativos y procesadores donde funciona, lenguaje utilizado, dependencias con otros programas, etc.) y propiedades extra-funcionales.

Por último, dado que no existen formas globalmente conocidas para documentar los componentes COTS, también son inexistentes los repositorios que almacenen especificaciones de componentes COTS, y por tanto, en mayor medida, no se puede pensar en procesos de mediación que permitan por un lado a los “fabricantes de componentes COTS” poder anunciar sus productos, y por otro a los “usuarios de componentes COTS" poder buscar los componentes COTS que necesitan.

1.4 DESARROLLO DE SOFTWARE USANDO COMPONENTES COTS

Los sistemas de componentes COTS se construyen mediante la integración a gran escala de componentes adquiridos a terceras partes. La naturaleza de la caja negra de estos componentes, la posibilidad de que exista una incompatibilidad con la arquitectura, y su corto ciclo de vida, requiere una aproximación de desarrollo diferente. En la Tabla 1.2 se muestran algunas de las actividades en el desarrollo de software basado en componentes COTS. Estas actividades de desarrollo afectan tanto a la organización encargada de hacer la adquisición como a la organización responsable de llevar a cabo la integración de los componentes COTS.

ACTIVIDAD	DESCRIPCIÓN

Evaluación de Componentes	Cuando se diseña un sistema se debe localizar un conjunto de componentes COTS candidatos y evaluarlos con el propósito de seleccionar de aquellos componentes más adecuados para el sistema. Para un máximo beneficio en el uso de los componentes COTS, la evaluación se debe hacer a la vez que se definen los requisitos y se diseña la arquitectura. La evaluación debe requerir un conjunto de pruebas en fases muy tempranas del ciclo de desarrollo.
Diseño y Codificación	Se basa principalmente en la implementación de "envolventes" (código wrapper) y componentes de enlace (código glue).
Prueba	Las pruebas individuales y de integración se deben hacer como una caja negra. Como se ha mencionado, las pruebas individuales se deben hacer en las primeras fases del ciclo de desarrollo para llevar a cabo la evaluación de componentes COTS.
Detección de Fallos	Cuando la operatividad del sistema falla, los gestores deben ser capaces de determinar cual ha sido el/los componente/s que ha/n provocado el fallo.
Actualización de Componentes	Nuevas versiones de componentes COTS suelen aparecer en un corto periodo de tiempo. Los integradores deben poder: (a) evaluar toda nueva versión del componente; (b) determinar si la nueva versión debe ser integrada y cuando debe hacerlo; (c) realizar la instalación, integración y pruebas necesarias.
Gestión de Configuraciones	Los integradores deben llevar a cabo gestiones de configuración sobre varios componentes COTS. El modelo de componentes de un sistema basado en componentes COTS incluye: (a) determinar conjuntos de versiones de componentes compatibles con otros; (b) actualizar las versiones del componente como se requiere; (c) registrar toda versión de componente COTS que ha sido instalada; (d) generar un histórico de los componentes actualizados.

Tabla 1. 2 Actividades de los sistemas de componentes COTS[11]

[11] Meyers, B. C. y Oberndorf, P. Managing Software Acquisition: Open Systems and COTS Products. Pag: 78

La “Evaluación de componentes" conlleva conjuntamente las tareas de búsqueda, selección y evaluación de componentes comerciales. También es conocida como fase de adquisición de componentes comerciales, o simplemente fase de selección. La Figura 1.2 muestra un cuadro donde se comparan las distintas etapas de los procesos de desarrollo de los sistemas basados en componentes de software (DSBC) y los basados en componentes comerciales (COTS). En dicha figura se puede ver cómo la adquisición de componentes COTS coincide con las etapas de “Selección de componentes" y “Adaptación" en DSBC; o con las etapas de “Requisitos", “Especificación" y “Aprovisionamiento" del proceso RUP. Además, la etapa de “Diseño y codificación", “Prueba" y “Detección de fallos", coincide con la etapa de “Ensamblaje" en DSBC, y con las de “Ensamblaje" y “Pruebas" en RUP.

La selección de componentes COTS suele ser una tarea no trivial, donde hay que considerar diferentes aspectos de los componentes comerciales y de la arquitectura de software. En la actualidad representa el principal centro de interés en el área de los sistemas basados en componentes comerciales, y se pueden encontrar diversos enfoques. Por ejemplo, enfoques de la selección considerando aspectos de requisitos funcionales y/o extra-funcionales, o aspectos de atributos de calidad, o considerando métricas. Otros enfoques son por ejemplo los que consideran aspectos sobre cómo recoger los requisitos de un componente COTS en una arquitectura de software, o enfoques sobre aspectos de evaluación (adaptación o validación) de componentes comerciales.
En cuanto a los métodos de selección que consideran varios enfoques, se puede encontrar una gran variedad. Por un lado está el método CAP (COTS Acquisition Process), que se compone de tres partes: inicialización, ejecución y reutilización. La primera parte tiene que ver con procesos de adquisición y estimación de costos. La segunda parte guía en la evaluación de componentes COTS y en la toma de decisiones para la compra de los mejores componentes COTS (aquellos que se adecuan a las necesidades del usuario). Y la tercera parte recopila y almacena toda la información recogida en procesos anteriores para reducir el costo de futuros procesos en adquisición de componentes COTS.

A simple vista todo parece indicar que el desarrollo de sistemas basados en componentes COTS está empezando a ser factible debido numerosas razones:

a) En primer lugar debido a la mejora en la calidad y variedad de productos de componentes COTS;
b) También debido a presiones económicas para reducir costos de desarrollo y de mantenimiento del sistema;
c) Debido a la consolidación de una tecnología para la integración (ensamblaje) de componentes (la tecnología ORB);
d) Debido a una orientación en los últimos años hacia sistemas abiertos y estándares; y por último
e) Debido a un paulatino aumento en las prácticas de reutilización de software dentro de las organizaciones.

Sin embargo, hay otros autores que condicionan el desarrollo factible de sistemas basados en componentes comerciales a la existencia previa un mercado de componentes COTS consolidado.

El problema actual es que no existen unas estrategias de mercado estandarizadas, y son más bien estrategias particulares a cada fabricante de componentes COTS, o a lo sumo, estrategias de coalición entre dos o más fabricantes para lograr mayor cota de mercado. Incluso hay algunos autores como Wallnau que predicen que las estrategias de mercado nunca llegarán a ser estandarizadas, y por contra predicen una mayor consolidación de un mercado de componentes COTS multifacetas y de coaliciones.

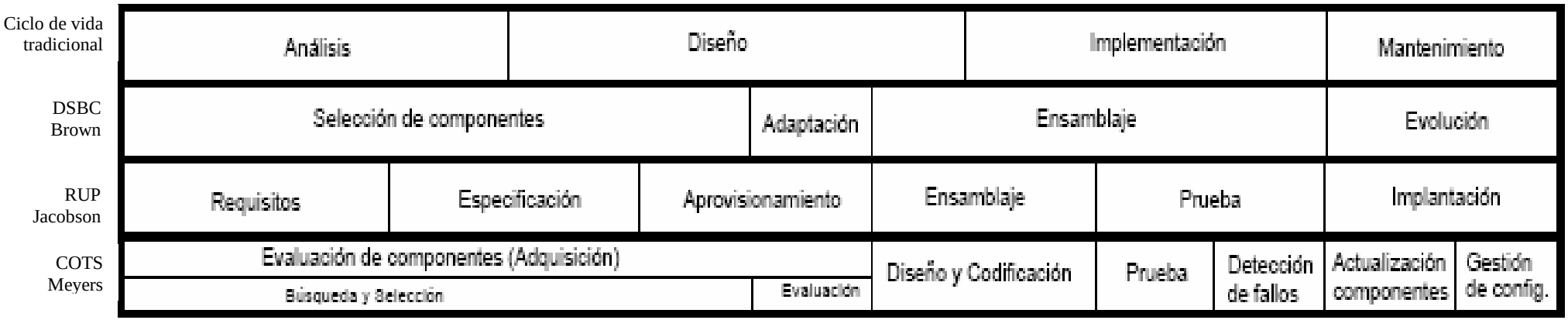

Figura 1 2 Comparación entre diferentes procesos de desarrollo basados en componentes

1.4.1 ANÁLISIS DE REQUISITOS

Las primeras fases del ciclo de vida en la construcción de un sistema de software comienzan con actividades de ingeniería para la identificación y el análisis de requisitos. Está comprobado que estas fases iniciales del desarrollo son las más críticas. De hecho, se sabe que aproximadamente el 60% de los errores detectados en las fases de diseño están directamente provocados por errores en las actividades de requisitos, aumentando con esto los plazos de entrega y los costos preliminares planificados.[12]

1.4.1.1 Ingeniería de requisitos tradicional

La “ingeniería de requisitos" se refiere al conjunto de tareas que tratan los requisitos de un sistema software. Desde una perspectiva tradicional, un requisito define una condición de necesidad, y consecuentemente, el diseño se hace como respuesta al requisito y para cumplir esta condición.

Como ha sucedido para otros términos vistos anteriormente, se puede encontrar también numerosas definiciones para el término “ingeniería de requisitos". En primer lugar está la perspectiva IEEE ofrecida en el glosario de términos de estándares de software, y que dice lo siguiente:

“La ingeniería de requisitos es el proceso de estudiar las necesidades de usuario para llegar a una definición de sistema, hardware o requisitos software... donde un requisito es definido como (1) una condición o capacidad que necesita un usuario para resolver un problema o para cumplir un objetivo; (2) una condición o capacidad que debe tener un sistema o componente para cumplir un contrato, estándar, especificación o cualquier otro documento impuesto formalmente; (3) una representación documentada de una condición o capacidad como en (1) o (2).”[13]

[12] Robertson, S. y Robertson, J. Mastering the RequirementsProcess. Pag: 180
[13] IEEE Standard Glossary of Software Engineering Terminology. Pag: 35

Sin embargo, esta definición es demasiado general y puede llevar a interpretaciones particulares o erróneas a cada organización. Una definición algo más especifica del término “ingeniería de requisitos" es la que ofrecen Jordon y Davis, que dice lo siguiente:

“La ingeniería de requisitos es el uso sistemático de principios, técnicas, lenguajes y herramientas que hacen efectivos el análisis, la documentación, la evolución de las necesidades de usuario y las especificaciones del comportamiento externo para cumplir aquellas necesidades de usuario.”[14]

Como se ve, esta definición centra su atención en tres áreas: el análisis, que se refiere a la identificación y recogida de requisitos; la documentación, que se refiere a la especificación y almacenamiento de los requisitos; y la evolución, que se refiere a que los requisitos sufren continuos cambios en las fases del ciclo de vida.

Otros autores dividen la ingeniería de requisitos en dos dominios: el dominio del problema y el dominio de la solución. Por un lado está el “dominio del problema", donde se analiza el sistema como un problema que hay que resolver y sus necesidades; y por otro lado está el “dominio de la solución", donde se estudia cómo se van a abordar cada una de las partes del problema identificadas en el dominio del problema.

En cualquier caso, lo que sí parece estar relativamente claro al hablar de requisitos, es en las categorías en las que se divide un requisito software y que son: (a) los requisitos funcionales, (b) los requisitos extra-funcionales, y (c) las restricciones de diseño.

[14] Jordon, K. y Davis, A. Requirements Engineering Metamodel: An Integrated View of Requirements. 475.

1.4.1.2 Prácticas de ingeniería de requisitos

Como prácticas de ingeniería de requisitos, tradicionalmente se utilizan diversos métodos y herramientas. En el caso de los métodos podemos encontrar diferentes categorías, como los métodos de especificación de requisitos, métodos de análisis de requisitos, y métodos de validación de requisitos. En esta línea se destaca el método REM (Requirement Engineering Method) que describe un modelo iterativo de procesos de especificación, análisis y validación de requisitos, basado en UML y unas plantillas y patrones de requisitos.

Como métodos de análisis de requisitos, se definen hasta cuatro categorías diferentes: (a) métodos orientados a proceso, (b) métodos orientados a datos, (c) métodos orientados a control, (d) métodos orientados a objeto.

Los métodos orientados a proceso tienen en cuenta la forma en la que el sistema transforma las entradas y salidas, haciendo menor énfasis en los datos y en el control. El análisis estructurado clásico es un caso de esta categoría.

Los métodos orientados a datos destacan el estado del sistema como una estructura de datos. A diferencia del análisis estructurado y las técnicas de análisis y diseño estructurado, que consideran los datos a nivel secundario, la modelización entidad-relación está principalmente orientada a datos.

Los métodos orientados a control hacen hincapié en la sincronización, bloqueos, exclusión, concurrencia y procesos de activación y desactivación. Las técnicas de análisis y diseño de datos, y extensiones de tiempo real (real-time) para análisis estructurado, son también orientados a control.

Finalmente, los métodos orientados a objetos basan el análisis de requisitos en clases de objetos y sus interacciones. En esta categoría incluimos las técnicas para describir casos de uso.

En cuanto a las herramientas utilizadas en ingeniería de requisitos, podemos encontrar una gran variedad, clasificadas básicamente en cinco categorías: (a) herramientas de edición gráfica, (b) herramientas de trazabilidad, (c) herramientas para modelar comportamiento, (d) herramientas de bases de datos y procesamiento de textos, y (e) herramientas híbridas (como combinación de las anteriores).

Hoy día, la tendencia es a utilizar notación UML como herramienta de edición gráfica para modelar sistemas de software (y no solo para objetos), ya que integra múltiples técnicas que pueden ser utilizadas para el análisis de requisitos, como los diagramas de secuencias, diagramas de casos de uso, y diagramas de estados, entre otros.

1.4.1.3 Ingeniería de requisitos y componentes COTS

Inicialmente, para la construcción de un sistema software basado en componentes, los ingenieros seguían el enfoque tradicional descendente basado en el modelo de desarrollo de software en cascada clásico. En este enfoque existe primero una fase de análisis, donde se identifican y definen los requisitos de los componentes software; luego se diseñan los componentes y la arquitectura de software del sistema; se llevan a cabo labores de implementación y pruebas por separado de los componentes, y ensamblaje de estos (creando componentes ORB y envolventes) y pruebas finales. Sin embargo, este modelo es totalmente secuencial, desde el análisis de requisitos hasta llegar a obtener el producto final.

Con el tiempo se ha visto que el modelo de desarrollo de software en cascada no era el más idóneo para la construcción de sistemas basados en componentes. Los procesos de reutilización de software hacen que los requisitos de algunos componentes puedan ser satisfechos directamente, sin necesidad de tener que llegar a etapas de implementación (para esos componentes). En el mejor de los casos, los requisitos de los componentes involucrados en los procesos de reutilización, estaban completamente controlados, ya que las técnicas de reutilización de componentes se aplicaban sobre repositorios, catálogos o librerías de componentes que la propia organización había desarrollado en proyectos anteriores.

El problema aparece cuando, a falta de estos repositorios de componentes internos, la organización decide incluir componentes desarrollados fuera de ésta en los procesos de reutilización, como es el caso de los componentes COTS. En este caso, los procesos de adquisición de componentes COTS involucran tareas de búsqueda, selección y evaluación de esta clase de componente. Como resultado, se podría dar el caso de que, por desconocimiento, algunos requisitos que deberían haber sido impuestos para un componente podrían omitirse en su planteamiento inicial (cuando se hizo la definición de los requisitos del componente).

El modelo de desarrollo en espiral es el más utilizado en la construcción de sistemas basados en componentes COTS. Este tipo de desarrollo asume que no todos los requisitos de un componente puedan ser identificados en las primeras fases del análisis de requisitos, pudiendo aparecer nuevos de ellos en cualquier otra fase del ciclo de vida, que tengan que ser también contemplados; o incluso puede que los requisitos actuales tengan que ser refinados como resultado de la evaluación de los componentes adquiridos.

Existen varias técnicas que pueden ser utilizadas para evaluar productos de componentes COTS, todas ellas llevadas a cabo de “forma manual" por el equipo de personas encargado de realizar las tareas de evaluación. Algunos ejemplos de estas técnicas de evaluación son:

- Análisis de la teoría existente, referencias de otros usuarios y del vendedor.
- Análisis de la conexión diseño-implementación (conocido como Gap analysis), que consiste en determinar qué requisitos son, o no, satisfechos por el producto (siendo evaluado) y las características del producto que no han sido recogidas como requisitos.
- Mediante demostraciones facilitadas por el vendedor.
- Mediante problemas de modelo, que son pequeños experimentos que se centran en cuestiones especificas de diseño y de comportamiento del producto.
- Prototipos de otros subsistemas que utilizan componentes COTS.
- Utilizando técnicas de evaluación de otros usuarios, como por ejemplo la técnica RCPEP, que consiste en un proceso de evaluación de productos de componentes COTS guiado por requisitos.

Hoy día también existen diversos métodos que se están aplicando en ingeniería de requisitos para componentes COTS. Ejemplos de estos métodos son, PORE (Procurement-Oriented Requirement Engineering) y ACRE (Acquisition of Requirements). El primero utiliza un método basado en plantilla a tres niveles para la recogida de requisitos y el segundo proporciona un marco de trabajo para la selección y utilización de diversas técnicas de adquisición de requisitos.

Otro método conocido es OTSO (Off-The-Shelf Option), desarrollado en procesos de selección de componentes reutilizables, y que tiene en cuenta requisitos funcionales específicos de la aplicación, aspectos de diseño y restricciones de la arquitectura.

Otro método es el que utiliza el lenguaje NoFun, este lenguaje utiliza UML para la definición de comportamiento extra-funcional de componentes software como atributos de calidad. Además, en una reciente revisión del lenguaje, NoFun permite definir diferentes tipos de componentes y relaciones entre componentes y subcomponentes, y adopta el estándar ISO/IEC-9126 para tratar atributos de calidad.

También están los métodos relacionados con atributos de calidad para la selección de componentes COTS, donde se proponen unos modelos de calidad y de certificación de componentes basados en el modelo de calidad de ISO, ISO/IEC-9126 y en el lenguaje NoFun como notación estructurada para la formalización de estos modelos.

Para finalizar, están los métodos para la construcción de componentes COTS considerando atributos de calidad (funcionales y extra-funcionales) y métricas para la evaluación y selección de componentes COTS. En esta línea se destaca el modelo COCOTS, basado en el modelo de construcción de software COCOMO-II.

1.4.2 ARQUITECTURAS DE SOFTWARE

Las arquitecturas son fundamentales en cualquier sistema, especialmente para los sistemas abiertos. Como en un modelo de referencia, una arquitectura permite centrarse en las características y funciones de un sistema, pero con la diferencia de que además también permite especificar algunas características de implementación del mismo.

Existen algunas definiciones concretas para el término “arquitectura de software". Debido a la extensa y variada gama de trabajos en el campo de las arquitecturas, sólo se hará referencia a dos de las definiciones más usadas últimamente: la ofrecida por el estándar 1471 de IEEE, y la ofrecida por Bass, Clements y Kazman.

El estándar 1471 de IEEE identifica ciertas prácticas para establecer un marco de trabajo (framework) y un vocabulario unificado para conceptos relacionados con las arquitecturas de software. El estándar define una arquitectura de software como:

“Parte fundamental de un sistema expresado por sus componentes, sus relaciones con otros componente y otros entornos, y los principios que guían en su diseño y evolución.”[15]

Otra definición interesante de arquitectura de software es la que ofrecen Bass, Clements y Kazman. Esta definición es ampliamente adoptada por otros autores en arquitecturas de software. La definición de Len Bass dice lo siguiente:

“Arquitectura de software de un programa o sistema de computación es la estructura o estructuras del sistema, que están compuestas de componentes software, de las propiedades visibles de esos componentes, y las relaciones entre ellos.”[16]

Hay un par de aspectos a considerar para esta definición. En primer lugar, el término “componente" se refiere a un elemento de software simple o a una colección de otros elementos de software. En segundo lugar, las propiedades visibles se refieren a los requisitos de componente (funcionales y extra-funcionales) identificados en la fase de análisis de requisitos.

[15] Maier, M. W., Emery, D., y Hilliard, R. Software Architecture: Introducing IEEE Standard 1471. Pag: 107.
[16] Bass, L., Clements, P., y Kazman, R. Software Architecture in Practice. Pag: 214

1.4.2.1 Características de las arquitecturas de software

Las arquitecturas de software generalmente juegan el papel de “pasarelas" entre los requisitos y la implementación. Mediante una descripción abstracta de un sistema, la arquitectura expone ciertas propiedades, mientras oculta otras.

Las arquitecturas de software pueden jugar un importante papel en al menos seis aspectos del desarrollo de software:

a) Comprensión del sistema: Una arquitectura de software facilita la comprensión de un sistema, al poder representarlo con un alto nivel de abstracción, y donde aspectos de diseño del sistema pueden ser fácilmente comprendidos a alto nivel.
b) Reutilización: Las descripciones arquitectónicas soportan reutilización de múltiples formas, generalmente de componentes y marcos de trabajo (framework). Ejemplos de estos son los estilos arquitectónicos y los patrones de diseño arquitectónicos.
c) Construcción: Una descripción arquitectónica permite tener una visión parcial del sistema que hay que construir, describiendo sus componentes y las dependencias entre ellos.
d) Evolución: Una arquitectura permite separar lo que concierne a la parte funcional de un componente de las formas en las que este componente puede ser conectado a otros componentes. Esta separación facilita que luego se puedan hacer cambios en la arquitectura por aspectos de interoperabilidad, prototipado y reutilización.
e) Análisis: Una arquitectura de software es una buena oportunidad para hacer de nuevo, prácticas de análisis y refinar los requisitos identificados en fases de análisis de requisitos. Algunas prácticas de análisis que se pueden aplicar a este nivel son, por ejemplo: comprobaciones de consistencia, análisis de dependencias o comprobaciones para ver si se cumplen con las restricciones impuestas en las partes de la arquitectura.
f) Decisión: Una arquitectura permite desvelar ciertos detalles que pueden decidir las estrategias de implementación a seguir, o modificar o incluir nuevos requisitos.

En una arquitectura de software se describen los detalles de diseño de una colección de componentes y sus interconexiones, que conforman una vista abstracta a alto nivel del

sistema que se está diseñando, y donde se consideran los requisitos identificados en la fase de análisis de requisitos del sistema.

Actualmente, en la comunidad de arquitecturas de software existe una gran variedad de elementos arquitectónicos que simplifican las tareas de diseño en la construcción de una arquitectura. Estos elementos arquitectónicos se conocen con el nombre de “estilos arquitectónicos". Un estilo arquitectónico está compuesto por un conjunto de estilos de componente a nivel arquitectónico y por unas descripciones de “patrones" de interacción. Estos tipos de componente son utilizados para modelar las interacciones que tienen lugar en una infraestructura de componentes. De igual forma que los patrones de diseño orientados a objetos ayudan a los desarrolladores a diseñar sus clases, los estilos arquitectónicos sirven de ayuda en las tareas de diseño de componentes en una arquitectura de software.

Tradicionalmente una arquitectura de software se ha centrado en la descripción y análisis de estructuras “estáticas". Sin embargo, en sistemas complejos (abiertos, distribuidos, adaptativos y evolutivos), donde intervienen por ejemplo componentes de granularidad gruesa, como los componentes comerciales, y en los que la estructura evoluciona a lo largo del tiempo, el sistema podría necesitar de patrones arquitectónicos “dinámicos", donde se propone un arquitectura de software dinámica que utiliza un enfoque reflexivo para permitir la reconfiguración automática de la arquitectura como respuesta a la evolución del sistema.

1.4.2.2 Lenguajes para la definición de arquitecturas de software

En una arquitectura de software se describen los detalles de diseño de una colección de componentes y sus interconexiones, que conforman una vista abstracta a alto nivel del sistema que se está diseñando, y donde se consideran los requisitos identificados en la fase de “análisis de requisitos" del sistema.

Las interconexiones entre los componentes permiten definir aspectos de interoperabilidad y analizar detalles de compatibilidad entre estos, y suelen ser diseñados como componentes independientes que controlan aspectos de interconexión, como los protocolos de

interconexión, que establecen el orden en el que se establecen las llamadas a las operaciones entre dos componentes, y la compatibilidad sintáctica y de comportamiento en las llamadas controladas por la interconexión. A estas interconexiones, o componentes de interconexión, se les denominan “conectores".

Para describir una arquitectura de software se utiliza un lenguaje para la descripción de arquitecturas (Architecture Description Language, ADL). En general, un LDA debería ofrecer las siguientes características:

1) Una colección de elementos que permita modelar las partes de la arquitectura, como componentes, conectores o puertos, entre otros.
2) Métodos y herramientas que faciliten la construcción de la arquitectura, como compiladores, herramientas con notación gráfica para “dibujar" los elementos arquitectónicos (componentes, puertos, conectores, etc.).
3) Que soporte los aspectos de comprensión, reutilización, construcción, evolución, análisis y decisión.

No obstante, tradicionalmente un LDA ha sido identificado como un lenguaje que ofrece una colección de elementos para modelar la arquitectura de software siguiendo el modelo “Componente-Puerto-Conector":

- Componentes: Representan los elementos computacionales de un sistema, y pueden tener múltiples interfaces definidas en los puertos.
- Puertos: Representan la forma de acceso al componente. Los puertos definen las interfaces que el componente proporciona y las interfaces que éste requiere de otros componentes para funcionar.
- Conectores: Son las interconexiones entre componentes, y se realizan por medio de los puertos. Un conector queda definido como un componente independiente con tareas exclusivas de interconexión entre dos componentes.

Tal y como se puede comprobar en la Tabla 1.3., hoy día existe una gran variedad de LDAs, y difieren o asemejan entre ellos por poseer o carecer algunas cualidades. Por ejemplo, en las columnas de la 3 a la 8 se recogen algunas de estas cualidades (o

propiedades) deseables en un LDA. Estas columnas (propiedades) se interpretan en la tabla de la siguiente forma: (columna 3) el tipo de notación utilizada para escribir la arquitectura (gráfica o textual); (columna 4) lenguajes que soporta; (columna 5) si soporta trazabilidad; (columna 6) si contempla la evolución del sistema; (columna 7) si permite reconfiguración de la arquitectura; (columna 8) si soporta la definición de propiedades extra-funcionales.

	Ref.	G/T	Lenguajes	Trazab.	Evol.	Dinam.	Extra Func.
ACME	[Garlan et al., 2000]	T	propio	No	Si	No	No
AESOP	[Garlan et al., 1994]	G	C++	No	No	No	No
C2	[Medvidovic et al., 1996]	G	C++, Java y Ada	No	Si	Si	No
DARWIN	[Magee y Kramer, 1996]	G	C++	Si	No	Si	No
LEDA	[Canal, 2000]	T	Java	No	Si	Si	No
METAH	[Binns et al., 1995]	G	Ada	Si	No	No	Si
RAPIDE	[Luckham et al., 1995]	G	VHDL, C++ y Ada	Si	No	Si	Si
SADL	[Moriconi et al., 1995]	T	propio	Si	No	No	No
UNICON	[Shaw et al., 1995]	G	propio	Si	No	No	No
WRIGHT	[Allen y Garlan, 1997]	T	propio	No	Si	No	No

Tabla 1. 3 Cuadro Comparativo para algunos LDAs conocidos

Recientemente, se ha visto una tendencia a utilizar notación UML para la descripción de arquitecturas de software, algunos autores analizan una propuesta para la descripción en UML de arquitecturas de software basadas en componentes COTS; otros analizan cómo incluir, mediante notación UML, los tipos de requisitos extra-funcionales de NoFun dentro de una arquitectura de software. También es importante destacar en este sentido la propuesta “Componentes UML", donde se habla de arquitecturas de componentes modeladas en UML.

Tradicionalmente, UML no ha sido considerado como un LDA por la falta de una notación suficiente para describir elementos arquitectónicos, como los conectores, protocolos o propiedades. Los recientes trabajos recurren a las extensiones de UML, como las restricciones, valores etiquetados, estereotipos y notas, para suplir esta carencia arquitectónica.

Sin embargo, el uso de diagramas de clases UML para modelar la descripción de una arquitectura de software puede dar lugar a representaciones gráficas demasiado extensas, ya que cada elemento arquitectónico (componentes, conectores, puertos y protocolos) debe ser representado como una clase estereotipada (por ejemplo., <<componente>>,

<<conector>>). Además, los sistemas basados en componentes COTS suelen ser sistemas a gran escala, compuestos por un gran número de componentes (comerciales y no comerciales) y con múltiples conexiones. Por tanto, esto multiplica el número de clases posibles en una definición UML de una arquitectura de software.

1.4.2.3 Componentes UML

Los Componentes UML son una propuesta original de John Cheesman y John Daniels para modelar arquitecturas de componentes y especificar componentes de software utilizando extensiones de UML con estereotipos, y diseño por contratos utilizando OCL (Object Constraint Language).

Como muestra la Figura 1.3, los Componentes UML están inspirados en trabajos previos en los cuales los propios autores han participado. Los trabajos directamente relacionados son: UML de Rumbaugh, donde John Cheesman colabora con OIM (Open Information Model); los procesos de desarrollo de software RUP (Racional Unified Process) y Catalysis; y el método Advisor para el desarrollo basado en componentes (inspirado en Catalysis). Por otro lado, Catalysis se inspira en el método Syntropy, un trabajo original de John Daniels que ha servido de base para desarrollar OCL y OIM.

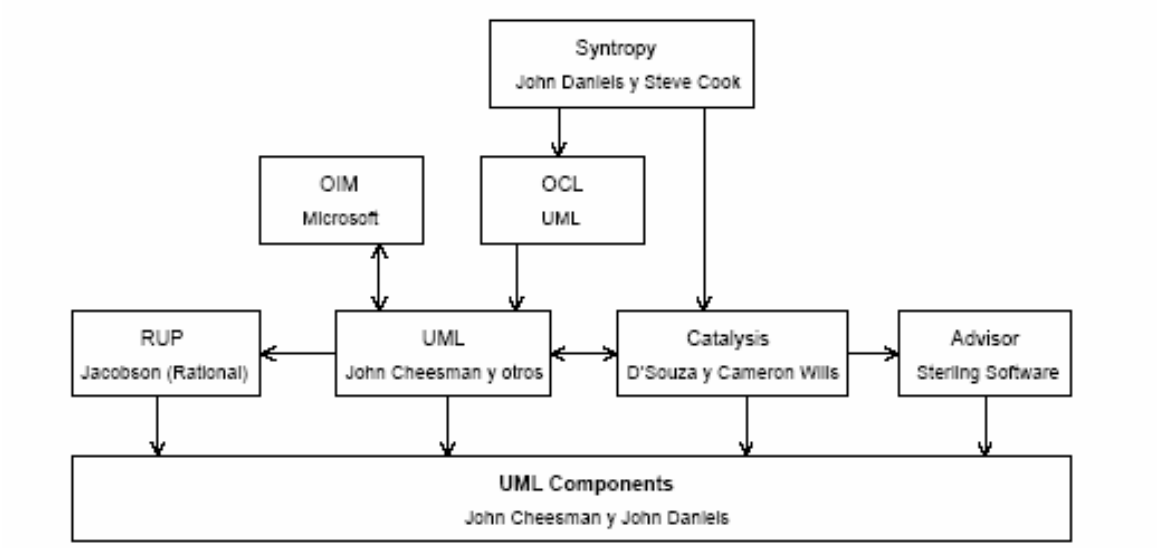

Figura 1 3 Trabajos Base de los Componentes UML[17]

En componentes UML, una arquitectura de componentes es un conjunto de componentes de software a nivel de aplicación, con sus (1) relaciones estructurales y (2) sus dependencias de comportamiento. Una arquitectura de componentes puede ser utilizada para modelar una aplicación de software (a partir de componentes) o para modelar un sistema de software como un conjunto de aplicaciones (consideradas estas como componentes y subcomponentes). Las relaciones estructurales (1) significan asociaciones y relaciones de herencia entre especificaciones de componente e interfaces de componente, y relaciones de composición entre componentes. Las dependencias de comportamiento (2) son relaciones de dependencia: (a) entre componentes, (b) entre componentes e interfaces, (c) entre interfaces, y (d) entre subcomponentes y componentes, (como se muestra en la Figura 1.4 A).

[17] Cheesman, J. y Daniels, J. UML Components. A Simple Process for Specifying Component-Based Software. Pag: 42

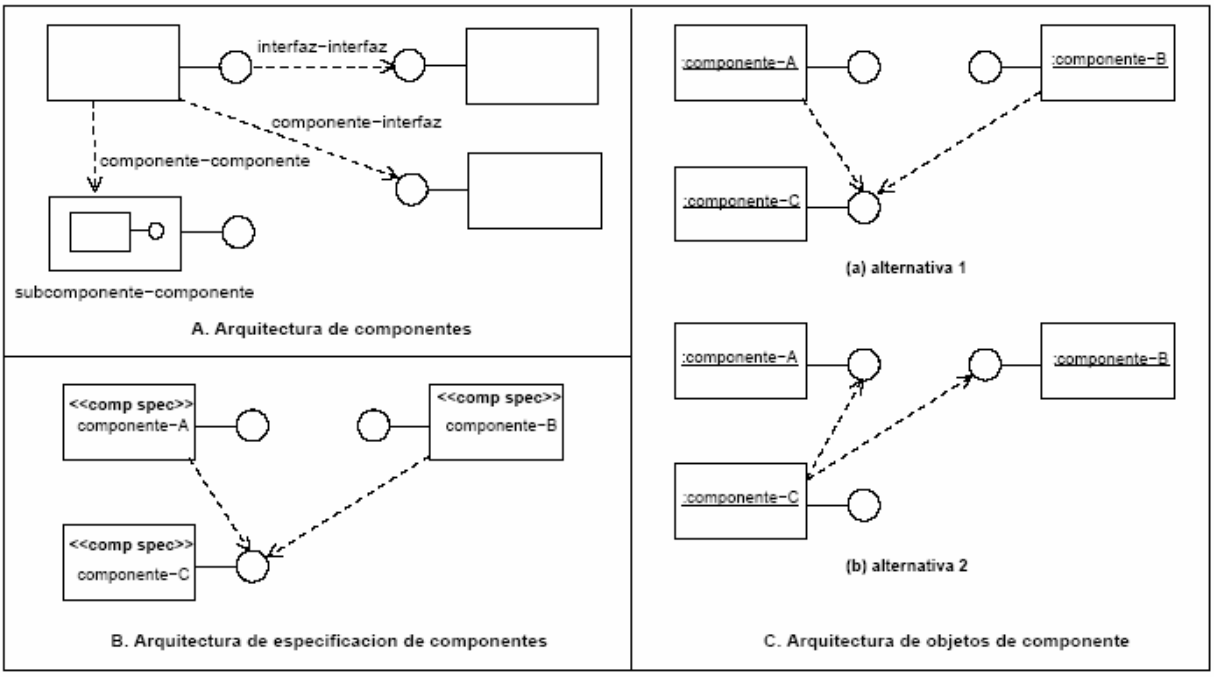

Figura 1 4 Dependencias en una arquitectura de componentes

Además, una arquitectura de componentes se puede centrar en especificaciones de componente, en implementaciones de componente o en objetos de componente.

Un diagrama de una arquitectura de especificaciones de componente contiene sólo especificaciones de componente e interfaces. Un diagrama de una arquitectura de implementaciones de componente muestra las dependencias que existe entre las implementaciones de un componente en particular. Y por último, un diagrama de una arquitectura de objetos de componente especifica la relación entre las instancias de cada componente. En la Figura 1.4.C se muestra dos posibles alternativas de arquitecturas de objetos de componente que cumplen con la arquitectura de especificaciones de componente de la Figura 1.4.B.

Los Componentes UML utilizan los diagramas de casos de uso para modelar requisitos, y los diagramas de clases y colaboraciones para modelar especificaciones. Por ejemplo, una especificación de un sistema de componentes está compuesta de cuatro partes: (a) los tipos de datos, (b) las especificaciones de interfaz, (c) las especificaciones de componente, y (d) la arquitectura de componentes. Para modelar estas partes se utilizan los diagramas de clases, y en Componentes UML cada diagrama resultante recibe el nombre de: (a) diagrama de tipos de datos, (b) diagrama de especificaciones de interfaz, (c) diagramas de

especificaciones de componente, y (d) diagrama de arquitectura de componentes. Para esta última (los diagramas de arquitectura de componentes) las interacciones se modelan con diagramas de colaboración, y reciben el nombre de diagramas de interacciones de componente.

Los Componentes UML utilizan los estereotipos para extender UML. Por ejemplo, para los diagramas de tipos de datos se utilizan los estereotipos <<type>> y <<datatype>>; para los diagramas de especificaciones de interfaz se utilizan estereotipos como <<interface type>> o <<info type>>; para los diagramas de especificaciones de componente se utilizan estereotipos como <<comp spec>> (como se vio en la Figura 1.4) o <<offers>>. En el diagrama de arquitectura de componentes se pueden utilizar todos los estereotipos.

1.4.2.4 UML-RT

Aunque es algo más antigua que las demás, la notación de Bran Selic denominada ROOM (Real-time Object Oriented Modelling), y usada para modelar objetos complejos en tiempo real, ha sido una de las más empleadas para extensiones y propuestas de LDA futuras. Inicialmente esta propuesta no se basaba en UML, pero recientes trabajos la han extendido para conocerse ahora como UML-RT, una mezcla entre UML estándar y ROOM. En la actualidad UMLRT ha sido adoptado por Rational en su herramienta Rational Rose RealTime.

UML-RT es una notación gráfica que se basa en un diagrama tradicional de “líneas-y-cajas". En la Figura 1.5 se puede ver un ejemplo de este tipo de diagrama. Según la notación de UML-RT, los componentes se representan mediante cajas, denominadas “cápsulas", que pueden contener a su vez otras cápsulas. Esto último se representa en el diagrama mediante dos pequeños cuadros conectados, como se puede ver en la esquina inferior derecha de la cápsula GTS. Esto nos asegura que GTS es el resultado de la composición de otras cápsulas internas a ella.

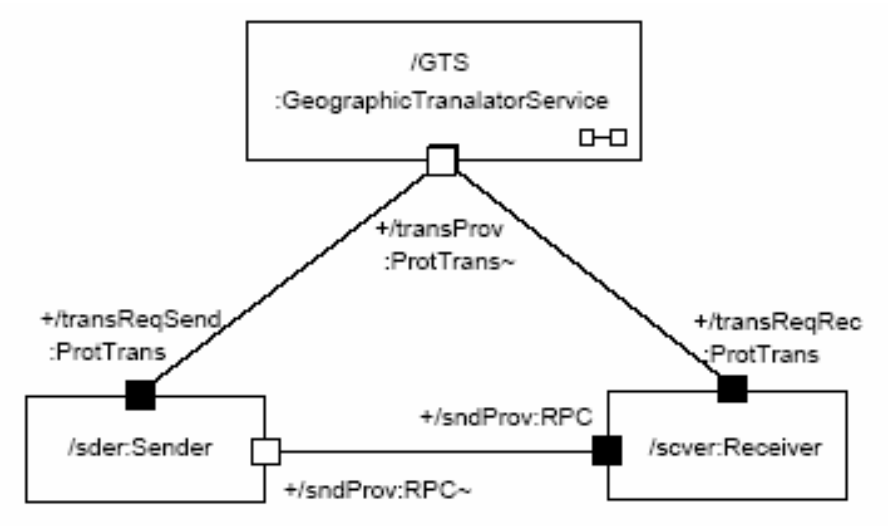

Figura 1 5 Un ejemplo que utiliza notación UML-RT

Cada cápsula (componente) puede tener una o más interfaces, denominados “puertos" y representados por unos cuadrados pequeños de color negro o blanco. Un puerto de color

blanco representa los mensajes de entrada y los de color negro los de salida. En cualquier caso, esto se interpretará como las interfaces que el componente oferta (color blanco) y las que requiere para poder funcionar (color negro).

Según esto, en el diagrama de la Figura 1.5 intervienen tres componentes: *GTS, sver y rcver*. Una notación de la forma */GTS:GeographicTranlatorService* significa que GTS es una instancia del componente base *GeographicTranslatorService*. Cada puerto se denota con un nombre y un “protocolo" de comunicación, que establece el orden en el que se envían los mensajes del puerto que encapsula. Por ejemplo, un puerto *+transReqSend:ProtTrans* significa que es una interfaz requerida llamada *transReqSend* y que respeta un protocolo llamado *ProtTrans*. Una interfaz ofertada se representa de forma similar, pero con su “dual", *+transReqSend:ProtTrans~*.

Para finalizar, los “conectores" se representan como simples líneas que unen en sus extremos dos puertos. Si un conector une un puerto con más de uno, estos se deben duplicar en la cápsula que lo requiere. Esto se representa mediante un doble cuadro pequeño, y fuera se indica la cardinalidad de la repetición (aunque esto último no es obligatorio). Por ejemplo, el componente *GTS* replica dos veces la interfaz *transProv* porque enlaza con dos conectores, uno proveniente de la interfaz *transReqRec* del componente *rvcer*, y otro proveniente de la interfaz *transReqSend* del componente *sder*.

1.4.3 DOCUMENTACIÓN Y ESPECIFICACIÓN DE COMPONENTES COMERCIALES.

Un componente software, en general, puede quedar especificado por medio de sus interfaces, que describen las operaciones, atributos, parámetros y otra información sintáctica. No obstante, esta información sintáctica no es suficiente a la hora de construir aplicaciones. También es necesaria la información de los protocolos, que describen la interacción del componente, o la información semántica, que describe el comportamiento de las operaciones del componente. Para el caso de los protocolos, existen diferentes formalismos para describirlos, como máquinas de estados finitas, redes de Petri, lógica temporal o el ¶-cálculo. Para el caso de la información semántica, se usan formalismos

como pre/post condiciones e invariantes, ecuaciones algebraicas o el cálculo de refinamiento.

De forma similar, es necesario un mecanismo para documentación de componentes comerciales, muy útil para tareas de búsqueda, selección y ensamblaje de componentes. Para estas labores, es necesario que la documentación de los componentes contenga información del tipo funcional como signaturas, protocolos y comportamiento, y también información no funcional y no técnica del componente.

1.4.3.1 Descripción funcional

Una descripción funcional de un componente recoge información de los requisitos funcionales a nivel sintáctico y semántico de las interfaces proporcionadas y requeridas por el componente. Una descripción funcional también cubre aspectos relacionados con los eventos producidos o consumidos por el componente, y también aspectos de los protocolos de interacción de las interfaces del componente con otros componentes.

Siguiendo las pautas adoptadas para la exposición del modelo de documentación COTS, se define en primer lugar el significado de "descripción funcional". Para la parte funcional de un documento COTS, el modelo establece con carácter obligatorio la presencia de al menos una interfaz proporcionada por el componente. Tanto las interfaces requeridas por el componente, como los eventos producidos y consumidos por éste y sus protocolos de interacción, pueden no aparecer en un documento COTS (debido al carácter opcional en su definición).

1.4.3.1.1 Interfaces

El modelo de documentación de componentes COTS contempla una interfaz de componente como una colección de información que está relacionada con aspectos sintácticos y semánticos de la interfaz.

Los aspectos sintácticos se refieren a la forma en la que tradicionalmente se define una interfaz, esto es, con sus atributos, operaciones (o m´rtodos) y eventos, como se hace en los modelos de componentes EJB, CCM o EDOC.

Los aspectos semánticos como los contratos, recogen el comportamiento individual de las operaciones de interfaz.

1.4.3.1.2 Comportamiento de las interfaces

Similar a como se hace para la descripción sintáctica, el modelo de documentación recoge la descripción semántica de una interfaz mediante una descripción del comportamiento de la interfaz junto con la notación en la que ésta viene expresada. Las notaciones más usuales para recoger el comportamiento de un componente son aquellas basadas en prepost-condiciones, como por ejemplo, JML (Java Modeling Language o JavaLarch) y OCL (Object Constraints Language).

Asociada a la descripción de comportamiento, el modelo de documentación también admite la presencia (opcional) de dos tipos de operadores de emparejamiento semántico: uno fuerte (o exacto), y otro débil (o relajado).
Estos dos operadores son los que luego utilizará el modelo de mediación para los procesos de búsqueda y selección en las labores de reemplazabilidad semántica entre interfaces.

Como en el caso de los operadores sintácticos, el modelo de documentación establece con carácter opcional la presencia de estos dos operadores de emparejamiento semántico dentro de un documento COTS: pudiendo aparecer los dos, uno de ellos o incluso ninguno.

1.4.3.1.3 Eventos

El modelo de documentación de componentes comerciales contempla la posibilidad de incluir la definición de los eventos producidos y consumidos por un componente en la parte funcional de un documento COTS, tal y como se define en los modelos de componentes CCM, EJB, o EDOC.

1.4.3.1.4 Coreografía (protocolos)

La última parte de una descripción funcional de un documento COTS hace referencia a la coreografía del componente, normalmente conocida como información de protocolo. Un protocolo establece el orden en el cual las operaciones de las signaturas de una interfaz deben ser llamadas. Además el orden de estas llamadas puede variar dependiendo del tipo del componente con el que interacciona y del escenario donde se lleva a cabo la interacción (en este caso, a las interfaces se las denomina roles). Por tanto, a nivel de componente, se puede hablar de diferentes protocolos en función del escenario donde éste sea utilizado.
La descripción funcional de un documento COTS establece con carácter opcional la presencia de la coreografía de un componente, y que, en caso de existir, ésta queda establecida con una descripción de la coreografía junto con la notación en la cual ésta viene expresada, por ejemplo, usando formalismos como redes de Petri, máquinas de estados finitas, lógica temporal o el ¶-cálculo de Milner, entre otros.

1.4.3.2 Descripción extra-funcional

La segunda parte de un documento COTS tiene que ver con la información extra-funcional de un componente. La información extra-funcional cubre aspectos que no están directamente relacionados con la funcionalidad del componente, como la calidad de servicio, el contexto del funcionamiento, y otras propiedades extra-funcionales.

Esta información juega un papel muy importante en algunas de las actividades del desarrollo de software basado en componentes, como son las actividades de búsqueda y selección de componentes, o las actividades de evaluación de componentes COTS. Estos requisitos extra-funcionales podrán tener, en algunos casos, una prioridad mayor que los requisitos funcionales, o incluso ser decisivos a la hora de decidir por un componente u otro en actividades de selección, en el caso de encontrar dos o más componentes con una funcionalidad similar.

Para la definición de los requisitos extra-funcionales, el modelo de documentación de componentes COTS adopta el modelo de propiedades de ODP, que usa tripletas de la forma: nombre, tipo, valor. Las propiedades son la forma usual en la cual se expresa en la

literatura los aspectos extra-funcionales de los objetos, servicios y componentes. El modelo de documentación también propone el uso de tipos W3C basados en XML para describir las propiedades, aunque cualquier notación sería válida para describirlas, como por ejemplo el estilo CCM de OMG que utiliza también un vocabulario XML.

La parte extra-funcional de un componente puede quedar especificada por una secuencia de propiedades capturando, cada una de ellas, un atributo extra-funcional de componente con tres partes: (a) el nombre del atributo, (b) el tipo de dato del valor capturado por el atributo, y (c) su valor actual. Además de representar propiedades simples, el modelo también permite definir propiedades complejas, y aspectos de trazabilidad entre los atributos extra-funcionales y la funcionalidad del componente.

1.4.3.3 Descripción de empaquetamiento <packaging>

La tercera parte de un documento COTS tiene que ver con la información de empaquetamiento del componente. Esta información está relacionada con aspectos de descarga e instalación del componente y aspectos de implementación, como por ejemplo los tipos de sistemas operativos o procesadores válidos para que el componente pueda funcionar, o las dependencias del componente con aquellos programas que deben estar instalados en el entorno de implantación antes de hacer la descarga, entre otros aspectos de empaquetamiento.

1.4.3.4 Descripción de marketing

La cuarta y última parte de un documento COTS tiene que ver con una descripción no técnica del componente y del fabricante del componente. En el modelo de documentación de componentes COTS, a esta clase de información se la denomina “descripción de marketing" (o requisitos de marketing).

1.4.4 EL SERVICIO DE MEDIACIÓN (TRADER)

El servicio de trading, conocido también como función de mediación, o simplemente trader, es un objeto software que sirve de intermediario entre unos objetos que ofertan

ciertas capacidades, que se denominan servicios, y otros objetos que demandan la utilización dinámica de estas capacidades. La función de trading es una de las 24 funciones del modelo ODP (Reference Model of Open Distributed Processing, RM-ODP), establecida como norma por ISO/ITU-T. Esta especificación ha sido adoptada por el Object Management Group (OMG) con el nombre de CosTrading para el servicio de trading de CORBAservices, y en la actualidad existen diversas implementaciones disponibles en el mercado.[18]

Sin embargo, ya que la actual función de trading de ODP está limitada sólo para objetos, y no es suficiente para el caso de los componentes COTS, existe también el Proceso de Mediación de COTStrader.

[18] IRIBARNE, L., TROYA, J. M., y VALLECILLO, A., Study of the RM-ODP Trading Function apud ISO/IEC-ITU/T, Information Technology - Open Distributed Processing - Trading function: Specification. ISO/IEC 13235-1, UIT-T X.950, 1997.

1.4.4.1 El proceso de mediación de COTStrader

El proceso de mediación involucra dos actividades principalmente, la actividad de registrar o anunciar servicios de componente COTS y la actividad de buscar servicios de componente COTS con ciertas características. Cuando un proveedor de servicios desee publicar un servicio particular, éste se pondría en contacto con un mediador a través de la interfaz Register, ofreciéndole una plantilla COTScomponent con la especificación del componente COTS que desea publicar. Luego, el mediador almacenará esta plantilla en algún repositorio asociado.

Por otro lado, el proceso de mediación se lleva a cabo realmente en las actividades de consulta a través de la interfaz Lookup del mediador. Cuando se recibe una consulta, el mediador intenta buscar aquellas descripciones de componente del repositorio, que casan con la descripción de componente requerida. Las condiciones de la consulta (en contenido y forma) y los criterios de búsqueda deberían estar establecidos adecuadamente en una plantilla de consulta COTSquery, ofrecida como argumento en el momento de hacer la llamada a la operación query() de la interfaz Lookup.

El algoritmo de búsqueda que sigue el mediador COSTtrader comprueba si una plantilla de consulta Q casa con una plantilla T residente en el repositorio asociado. El mediador recorre el repositorio completo y va almacenando aquellas plantillas T que van casando en una lista de candidatos.

En el caso de los elementos funcionales, las plantillas pueden especificar o no los programas de emparejamiento para cada elemento particular: interfaces, comportamiento o protocolo. Los eventos son emparejados sólo a nivel sintáctico.

Tras repetir este proceso para todos los elementos en Q, la plantilla T es: (a) descartada si cualquiera de las pruebas ha fallado, (b) es incluida en la lista de candidatos si todas las pruebas se han llevado a cabo con éxito, o (c) es considerada como candidata “potencial" si todas las pruebas se han pasado satisfactoriamente porque no se ha encontrado un programa de emparejamiento para algún elemento funcional particular. La forma de tratar

los candidatos “potenciales" dependerá del orden de emparejamiento seleccionado por el cliente.

1.4.5 PROPUESTA DE PROCESO PARA EL DESARROLLO DE SOFTWARE USANDO COMPONENTES COMERCIALES COTS

La propuesta que se plantea para el proceso de desarrollo de software usando componentes comerciales COTS contempla las etapas indicadas en la Figura 1.6.

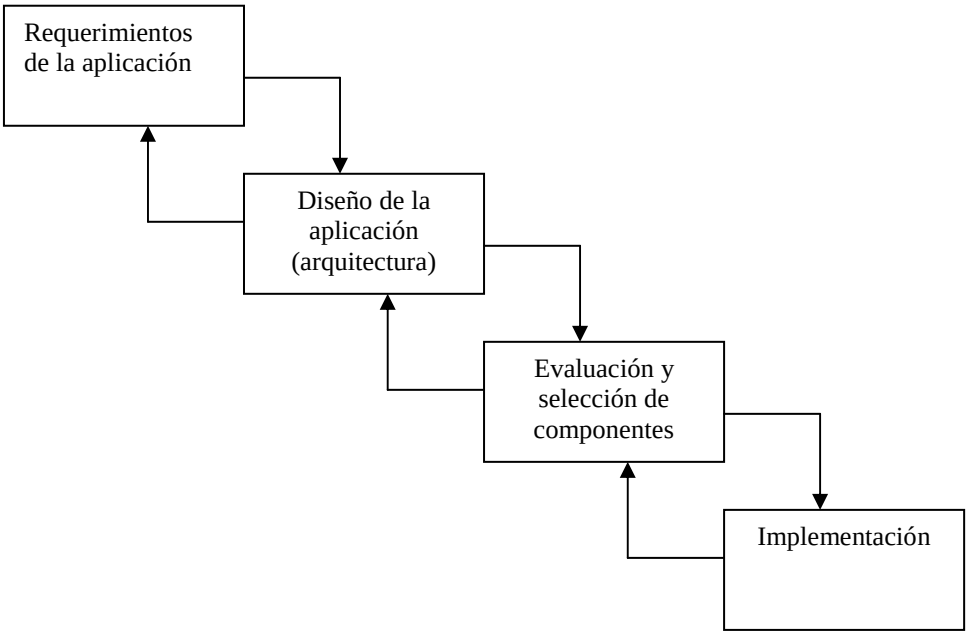

Figura 1 6 Etapas del desarrollo de software basado en componentes comerciales COTS

1.4.5.1 Requerimientos de la aplicación

Para esta fase, el proceso se desarrolla en base al modelo en Espiral[19], la idea de este modelo se sustenta en que el conjunto de los requisitos del sistema no puede ser determinado completamente al inicio de su construcción, y todo intento para establecerlos probablemente fallaría. Para hacer frente a este dilema, el proceso de desarrollo debe hacer uso de frecuentes prototipos de arquitectura del sistema, requerir la presencia permanente

[19] HEINEMAN, G. T. y COUNCILL, W. T. Component-Based Software Engineering. Putting the Pieces Together. Addison-Wesley, 2001

de los usuarios finales del sistema durante todo el desarrollo, y realizar progresivos refinamientos de los prototipos y objetivos del sistema, entre otros aspectos.

En este modelo, las especificaciones de los requisitos inicialmente se recogen a nivel general y se van detallando sistemática y progresivamente a medida que se va construyendo el sistema.

Durante todo este proceso, simultáneamente se realizan rápidos prototipados de la arquitectura del sistema que son continuamente revisados y que pueden alterar el estado de la colección de los requisitos, modificando o desapareciendo requisitos y/o dependencias entre requisitos existentes, o incluyendo algunos nuevos.

1.4.5.2 Diseño de la aplicación

En la fase de diseño de la aplicación, es decir en relación a la arquitectura de software, son numerosas las propuestas existentes de lenguajes para la definición de arquitecturas (LDA). Sin embargo, en el proceso planteado se ha adoptado la notación UML-RT para modelar la descripción de una arquitectura de software. Una de las ventajas que tiene este tipo de LDA, frente a las demás, es que dispone de un conjunto de símbolos que ayudan a crear rápidamente una vista de componentes de la arquitectura de software, eliminando vistas con muchas clases UML.

Las ventajas que llevan a utilizar UML-RT como LDA para el proceso de DSBC-COTS que se propone son:

a) Como la tendencia actual es utilizar UML para describir arquitecturas, y dado que UML-RT también lo es, es posible utilizar las representaciones de modelado tradicionales, como los diagramas de clases, diagramas de caso de uso, diagramas de estados o diagramas de secuencias, entre otros.
b) UML-RT adopta la notación original de ROOM (Real-time Object Oriented Modelling) que utiliza un conjunto reducido de notaciones gráficas que cubren todas las necesidades para hacer una representación visual de una arquitectura de software en poco tiempo.

c) Los sistemas basados en componentes COTS suelen requerir un prototipado rápido de la arquitectura de software para permitir continuos refinamientos de la misma. UML-RT permite hacer un prototipado rápido de una arquitectura de software.

Las notaciones que se incluirán para el modelado de sistemas son:

- Notaciones estructurales:
    - Diagramas de clase
    - Diagramas de objetos
- Notaciones orientadas al usuario o funcionales:
    - Diagramas de casos de uso
- Notaciones de comportamiento:
    - Diagramas de secuencias
    - Diagramas de colaboración
    - Diagramas de estado
    - Diagramas de actividad
- Notaciones orientadas a la implementación:
    - Diagramas de componentes
    - Diagramas de despliegue (deployment)

1.4.5.3 Selección y Evaluación de los componentes

Para construir una aplicación de software, el proceso propuesto sugiere que después de que se han definido los componentes en la arquitectura (“componentes abstractos”), éstos se enfrentan a la colección de componentes COTS (“componentes concretos”) disponibles en repositorios de software. Este proceso de enfrentamiento corresponde a un proceso de búsqueda y selección de componentes que produce una lista de componentes candidatos que pueden formar parte de la aplicación: bien porque proporcionan algunos de los servicios requeridos o bien porque cumplen algunos de los requisitos de usuario extra funcionales, como por ejemplo el precio, o limitaciones de seguridad, entre otros.

A partir de esta lista con los posibles componentes candidatos que pueden formar parte de la aplicación, otro proceso se encarga de realizar todas las posibles combinaciones entre estos componentes para construir el sistema. Estas diferentes combinaciones (configuraciones) deben ser generadas para tomar ciertas decisiones sobre: qué configuración es la que mejor se enlaza con las restricciones impuestas en la arquitectura, qué componentes abstractos no han sido solucionados y cuáles de ellos necesitarán ser implementados, y cómo debería cambiarse la arquitectura de software (y en qué casos esto es factible) para incorporar los componentes COTS encontrados; a este proceso se conoce como evaluación de componentes.

1.4.5.4 Implementación

Después de haber realizado los pasos anteriores, los requisitos del sistema son enlazados con los requisitos de la configuración de arquitectura, obtenida y seleccionada en el diseño de la aplicación. Si es necesario, estos requisitos del sistema podrían ser revisados adecuadamente para ajustarlos a las nuevas necesidades del sistema y ser enfrentados de nuevo contra los repositorios de componentes concretos. El ciclo descrito se repite de forma indefinida hasta obtener una configuración de arquitectura de software que cumpla todos los requisitos de usuario de la aplicación, para de esta manera comenzar con la implementación de la misma.

CAPITULO 2 CASO DE ESTUDIO

2.1 INTRODUCCIÓN

En el presente capítulo se va a desarrollar un caso de estudio para construir una aplicación de software con componentes COTS. Para ello, el ámbito de la aplicación estará dentro del campo académico, específicamente en la educación básica.

Es fundamental que se considere desde el punto de vista didáctico el uso de nuevas tecnologías como medios o herramientas al servicio de los procesos de enseñanza-aprendizaje. Su importancia dependerá del tipo de alumno, actividad, contenidos u objetivos con el que se quiera trabajar, considerando siempre la relación que se quiera establecer con los elementos del acto didáctico. Ya que hay que tener en cuenta que los medios por sí mismos no son capaces de mejorar la enseñanza o el aprendizaje, sólo lo hacen atendiendo a la funcionalidad para la que han sido seleccionados y a los requerimientos del propio proceso de enseñanza-aprendizaje en el que deban ser utilizados.

A la hora de realizar una selección de los medios o recursos más adecuados para el proceso de aprendizaje se deben tener en cuenta los siguientes aspectos:

- Grupo de incidencia: cuáles son los usuarios potenciales del proceso (edad, nivel educativo, conocimiento que tienen sobre los recursos, habilidades básicas adquiridas respecto a las Tecnologías de la Información y Comunicación -TIC).
- Objetivos: definir de manera clara y concisa los objetivos que queremos conseguir mediante el uso de los recursos o, como mínimo, definición de los procesos sobre los que pretendemos incidir.
- Tareas: cuáles son las tareas que los alumnos deberán desempeñar mediante el uso de los diferentes medios y recursos.
- Medios: ¿Cuáles son los medios que tenemos disponibles? ¿Cuál es la infraestructura tecnológica de que disponemos?

- Toma de decisiones: una vez analizados estos cuatro aspectos, se está en disposición de determinar cuáles son los medios y los recursos que se utilizarán. [20]

2.2 REQUERIMIENTOS DE LA APLICACIÓN

Para seleccionar la aplicación que se va a realizar, tomando en cuenta el proceso de aprendizaje, seguiremos los aspectos indicados anteriormente:

- Grupo de incidencia: La aplicación que se desea realizar está dirigida a maestros y estudiantes de educación básica (de segundo a quinto año), es necesario que los niños sepan leer y escribir y que tengan mínimos conocimientos del uso de un computador.
- Objetivos: Ayudar en el aprendizaje de la lengua materna e idiomas extranjeros a nivel inicial, mediante la definición de términos por medio de recursos multimedia (gráficos, sonidos y videos).
- Tareas: Los alumnos deberán tener conocimientos previos sobre el significado de los diferentes términos que el profesor presentará, para poder asociarlo con su respectivo recurso multimedia.
- Medios: Para realizar la aplicación, tanto maestros como alumnos deben tener, como mínimo, acceso a un computador; además, para los maestros es necesario que tengan como material de apoyo, los recursos multimedia que desean utilizar.
- Toma de decisiones: De acuerdo a los aspectos anteriores, se ha determinado que la aplicación que se va a realizar tiene como nombre AMASOP (Aplicación Multimedia de Asociaciones para Profesores), y tiene como objetivo ser una herramienta orientada a maestros de educación básica que les permita realizar sus propias actividades multimedia para elaboración de asociaciones, donde una asociación es una actividad que implica descubrir la relación entre elementos de dos conjuntos distintos. La información que hay que vincular puede presentarse

[20] IUP (INSTITUTO UNIVERSITARIO DE POSTGRADO), Maestría en Nuevas tecnologías Aplicadas a la Educación, Módulo 2: Las nuevas tecnologías en el diseño curricular, marzo 2006.

como texto, gráficos, sonidos o vídeo.[21] Así por ejemplo, se puede asociar el nombre de un animal con su imagen.

Para la fase de especificación de requerimientos de software, el documento de la IEEE es un pilar importante y se seguirán las recomendaciones del estándar IEEE830-1998.

2.3 USO DEL ESTANDAR IEEE830-1998

2.3.1 ALCANCE

Ya que el caso de estudio que se va a realizar pretende dar un enfoque claro y breve del uso de componentes COTS, se va a desarrollar una pequeña aplicación (AMASOP), que será una herramienta que permita a los educadores la realización de una actividad didáctica multimedia conocida como: elaboración de asociaciones, la cual puede contener texto, gráficos, sonidos y videos, que permitirán obtener una actividad muy atractiva y motivadora en el aprendizaje de idiomas.

2.3.2 FUNCIONES DEL PRODUCTO

AMASOP será diseñada para colaborar con los educadores en las actividades didácticas en lo que se refiere a elaboración de asociaciones, cuyo objetivo es descubrir la relación entre elementos de dos conjuntos distintos.

En forma general, las funciones que cumplirá la aplicación son:

- Realizar actividades multimedia que contengan información a ser vinculada, la cual puede presentarse como texto, gráficos, sonidos, animaciones y videos.
- Ser una aplicación de escritorio y fuera de línea, que trabaje en un ambiente Windows.

2.3.3 CARACTERÍSTICAS DEL USUARIO

[21] IUP (INSTITUTO UNIVERSITARIO DE POSTGRADO), Maestría en Nuevas tecnologías Aplicadas a la Educación, Módulo 4: El uso didáctico de la red Internet, marzo 2006.

AMASOP va a estar dirigida a un solo tipo de usuario: el docente, quien realizará la actividad multimedia. Por esto, no será necesario manejar perfiles de usuario.

2.3.4 RESTRICCIONES

2.3.4.1 Físicas

La aplicación debe ser instalada en la máquina del usuario para que éste pueda utilizarla.

2.3.4.2 Lógicas

AMASOP puede ser utilizado a cualquier hora del día y sin ningún tipo de restricción, pues es una aplicación de escritorio y fuera de línea.

2.3.4.3 De acceso al Sistema

Para ingresar a la aplicación no se necesitará de un login y password, sino que el usuario accede directamente al sistema.

2.3.4.4 De comercialización del Sistema

Esta aplicación será de libre difusión y puede ser utilizada por cualquier persona que así lo requiera.

2.3.5 ASUNCIONES Y DEPENDENCIAS

Los siguientes factores son imprescindibles para el correcto desarrollo y futuro desempeño de AMASOP:

- El funcionamiento correcto del hardware en el cual se usará el programa.
- Sistema Operativo Windows 98 o superior
- Licencia de Microsoft Visual Studio .Net
- Microsoft Office 2000 o superior.

- Licencia para Rational Rose Enterprise 2003.
- Software necesario para visualización de imágenes y de video.
- Hardware para multimedia.

2.3.6 REQUISITOS ESPECÍFICOS

Dentro de los requisitos específicos de la aplicación, se determinarán todos los requerimientos que permitirán a los diseñadores modelar la aplicación, de tal manera que satisfaga dichos requerimientos.

Es importante mencionar que ya que en el proceso de desarrollo el modelo a seguir es en espiral, la especificación de requisitos puede cambiar o evolucionar según progresa el proceso de desarrollo del software.

2.3.6.1 Interfaces de usuario

AMASOP pondrá toda su funcionalidad a través de una interfaz gráfica que tendrá botones de comando, de selección y cajas de texto.

2.3.6.1.1 Interfaces hardware

Cuenta con las siguientes interfaces en HW:

- Monitor
- Teclado
- Mouse
- CPU
- Parlantes

2.3.6.1.2 Interfaces software

La aplicación será desarrollada en Visual Studio .Net. Para el diseño de la misma se utilizará las herramientas de Rational Rose para modelado Una vez implementada, funcionará en un ambiente Windows. De ser necesarias otras interfaces de software de acuerdo a los componentes que se vayan a utilizar para la carga y visualización de imágenes o videos, se indicarán las mismas en la fase de Selección y Evaluación de componentes.

2.4 DISEÑO DE LA APLICACIÓN

Para realizar el diseño de AMASOP, de acuerdo a los requerimientos establecidos, se han escogido los siguientes diagramas UML.

- Diagramas de Caso de Uso
- Diagramas de Secuencia
- Diagrama de Componentes
- Diagrama de Despliegue

2.4.1 DIAGRAMAS DE CASO DE USO

2.4.1.1 Especificación de casos de uso de la aplicación

Los actores que se tendrán para esta aplicación son los profesores, quienes manejarán la herramienta, y los estudiantes que utilizarán el producto que es devuelto por la aplicación.

2.4.1.1.1 Diagrama de Casos de Uso para el actor profesor

El actor profesor será el principal usuario de la aplicación, se encargará de proveer toda la información necesaria para la actividad de asociación y de generar el ejecutable para que el estudiante pueda proceder a realizar la asociación.

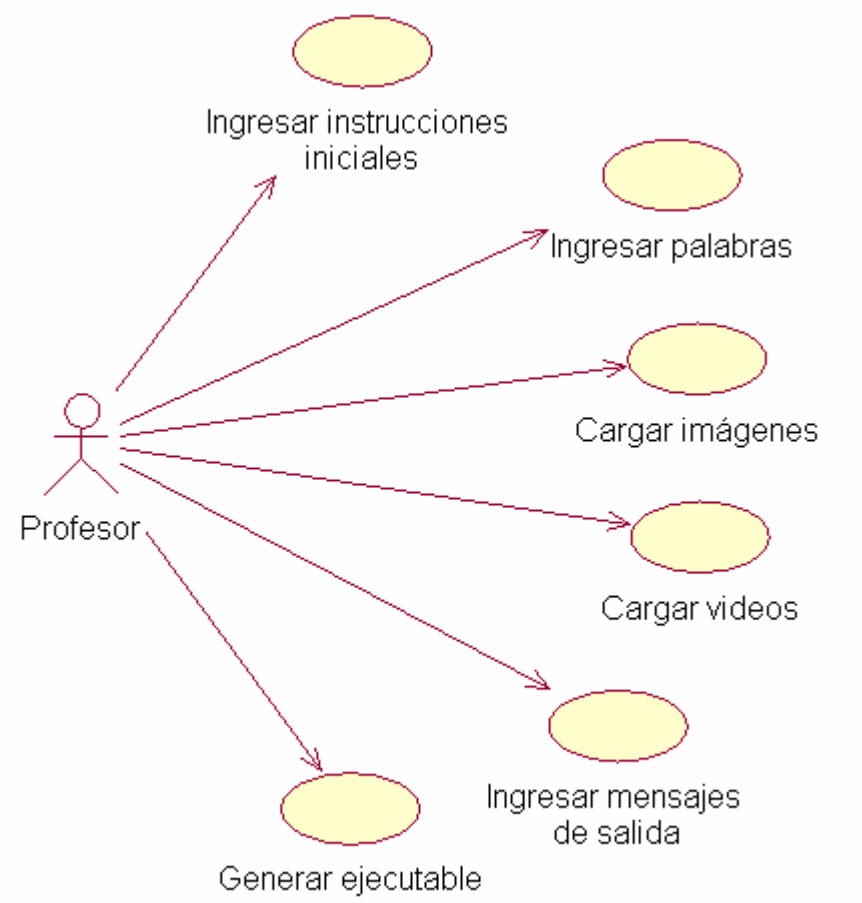

Figura 2 1 Diagrama de Casos de Uso para el actor Profesor

CASO DE USO 01: **Ingresar instrucciones iniciales**
ACTORES: **Profesor**
PRE-CONDICIÓN: **Tener claro el tema sobre el cual se desea realizar la asociación.**
POS-CONDICIÓN: **Registrar en la base de datos la instrucción de acuerdo al tema de la asociación.**
DESCRIPCIÓN: **En este caso de uso se debe ingresar las instrucciones que el alumno recibirá para iniciar con la actividad, esta instrucción deberá indicar claramente lo que el alumno tendrá que hacer.**

Tabla 2. 1 Descripción del caso de uso: Ingresar instrucciones iniciales

CASO DE USO 02: **Ingresar palabras**
ACTORES: **Profesor**
PRE-CONDICIÓN: **Escoger el código de la asociación a la cual va a pertenecer cada término. Saber la cantidad y qué términos se van a utilizar en la asociación.**
POS-CONDICIÓN: **Cada término deberá estar asociado a su respectivo recurso multimedia para que puedan ser registrados en la base de datos.**

DESCRIPCIÓN: **Para realizar la asociación de un texto con una imagen o un video, será necesario escribir la palabra o frase que se refiera a la imagen o el vídeo que se presenten.**

Tabla 2. 2 Descripción del caso de uso: Ingresar palabras

CASO DE USO 03: **Cargar imágenes**
ACTORES: **Profesor**
PRE-CONDICIÓN: **Haber ingresado el término correspondiente a la imagen que se va a cargar. Las imágenes deben estar en un formato compatible con el visor de imágenes.**
POS-CONDICIÓN: **Si es la primera vez que se ingresó un término, se debe cargar una imagen, para los demás términos se cargarán solamente imágenes.**
DESCRIPCIÓN: **Si se desean utilizar imágenes en la asociación, es necesario cargar en la aplicación dicha imagen vinculada al texto correspondiente, para que de esta manera, la imagen sea almacenada.**

Tabla 2. 3 Descripción del caso de uso: Cargar imágenes

CASO DE USO 04: **Cargar videos**
ACTORES: **Profesor**
PRE-CONDICIÓN: **Haber ingresado el término correspondiente al video que se va a cargar. Los videos deben estar en un formato compatible con el visor de videos.**
POS-CONDICIÓN: **Si es la primera vez que se ingresó un término, se debe cargar un video, para los demás términos se cargarán solamente videos.**
DESCRIPCIÓN: **Para utilizar videos, el proceso es similar a las imágenes, es decir, es necesario en la aplicación cargar el video que se vaya a utilizar y vincularlo a la frase o texto, de ésta manera el video será almacenado.**

Tabla 2. 4 Descripción del caso de uso: Cargar videos

CASO DE USO 05: **Ingresar mensajes de salida**
ACTORES: **Profesor**
PRE-CONDICIÓN:
POS-CONDICIÓN: **Registrar los mensajes de salida.**
DESCRIPCIÓN: **Se deben escribir dos mensajes de salida por cada asociación, los cuáles se usarán dependiendo de si el alumno realizó o no correctamente la actividad.**

Tabla 2. 5 Descripción del caso de uso: Ingresar mensajes de salida

CASO DE USO 06: **Generar ejecutable**
ACTORES: **Profesor**
PRE-CONDICIÓN: **Haber registrado correctamente en la base de datos, las instrucciones de entrada, los textos, las imágenes, los videos y los mensajes de salida de la asociación.**
POS-CONDICIÓN: **Verificar que la actividad de asociación realizada, se ajuste al proceso de enseñanza, de tal manera que el producto de la aplicación sirva como un programa de ejercitación para los alumnos y como una tarea más de aprendizaje o de evaluación. Verificar que el producto obtenido sea el deseado.**
DESCRIPCIÓN: **El momento que todos los archivos estén cargados, se procede a realizar una vista previa del producto final, si ésta es de satisfacción del profesor, se procederá a generar ejecutable.**

Tabla 2. 6 Descripción del caso de uso: Generar ejecutable

2.4.1.1.2 Diagrama de Casos de Uso para el actor estudiante

El estudiante será el encargado de correr el ejecutable de la actividad para así realizar la asociación.

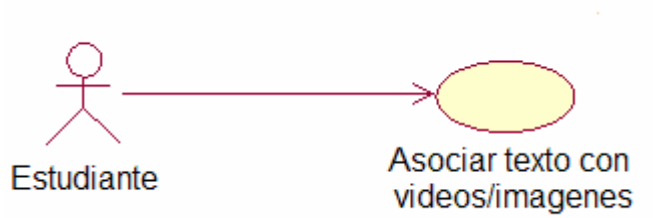

Figura 2 2 Diagrama de Casos de Uso para el actor estudiante

CASO DE USO 07: **Asociar texto con videos/imágenes**
ACTORES: **Estudiante**
PRE-CONDICION: **Tener el ejecutable de la actividad e instalado el software extra que sea necesario. Tener el conocimiento previo sobre el tema del cual trata la asociación.**
POS-CONDICION: **En caso de que la actividad no haya sido realizada correctamente, se deberá revisar nuevamente el tema.**
DESCRIPCION: **De acuerdo a la instrucción que se dé, se debe proceder con la asociación de todos los textos con los videos o imágenes respectivos. De esta manera se finalizará con el uso de la aplicación.**

Tabla 2. 7 Descripción Caso de Uso: Asociar texto con videos/imágenes

2.4.2 DIAGRAMAS DE SECUENCIA

2.4.2.1 Diagrama de secuencia para el actor Profesor

La secuencia de mensajes entre la aplicación y el actor profesor, será de la siguiente manera: El profesor deberá ingresar las instrucciones que querrá dar a su alumno de acuerdo al tema de la asociación que va a realizar. Debe proceder a cargar los textos, imágenes y videos respectivos y finalmente ingresar los mensajes que mostrará a su alumno cuando éste haya finalizado la asociación ya sea de manera correcta o incorrecta. De esta manera, procederá a realizar una vista previa de la asociación, de acuerdo a los archivos cargados, se procede a aceptar esta vista y finalmente se devolverá un mensaje indicando que el ejecutable ha sido generado.

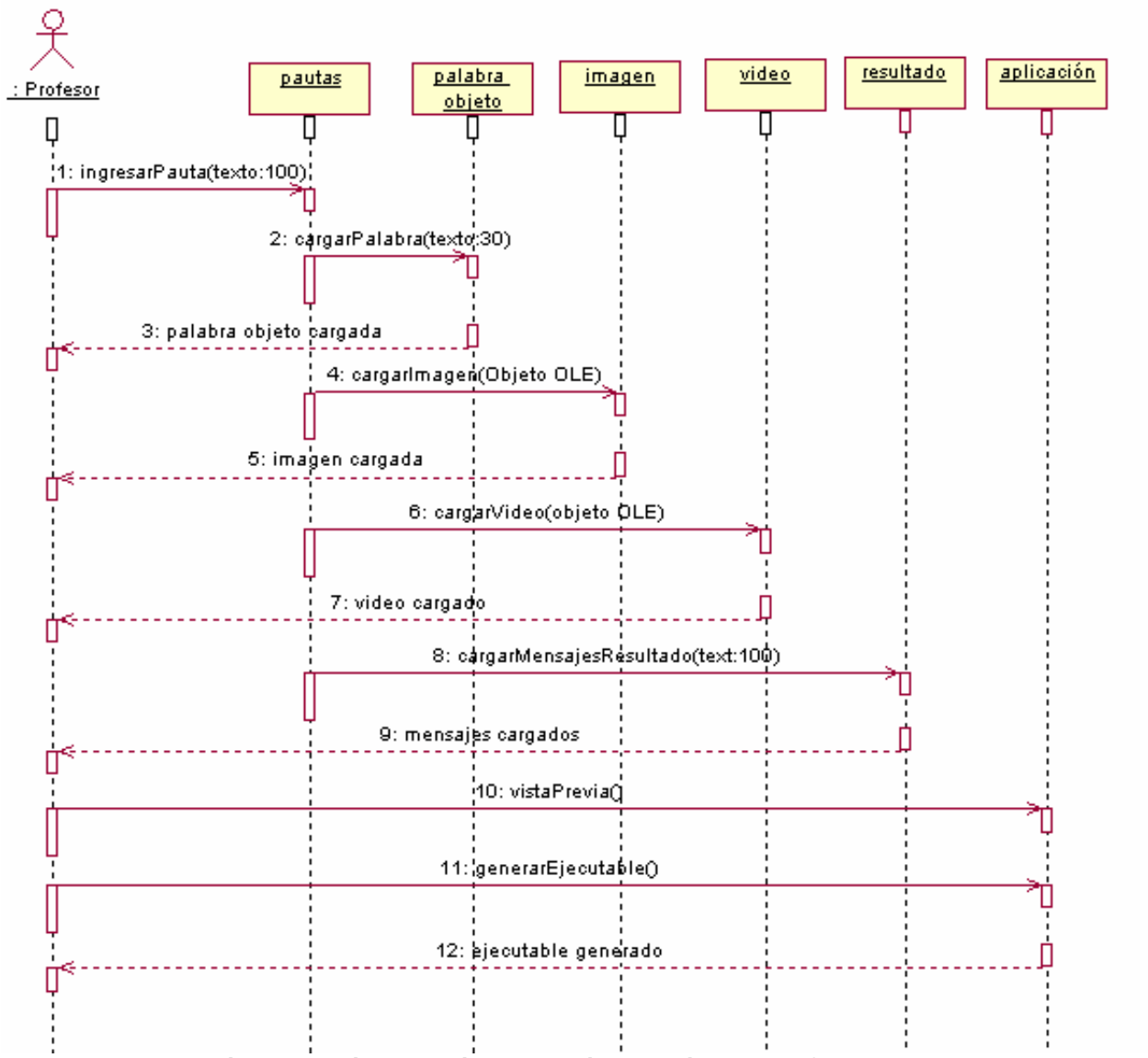

Figura 2 3 Diagrama de secuencia para el actor Profesor

2.4.2.2 Diagrama de secuencia para el actor Estudiante

La secuencia de mensajes entre la aplicación y el actor estudiante será de la siguiente manera: El alumno, al correr el ejecutable, ingresará directamente a ver las columnas de texto y video o imágenes que va a tener que asociar. Cuando hay finalizado, la aplicación verificará si la asociación ha sido o no correctamente realizada, devolviendo un mensaje en cada uno de los casos.

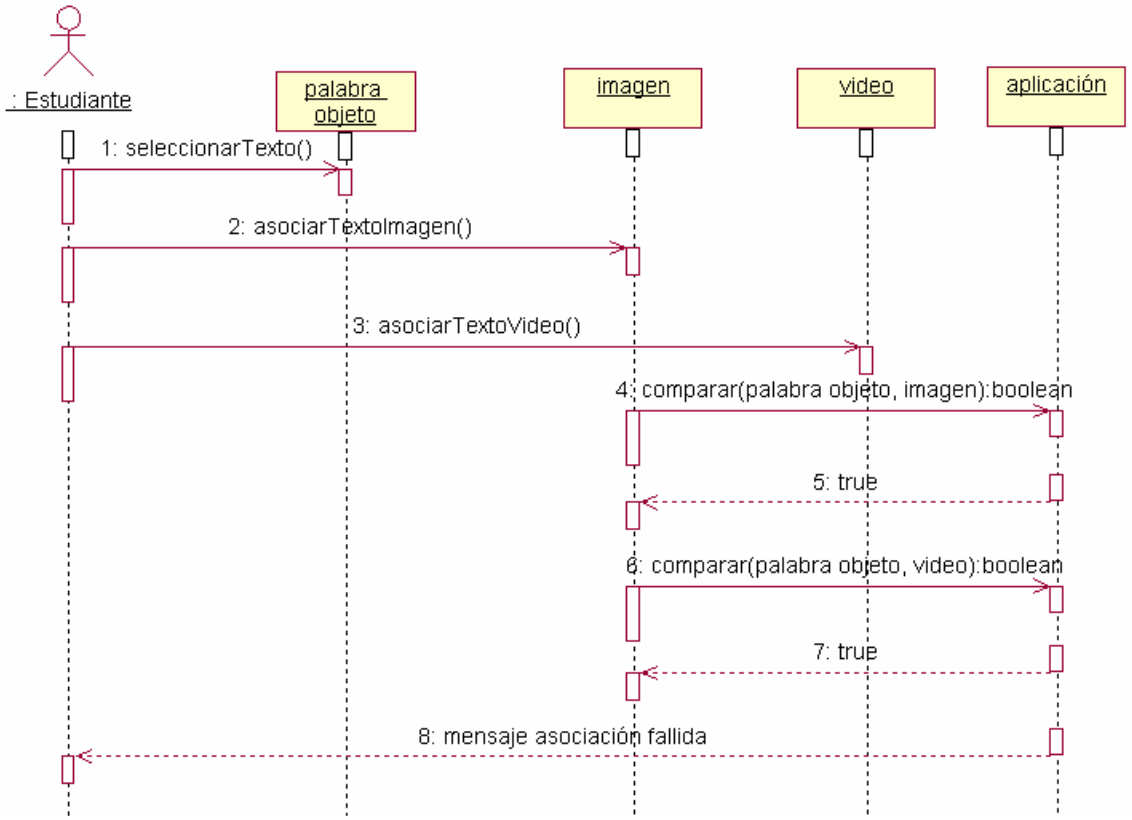

Figura 2 4 Diagrama de Secuencia Realizar Asociación para el actor Estudiante - Asociación exitosa

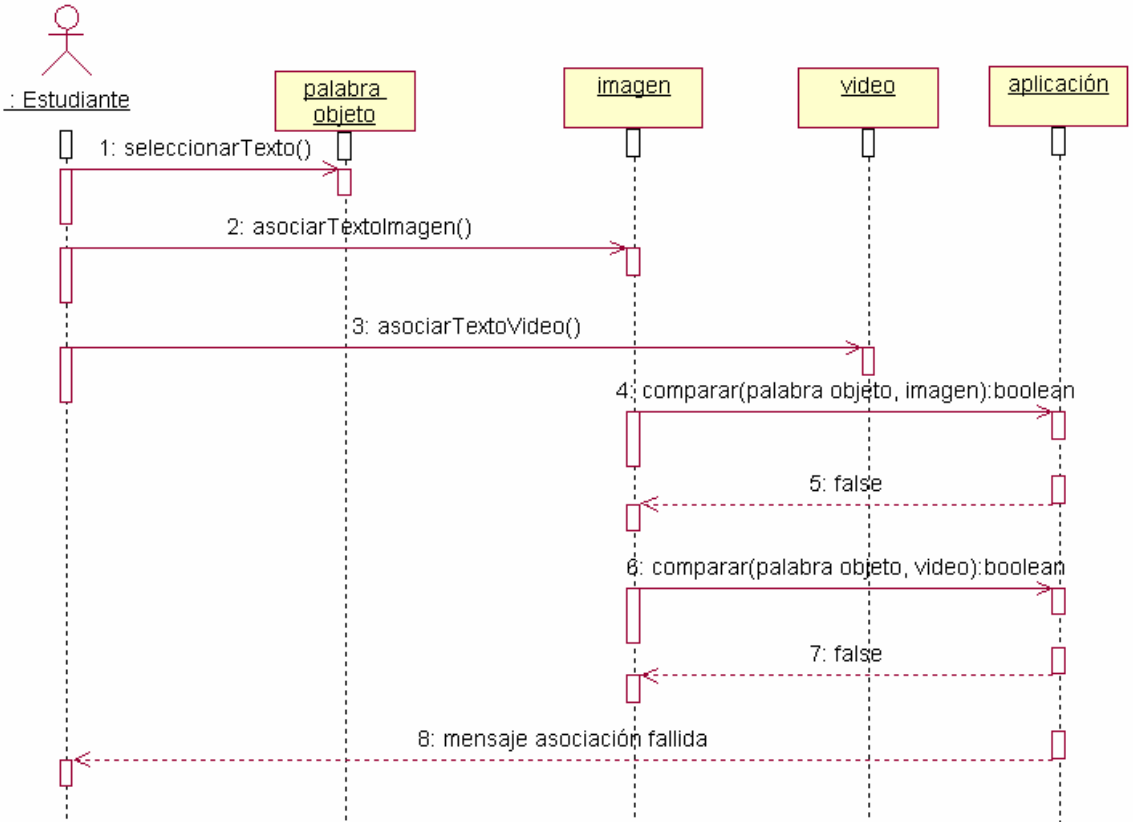

Figura 2 5 Diagrama de Secuencia Realizar Asociación para el actor Estudiante – Asociación fallida

2.4.3 DIAGRAMA DE COMPONENTES

La aplicación estará formada por dos sub-programas: amasop.gui y amasop_exe.gui. El subprograma amasop.gui es el módulo que está dirigido al profesor, mientras que el subprograma amasop_exe.gui es el módulo orientado a los alumnos. Los dos módulos necesitarán un componente que permita visualizar las imágenes y los videos que serán cargados por el profesor. En la etapa de evaluación y selección se definirán los componentes que serán utilizados para los fines deseados.

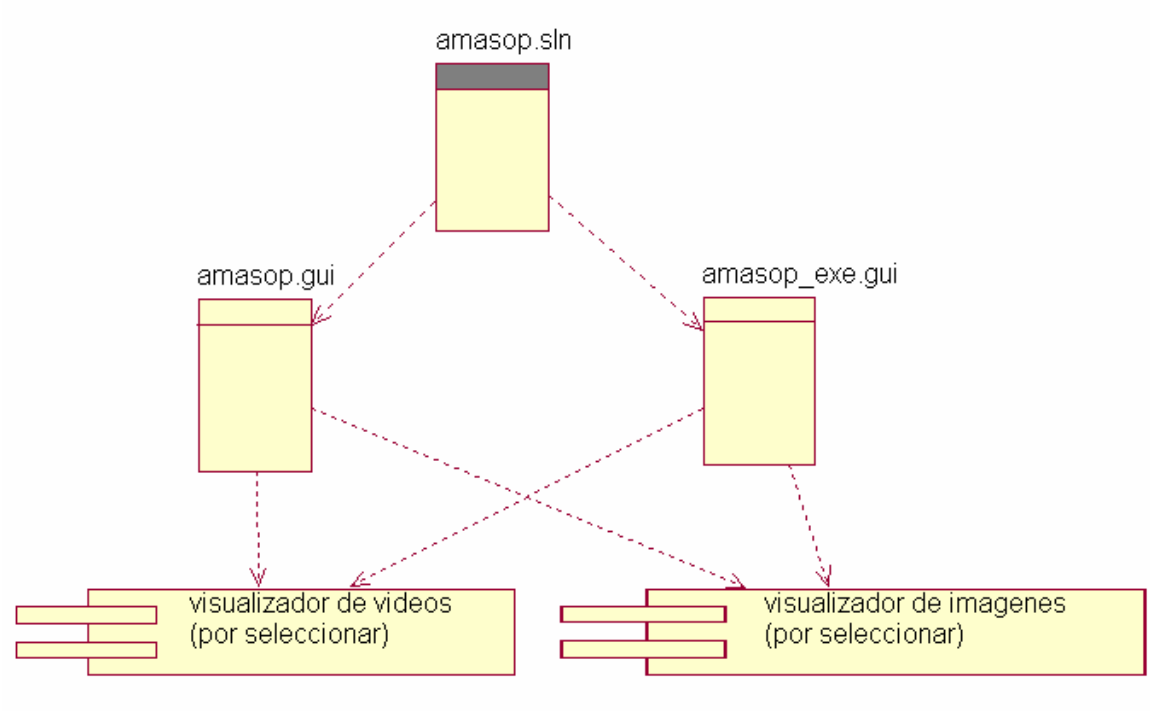

Figura 2 6 Diagrama de componentes

2.4.4 DIAGRAMA DE DESPLIEGUE

La aplicación tendrá como nodos de comportamiento, un cliente, en donde se necesitará tener el componente para lectura de videos.

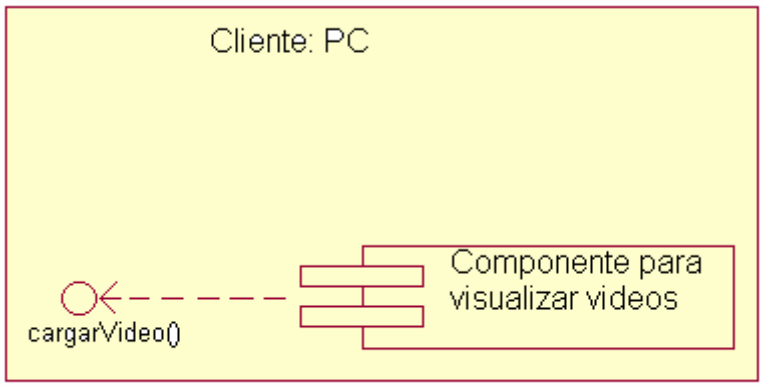

Figura 2 7 Diagrama de Despliegue

2.5 SELECCIÓN Y EVALUACIÓN DE COMPONENTES

En esta etapa se pretende realizar la búsqueda de los componentes que cumplan con los requisitos que la aplicación necesita. Con los componentes candidatos, se procede a realizar una evaluación para escoger cuál se ajusta mejor a los requerimientos tanto funcionales como extra funcionales de la aplicación.

Para la aplicación van a ser necesario dos tipos de componentes: en primer lugar, un componente que permita mostrar imágenes, y otro que permita visualizar videos. Una de las principales características que se buscan en los componentes, es que permitan la visualización de la mayor cantidad de formatos, tanto para imágenes o videos.

Los componentes candidatos para el caso de imágenes son:

- **ImgX Controls - ActiveX/DLL - V6.4[22]:** ImgX controls para Visual Basic y .NET, contienen un conjunto de controles ActiveX para la visualización y manipulación de imágenes, impresión, exploración, captura y más. Proporciona herramientas necesarias para la proyección de imágenes de alta calidad. Costo: 361.62usd.

[22] Componente seleccionado del repositorio http://www.componentsource.com/features/ximagp/index.html, noviembre 2006.

- **LEADTOOLS Raster Imaging Pro for .NET .NET - V14.5[23]:** Extiende el framework de .NET proporcionando soporte para más de 150 formatos de archivo incluyendo: Tiff, JPG, J2K, pdf, y GIF usando varios esquemas de compresión comovJPEG, JPEG2000, LZW, CCITT G3/G4, yvCMP. Costo: 835.80 usd.
- **Control PictureBox de .Net:** El control PictureBox de formularios Windows Forms se utiliza para mostrar gráficos en formato de mapa de bits, GIF, JPEG, metarchivo o icono. Costo: Incluido en Visual Studio .Net 2005.

Los componentes candidatos para el caso de videos son:

- **LEADTOOLS Multimedia - ActiveX/COM Object - V15.0[24]:** Para crear aplicaciones multimedia profesionales y de alta calidad. Leadtools Multimedia permite agregar a las aplicaciones funcionalidades multimedia tal como captura, juego, y corrección. Permite el uso de formatos WAV, AVI, ASF, WMA, WMV, MPEG-1, MPEG-2, OGG y más. Costo: 565.25 usd.
- **AutoPlay Media Studio - Application - V6.0[25]:** Para presentaciones interactivas, aplicaciones Windows, y mucho más. Es una herramienta de desarrollo de gran alcance que permitirá integrar fácilmente audio, vídeo, imágenes, texto, flash de Macromedia, y más. Costo: 485.10
- **Windows Media Series:** Windows Media no es únicamente un conjunto de herramientas y componentes individuales, sino que constituye una plataforma integral a partir de la cual los fabricantes independientes de software pueden crear sus propias soluciones de medios digitales. El Kit de desarrollo de software (SDK) de Windows Media Series proporciona herramientas que los programadores necesitan para crear sus propias soluciones en esta eficaz plataforma nueva. Permite la visualización de archivos asf, wma, wmv, dvd, avi, wav, mp3, mpeg, midi y más. Costo: Gratuito.

[23] Componente seleccionado del repositorio http://www.componentsource.com/products/514866/16556/summary.html, noviembre 2006.
[24] Componente seleccionado del repositorio http://www.componentsource.com/products/512633/18779/index.html, noviembre 2006.
[25] Componente seleccionado del repositorio http://www.componentsource.com/products/504787/15470/index.html, noviembre 2006.

Del grupo de componentes candidatos, se escogieron como componentes a ser utilizados a: Control PictureBox de .Net para el caso de las imágenes, y Windows Media Series para el caso de los videos, esto atendiendo a que uno de los requisitos no funcionales es el costo de los componentes, y ya que éstos son gratuitos, no se incurren en costos adicionales. Además cumplen con los requerimientos de la aplicación.

2.5.1 CARACTERÍSTICAS DE LOS COMPONENTES SELECCIONADOS

A pesar de que el costo será un requerimiento importante en la realización de AMASOP, no es el único que se debe cumplir, es por esto que a continuación se va a describir las características de los componentes seleccionados para verificar que cumplen los requisitos funcionales y de integración.

- **Control PictureBox de .Net:** El componente Control PictureBox es adecuado para ser usado en la aplicación que se va a desarrollar, pues éste permite mostrar gráficos de un archivo de mapa de bits, metarchivo, icono, JPEG, GIF o PNG. Además, el componente cumple con los requerimientos para el desarrollo de la aplicación, ya que permite agregar, mover, y cambiar el tamaño de las imágenes en tiempo de ejecución, y además, copiar imágenes seleccionadas directamente desde una base de datos.

  Este control presenta varias propiedades que pueden ser de gran utilidad al momento de desarrollar la aplicación, entre ellas tenemos que:

  - Se establece la propiedad Image en el objeto Image que se desea mostrar, ya sea en tiempo de diseño o en tiempo de ejecución.

  - Además, se puede especificar la imagen estableciendo la propiedad ImageLocation.
  - Cargar la imagen de forma sincrónica utilizando el método Load o de forma asincrónica mediante el método LoadAsync.
  - La propiedad SizeMode, establecida en los valores en la enumeración PictureBoxSizeMode, controla el recorte y la posición de la imagen en el área de presentación.

- Se puede cambiar el tamaño del área de presentación en tiempo de ejecución con la propiedad ClientSize.
- El control PictureBox se muestra de forma predeterminada sin bordes. Para proporcionar un borde estándar o tridimensional, se utiliza la propiedad BorderStyle con el fin de distinguir el cuadro de imagen del resto del formulario, incluso si no contiene ninguna imagen.
- PictureBox no es un control seleccionable, lo que significa que no puede recibir el foco de entrada.

El control PictureBox, es utilizado en plataformas Windows 98, Windows NT 4.0, Windows Millennium Edition, Windows 2000, Windows XP Home Edition, Windows XP Professional Edition, familia de Windows Server 2003, .NET Compact Framework - Windows CE .NET.

La integración con aplicaciones realizadas en Visual Studio .Net se lo hace mediante la biblioteca System.Windows.Forms (en System.Windows.Forms.dll).

Una de las características importantes de PictureBox es que, permite copiar una imagen almacenada directamente en una base de datos en un control PictureBox de un formulario de Windows Forms, sin tener que guardar la imagen en un archivo.

El conjunto de características descritas, la facilidad de integración, y el hecho de estar incluido en Visual Studio .Net, han sido las razones por las cuales se escogió al control PictureBox, como el componente para ser utilizado en AMASOP.

- **Windows Media Series:** Se decidió utilizar el componente Windows Media Series para la visualización de videos, pues cumple con los requerimientos de AMASOP, es decir: permite la visualización de archivos de video con extensiones dvd, vcd, avi, wmv y mpeg. Además, con este componente se tiene acceso a todas las funcionalidades de audio y video presentes en Windows Media Placer. Finalmente, cumple con el requerimiento más importante, pues este componente es completamente integrable en cualquier aplicación desarrollada en Visual Studio .Net.

Algunas características adicionales del componente seleccionado son que el kit de desarrollo de software (SDK) de Windows Media Series proporciona herramientas que los programadores necesitan para crear sus propias soluciones en esta eficaz plataforma. El SDK de Windows Media Series incluye los componentes siguientes:

- SDK de Formato de Windows Media
- SDK del Codificador de Windows Media
- El SDK del Reproductor de Windows Media
- El SDK de los Servicios de Windows Media

La integración de Windows Media Series en las herramientas de creación de contenido y de reproducción se puede conseguir de, al menos, dos formas distintas. Una primera opción es que un fabricante independiente de software utilice el SDK de Formato de Windows Media Series para integrar a un nivel importante los códecs[26] Windows Media Audio Series y Windows Media Video Series en su aplicación. De esta manera, se obtiene una eficacia y una flexibilidad inigualables, a la vez que se evita que los programadores deban crear los códecs, y editar y formatear el acceso de cero. Estos SDK son tan eficaces que Microsoft los ha utilizado para desarrollar los componentes que manipulan y reproducen archivos de Windows Media y otros formatos en el sistema operativo Windows XP y el Codificador de Windows Media Series, además de una serie de aplicaciones independientes.

Una segunda opción para el desarrollo de las herramientas de creación de contenido es que un programador utilice el SDK del Codificador de Windows Media Series. El conjunto de interfaces de este SDK facilita al programador la tarea de crear secuencias de comandos para el comportamiento del propio Codificador. Es una manera fácil de crear una eficaz aplicación de creación de medios a partir del Codificador de Windows Media Series, y además resulta ideal para crear herramientas de codificación por lotes o aplicaciones de codificación simplificadas para usuarios finales.

[26] Códec: Códec es una abreviatura de Codificador-Decodificador. Describe una especificación implementada en software, hardware o una combinación de ambos, capaz de transformar un archivo con un flujo de datos o una señal.

Los Servicios de Windows Media Series ofrecen capacidades enriquecidas y funcionalidades clave para la distribución del contenido multimedia digital. La flexible arquitectura de componentes de los Servicios de Windows Media Series permite a los programadores de software y a los profesionales de IT ampliar las capacidades nativas a través de componentes personalizados y la creación de aplicaciones nuevas para garantizar así la integración de los Servicios de Windows Media Series en sistemas y soluciones ya existentes. Pueden ser utilizados en aplicaciones que corran sobre Microsoft Windows 98, Windows Millennium Edition (Me), Windows 2000, y Windows XP.

Windows Media Series ofrece la mayor plataforma para desarrollar soluciones de Digital Media. La plataforma ofrece una alta escalabilidad, fiabilidad y fácil administración. Su arquitectura es extensible, permitiendo construir funcionalidades e integrando tecnología en toda la solución.

Se puede incluir Windows Media con una gran variedad de tecnologías, pudiendo ser:

- Navegadores Web de HTML. Internet Explorer y Netscape Navigator versión 4.7 o posteriores que estén soportados.
- Programas creados con Microsoft Visual C++.
- Programas basados en Microsoft Foundation Classes (MFC).
- Programas creados utilizando Microsoft .NET, Incluyendo programas escritos en lenguaje de programacion C#.
- Microsoft Office, etc.

CAPITULO 3 EVALUACIÓN DEL CASO DE ESTUDIO

En el presente capítulo se va a realizar una evaluación de calidad de la aplicación realizada en el caso de estudio y de su proceso de desarrollo.

3.1 PARÁMETROS DE EVALUACIÓN

Para obtener los parámetros con los cuales se evaluará el caso de estudio, se utilizará como método de evaluación al Modelo Sistémico de Calidad (MOSCA), el cual integra el modelo de Calidad del Producto y el modelo de Calidad del Proceso de Desarrollo, soportado en los conceptos de la Calidad Total Sistémica. A continuación se hará una descripción del método MOSCA para la medición de la calidad de los sistemas de software.

3.1.1 MODELO SISTÉMICO DE CALIDAD (MOSCA)

3.1.1.1 Matriz De Calidad Global Sistémica

La calidad del proceso garantiza la calidad del producto y consecuentemente no se pueden desligar estas dos calidades. Este es el concepto considerado en el enfoque sistémico propuesto por Callaos y Callaos[27]. La definición de la calidad sistémica de los sistemas se basa en la Matriz de Calidad Global mostrada en la Figura 3.1 , la cual consta de cuatro tipos de calidades y se basa en las dos perspectivas: Proceso y Producto. Estos cuatro tipos de calidades son las consideraciones de los Aspectos Internos y Contextuales de ambas perspectivas. Es decir, Aspectos Internos y Contextuales del Producto y Aspectos Internos y Contextuales del Proceso, considerando además, los puntos de vista del Cliente y del Usuario.

[27] CALLAOS, N. y B. CALLAOS, Designing with Systemic Total Quality, Conferencia Internacional de Sistemas de Información, Orlando, Florida, July 1996, pp. 548-560

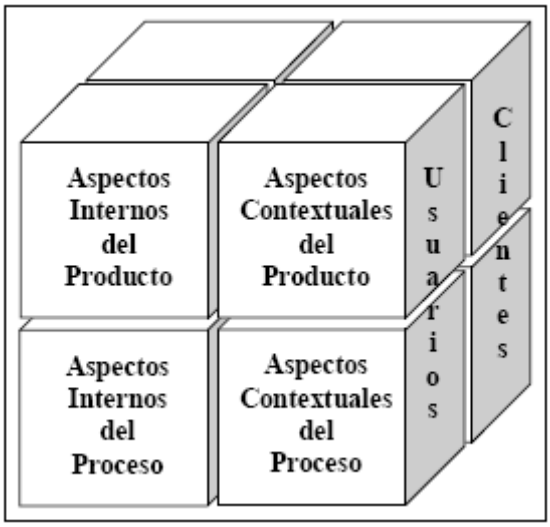

Figura 3 1 Matriz de Calidad Global Sistémica[28]

Esta división se justifica en un sentido, porque un proyecto incluye tanto los aspectos contextuales como los aspectos internos, y en el otro, porque el Sistema concebido (el producto) es diferente al Sistema de las actividades humanas (el proceso) mediante el cual el Sistema-Producto es diseñado. El aporte del enfoque de calidad sistémica es aquel que permite balancear las diferentes perspectivas de la calidad del software (Proceso y Producto).

- **Aspectos Internos del Producto:** Son determinados por actividades de diseño interno y programación, ya que un producto eficiente es conseguido cuando se aplican las prácticas correctas de diseño físico y programación.
- **Aspectos Contextuales del Producto:** Son determinados por las actividades de identificación de requerimientos, diseño de interfaces y diseño general de la red (ubicación de puntos), debido a que la misma está relacionada con la adecuación y confort del usuario.
- **Aspectos Internos del Proceso:** Están asociados con las actividades de gerencia de proyectos, las cuales incluyen el cumplimiento de fechas de entrega, aumento de la productividad y ahorro de recursos.

[28] CALLAOS, N. y B. CALLAOS, Designing with Systemic Total Quality, Conferencia Internacional de Sistemas de Información, Orlando, Florida, July 1996, pp. 548-560

- **Aspectos Contextuales del Proceso:** Se relacionan con las actividades generales de gerencia, tales como liderazgo, administración de cambio, relaciones humanas y grupales, ya que las mismas conducen a establecer buenas relaciones entre los integrantes del equipo responsable del desarrollo de software.

Según Callaos y Callaos, la calidad global no es la suma de las calidades parciales, sino el compromiso entre todo el conjunto de calidades que conlleve a un óptimo global con cierto sacrificio de los óptimos parciales. Resumiendo, se tiene que la calidad del software no es algo que depende de una sola característica en particular, sino que obedece al compromiso de todas sus partes. A su vez, permite englobar las dos tendencias actuales de modelos de calidad; estas son calidad del producto (software) y calidad del proceso, con un enfoque sistémico.

3.1.1.2 Modelo De Calidad Del Producto De Software Con Un Enfoque Sistémico

El modelo para la calidad del producto se basa en el Modelo de Calidad Sistémica de Callaos y Callaos. Los componentes que son tomados en cuenta en el modelo de calidad del producto son los siguientes:

- Los aspectos internos y contextuales del producto como calidad parcial del modelo de Calidad Sistémica de Callaos.
- Las características de calidad del modelo de Dromey y el estándar internacional ISO/IEC 9126: Eficiencia, Fiabilidad, Funcionalidad, Mantenibilidad, Portabilidad y Usabilidad.
- La relación utilizada en el modelo de McCall entre los atributos y calidad de las métricas.

Este modelo es mostrado en la Figura 3.2 y plantea, sobre la base de las 6 características de calidad del estándar internacional ISO/IEC 9126, un conjunto de sub-características y métricas asociadas que miden la calidad de un producto de software con un enfoque sistémico. Estas métricas (249 en total) propuestas, son el resultado del análisis de las cuatro dimensiones de la Matriz de Calidad Sistémica de Callaos que están asociadas a cada característica. Aunque el modelo soporta el enfoque sistémico propuesto por Callaos

y Callaos, el modelo se orienta únicamente a la medición de la calidad del producto de software omitiendo la calidad del proceso. A pesar de ello, el modelo menciona algunas sub-características de cada métrica asociada al proceso de desarrollo, más no trata esta dimensión en su totalidad.

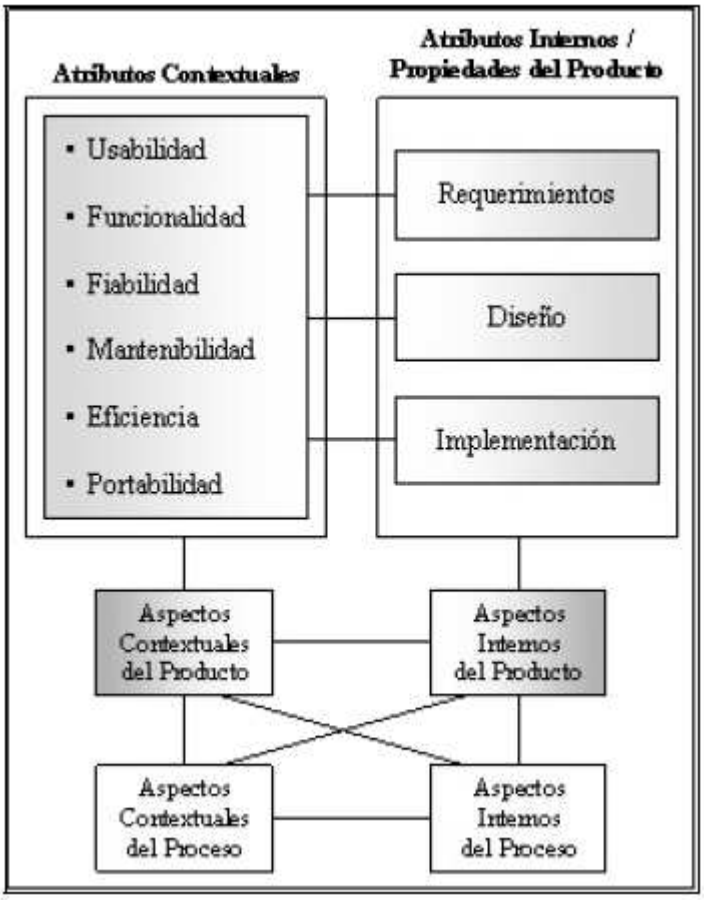

Figura 3 2 Modelo de Calidad del Producto de Software

Las 6 características del modelo, aunados a cada una de las sub-características, hacen del mismo un instrumento de medición de gran valor ya que cubre todos los aspectos necesarios e imprescindibles para medir directamente la calidad del producto de software; esto se debe a su similitud con el estándar internacional ISO/IEC 9126. Cabe acotar que, no todas las métricas se usan en todos los casos, sino que el modelo debe ser instanciado dependiendo de las características del producto deseado.

3.1.1.3 Modelo De Calidad Del Proceso De Software Con Un Enfoque Sistémico

Al igual que en el modelo de calidad del producto explicado anteriormente, éste también está relacionado con el modelo de Calidad Sistémica de Callaos, con el objetivo específico

de soportar el enfoque de Calidad Global Sistémica para las dos dimensiones asociadas al proceso (interna y contextual). Este modelo y sus características están basados en la adaptación del modelo del proceso SPICE -conocido también como ISO/IEC 15504 al modelo de calidad sistémica.

En la Figura 3.3 se ilustra el modelo del proceso, el cual presenta una estructura compleja que está definida por niveles, donde cada nivel superior está conformado por elementos del nivel inferior.

Este modelo consta de 4 niveles:

- **Nivel 0:** Ciclos de Vida: Ciclo de Vida Primario, Ciclo de Vida de Apoyo y Ciclo de Vida Organizacional.
- **Nivel 1:** Categorías de Procesos: Categoría Cliente-Proveedor (CUS), Categoría Ingeniería (ENG), Categoría de Soporte (SUP), Categoría de Gestión (MAN) y Categoría Organizacional (ORG).
- **Nivel 2:** Procesos. Cada categoría contiene un conjunto de procesos característicos, los cuales definen las áreas claves a satisfacer para lograr, asegurar, mantener y controlar la calidad.
- **Nivel 3:** Principios. Cada proceso tiene asociado un Principio (P), el cual se define como característica abstracta y genérica de la organización que sirve de indicador para determinar los niveles de calidad en el desarrollo de los Sistemas.
- **Nivel 4:** Prácticas Bases. Las Prácticas Bases son un conjunto de directrices a ser ejecutadas por la organización para lograr alcanzar un principio; donde cada una de estas Prácticas Bases apoya a una o a las dos dimensiones de la Matriz de Calidad Sistémica.

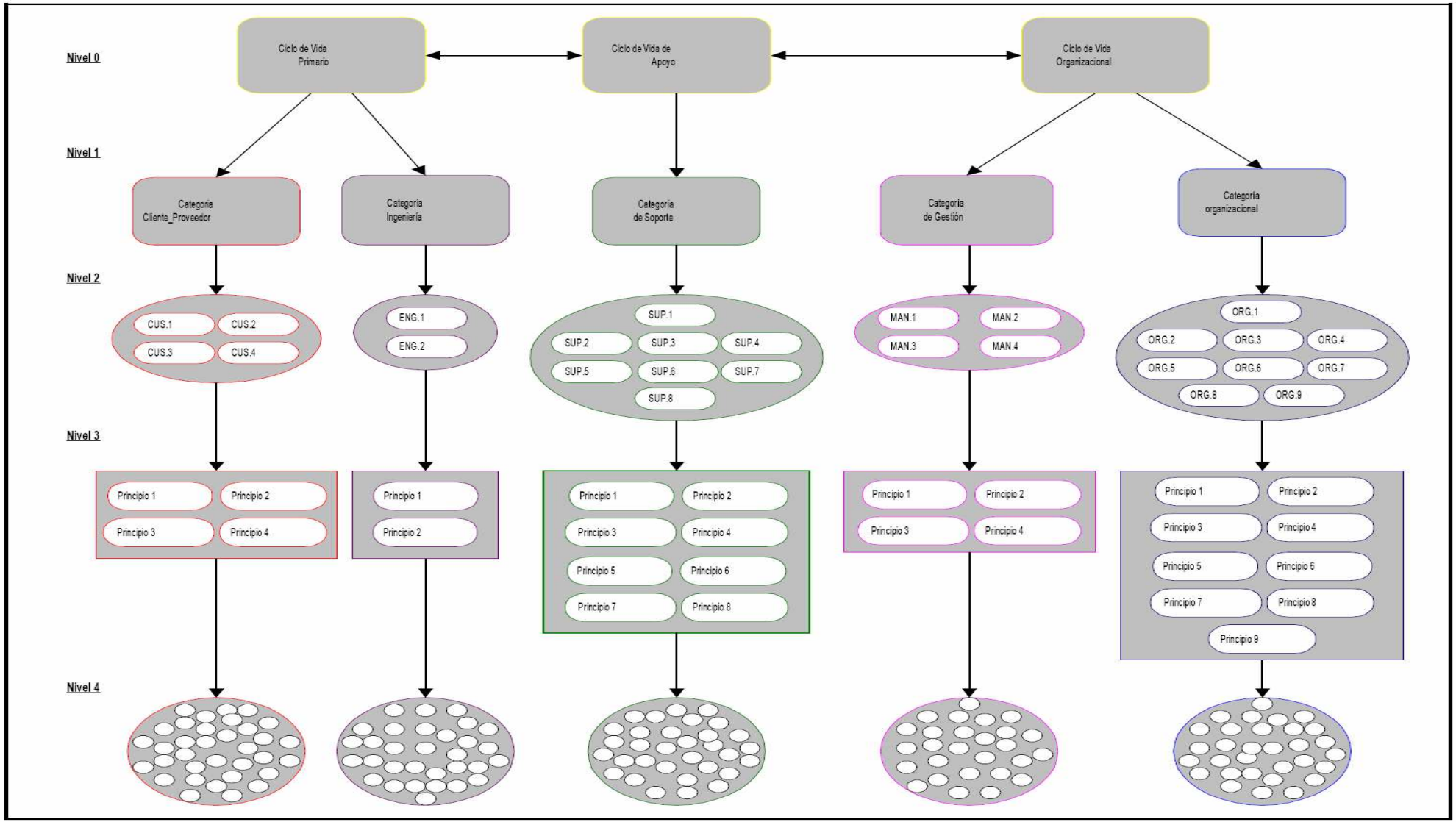

Figura 3 3 Estructura del Modelo de la Calidad del Proceso de Desarrollo de Software

El modelo de calidad del proceso, aunque soporta el enfoque sistémico propuesto por Callaos y Callaos, está orientado únicamente a la medición de la calidad del proceso de software, dejando de un lado la calidad del producto.

Hasta ahora se han descrito los modelos que permiten evaluar la calidad del producto de software y la calidad del proceso de desarrollo de manera aislada; es decir, mediante la implementación de modelos distintos que miden cada una de las calidades por separado aunque inspirados ambos en el modelo de Calidad Sistémica.

3.1.1.4 Modelo Sistémico De Calidad (MOSCA)

Partiendo de los dos modelos descritos anteriormente, se formula el prototipo de MOSCA (ver Figura 3.4), constituido por cuatro niveles, los cuales son explicados a continuación:

- **Nivel 0:** Dimensiones. Aspectos Internos del proceso, Aspectos Contextuales del proceso, Aspectos Internos del producto y Aspectos Contextuales del producto son las cuatro dimensiones propuestas en el modelo. Sólo un balance y una buena interrelación entre ellas garantizan la calidad Sistémica global de una organización.

- **Nivel 1:** Categorías. Se contemplan once categorías: seis pertenecientes al producto (ver Tabla 3.1) y las otras cinco al proceso de desarrollo (ver Tabla 3.2). Esta división no implica un desligamiento entre ellas, simplemente se realiza para identificar a que sector o sub-modelo pertenecen.

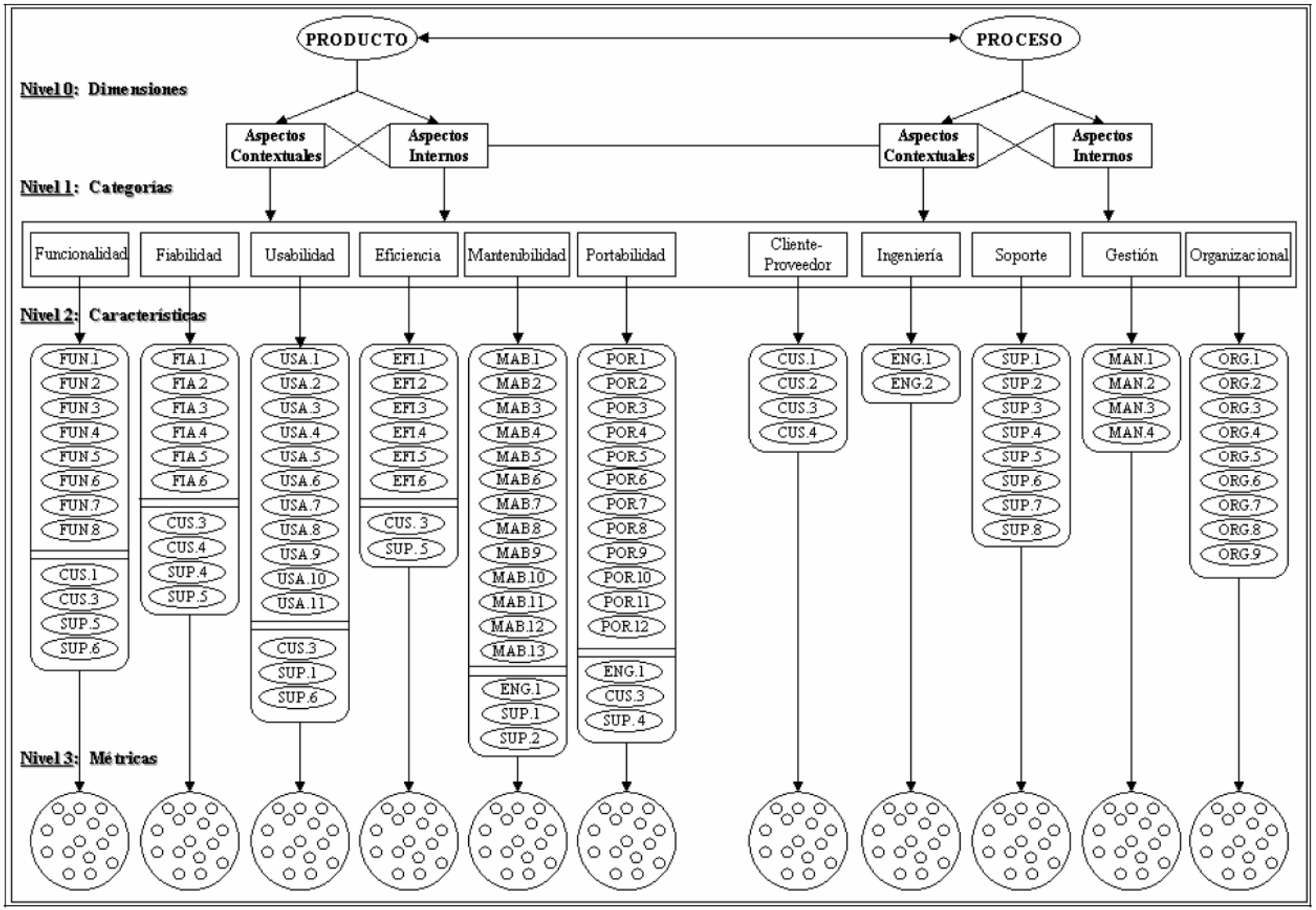

Figura 3 4 Diagrama del modelo MOSCA

Categoría del Producto	Definición
Funcionalidad (FUN)	**Capacidad del producto del software para proveer funciones que cumplan con necesidades específicas o implícitas, cuando el software es utilizado bajo ciertas condiciones.**
Fiabilidad (FIA)	**Capacidad del producto de software para mantener un nivel especificado de rendimiento cuando es utilizado bajo condiciones especificadas.**
Usabilidad (USA)	**Capacidad del producto de software para ser atractivo, entendido, aprendido y utilizado por el usuario bajo condiciones específicas.**
Eficiencia (EFI)	**Capacidad del producto de software para proveer un rendimiento apropiado, relativo a la cantidad de recursos utilizados, bajo condiciones específicas.**
Mantenibilidad (MAB)	**Capacidad del producto para ser modificado.**
Portabilidad (POR)	**Capacidad del producto de software para ser transferido de un ambiente a otro.**

Tabla 3. 1 Categorías del sub-modelo del producto

- **Nivel 2:** Características. Cada categoría tiene asociado un conjunto de características, las cuales definen las áreas claves a satisfacer para lograr, asegurar y controlar la calidad tanto en el producto como en el proceso. Entre las características asociadas a cada categoría del producto, se proponen una serie de características del proceso (ver Figura 3.4). Esto se debe, a que algunas características de la calidad del proceso, impactan directamente en las categorías del producto al igual que ciertas características de la calidad del producto definen categorías del proceso. Esto ayuda a precisar que si una vez medidas las características asociadas a una categoría en particular del producto, arroja resultados no deseados, se pueden analizar las características de la calidad del proceso asociadas a esa categoría del producto para encontrar las posibles causas.

Categoría del Proceso	Definición
Cliente – Proveedor (CUS)	**Está conformada por procesos que impactan directamente al cliente, apoya el desarrollo y la transición del Software hasta el cliente, y provee la correcta operación y uso del producto o servicio de software.**
Ingeniería (ENG)	**Consiste en procesos que directamente especifican, implementan o mantienen el producto de software, su relación con el Sistema y su documentación.**
Soporte	**Consta de procesos que pueden ser empleados por**

(SUP)	**cualquiera de los procesos (incluyendo a los de soporte) en varios niveles del ciclo de vida de adquisición.**
Gestión (MAN)	**Abarca los procesos que contienen prácticas genéricas, que pueden ser utilizadas por cualquier personal que dirija algún tipo de proyecto o proceso.**
Organizacional (ORG)	**Agrupa los procesos que establecen las metas comerciales de la organización y desarrollan bienes (valores) de proceso, producto y recurso, que ayudarán a la organización a alcanzar sus metas en los proyectos.**

Tabla 3. 2 Categorías del sub-modelo del proceso

En las Tablas 3.3 y 3.4 se muestran todas las características del modelo, agrupadas por cada una de las dimensiones de la Matriz Global Sistémica y acompañadas con la clave única que las identifica.

Categoría	Características: Aspectos Contextuales del Producto	Características: Aspectos Internos del Producto
Funcionalidad (FUN) Total de métricas: 46	**FUN 1. Ajuste a los propósitos (16)** **FUN 2. Precisión (10)** **FUN 3. Interoperabilidad (7)** **FUN 4. Seguridad (2)** **Sub-total de métricas: 35**	**FUN 5. Correctitud (8)** **FUN 6. Estructurado (1)** **FUN 7. Encapsulado (1)** **FUN 8. Especificado (1)** **Sub-total de métricas: 11**
Fiabilidad (FIA) Total de métricas: 32	**FIA 1. Madurez (17)** **FIA 2. Tolerancia a fallas (1)** **FIA 3. Recuperación (4)** **Sub-total de métricas: 22**	**FIA 4. Correctitud (8)** **FIA 5. Estructurado (1)** **FIA 6. Encapsulado (1)** **Sub-total de métricas: 10**
Usabilidad (USA) Total de métricas: 38	**USA 1. Facilidad de comprensión (5)** **USA 2. Capacidad de Aprendizaje (9)** **USA 3. Interfaz Gráfica (5)** **USA 4. Operabilidad (13)** **USA 5. Conformidad con los estándares** **Sub-total de métricas: 32**	**USA 6. Completo (1)** **USA 7. Consistente (1)** **USA 8. Efectivo (1)** **USA 9. Especificado (1)** **USA 10. Documentado (1)** **USA 11. Auto-descriptivo (1)** **Sub-total de métricas: 6**
Eficiencia (EFI) Total de métricas: 10	**EFI 1. Comportamiento del tiempo (2)** **EFI 2. Utilización de recursos (4)** **Sub-total de métricas: 6**	**EFI 3. Efectivo (1)** **EFI 4. No redundante** **EFI 5. Directo (1)** **EFI 6. Utilizado (1)** **Sub-total de métricas: 4**
Mantenibilidad (MAB) Total de métricas: 79	**MAB 1. Capacidad de análisis (2)** **MAB 2. Facilidad de Cambio (7)** **MAB 3. Estabilidad (4)** **MAB 4. Capacidad de prueba (3)**	**MAB 5. Acoplamiento (1)** **MAB 6. Cohesión (1)** **MAB 7. Encapsulado (1)** **MAB 8. Madurez del Software (17)** **MAB 9. Estructura de Control (4)** **MAB 10. Estructura de Información (9)** **MAB 11. Descriptivo (14)** **MAB 12. Correctitud (8)** **MAB 13. Estructural (5)** **MAB 14. Modularidad (3)**

Portabilidad (POR) Total de métricas: 44	**Sub-total de métricas: 16** **POR 1. Adaptabilidad (9)** **POR 2. Capacidad de Instalación (4)** **POR 3. Co-existencia (2)** **POR 4. Capacidad de reemplazo (2)**	**Sub-total de métricas: 63** **POR 5. Consistente (1)** **POR 6. Parametrizado (3)** **POR 7. Encapsulado (1)** **POR 8. Cohesivo (1)** **POR 9. Especificado (1)** **POR 10. Documentado (1)** **POR 11. Auto-descriptivo (1)** **POR 12. No redundante (1)** **POR 13. Auditoria (6)** **POR 14. Manejo de la Calidad (3)**
	Sub-total de métricas: 17	**Sub-total de métricas: 19**
	Calidad de los Datos (8) -abarca las dos dimensiones-.	

Tabla 3. 3 Distribución de las Características y métricas para medir la Calidad Sistémica del Producto de Software

Categoría	Características: Aspectos Contextuales del Proceso	Características: Aspectos Internos del Proceso
Cliente – Proveedor (CUS) Total de métricas: 57	**CUS 1. Adquisición del Sistema o producto de Software (24)** **CUS 3. Determinación de Requerimientos (20)**	**CUS 2. Suministro (8)** **CUS 4. Operación (5)**
	Sub-total de métricas: 44	**Sub-total de métricas: 13**
Ingeniería 13 (ING) Total de métricas: 29	**ENG 1. Desarrollo (12)**	**ENG 2. Mantenimiento de Software y Sistemas (17)**
	Sub-total de métricas: 12	**Sub-total de métricas: 17**
Soporte (SUP) Total de métricas: 130	**SUP 3. Aseguramiento de la Calidad (17)** **SUP 6. Revisión Conjunta (14)** **SUP 7. Auditoria (15)** **SUP 8. Resolución de Problemas (11)**	**SUP 1. Documentación (9)** **SUP 2. Gestión de Configuración (12)** **SUP 4. Verificación (6)** **SUP 5. Validación (6)** **SUP 6. Revisión Conjunta (14)** **SUP 7. Auditoria (15)** **SUP 8. Resolución de Problemas (11)**
	Sub-total de métricas: 57	**Sub-total de métricas: 73**
Gestión (MAN) Total de métricas: 91	**MAN 1. Gestión (14)** **MAN 3. Gestión de Calidad (10)** **MAN 4. Gestión del Riesgo (12)**	**MAN 1. Gestión (14)** **MAN 2. Gestión de Proyecto (19)** **MAN 3. Gestión de Calidad (10)** **MAN 4. Gestión del Riesgo (12)**
	Sub-total de métricas: 36	**Sub-total de métricas: 55**
Organizacional (ORG) Total de métricas: 123	**ORG 1. Lineam. Organizacionales (14)** **ORG 2. Gestión de Cambio (10)** **ORG 5. Mejoramiento del Proceso (16)** **ORG 8. Medición (11)** **ORG 9. Reuso (12)**	**ORG 3. Establecimiento del Proceso (11)** **ORG 4. Evaluación del Proceso (9)** **ORG 5. Mejoramiento del Proceso (16)** **ORG 6. Gestión de RRHH (16)** **ORG 7. Infraestructura (8)**
	Sub-total de métricas: 63	**Sub-total de métricas: 60**

Tabla 3. 4 Distribución de las Características y métricas para medir la Calidad Sistémica de Proceso de Desarrollo

- **Nivel 3:** Métricas. Para cada característica se propone una serie de métricas utilizadas para medir la calidad sistémica. Dada la cantidad de métricas

asociadas a cada una de las características que conforman MOSCA (ver Tablas 3.3 y 3.4), 679 en total, éstas no serán presentadas en el presente trabajo.

3.1.1.5 Algoritmo De Aplicación Del Modelo MOSCA

La Figura 3.5 explica gráficamente los pasos a seguir (algoritmo) para medir la calidad sistémica dentro de una organización a través de la aplicación del modelo MOSCA. Para ello, se deberá medir primero la calidad del producto de software y luego la calidad del proceso de desarrollo del mismo.

Según la Figura 3.5, siempre y en todos los casos se debe medir primero la categoría Funcionalidad del producto. Si cumple con todas las características necesarias que se proponen para esta categoría, entonces se deberá proceder a adaptar el sub-modelo del producto según las especificaciones del cliente. Si el producto no cumple con la categoría Funcionalidad, la evaluación finaliza; es decir, el resto del sub-modelo de producto y el sub-modelo del proceso no deberá ser evaluado. Esto se debe a que la categoría Funcionalidad es la más importante dentro de la medición de la calidad, ya que identifica la capacidad del mismo para cumplir las funciones para las que fue fabricado. Además, como aporte importante, se brinda al cliente las causas del por qué la Funcionalidad no pudo ser satisfecha y el nivel de calidad resultó ser nulo. Seguidamente se hace la instanciación del sub-modelo del producto. Para ello el cliente debe seleccionar dos (2) categorías de las cinco (5) restantes del modelo del producto; aquellas que considera que su producto de software debe cumplir y que desea que sean evaluadas. Una vez adaptado el sub-modelo del producto, se deberán evaluar cada una de las categorías seleccionadas por el cliente.

Finalmente, para poder medir la calidad del producto de software se presenta la Tabla 3.5, en la cual se relacionan el nivel de calidad con las categorías satisfechas. En este punto es preciso recordar que si no se satisface la categoría Funcionalidad el algoritmo finaliza y la calidad del producto de software será nula.

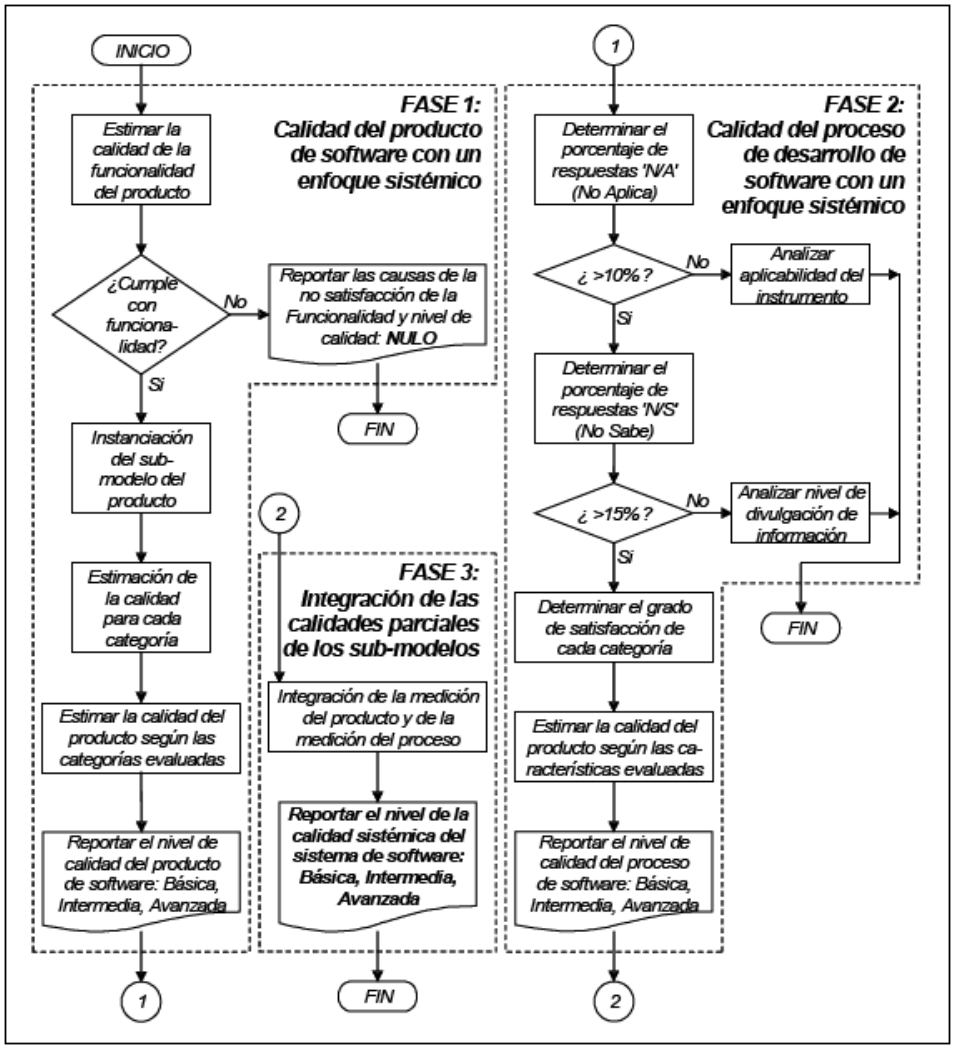

Figura 3 5 Algoritmo de Aplicación de MOSCA

Una vez terminada la evaluación del producto y sólo en caso de que este obtenga al menos un nivel de calidad básico, se procederá a medir la calidad del proceso a través del sub-modelo del mismo. Partiendo de las categorías evaluadas en el sub-modelo del Proceso, se estima la calidad de este según las categorías satisfechas:

Funcionalidad	Segunda Categoría Evaluada	Tercera Categoría Evaluada	Nivel de Calidad del Producto de Software
Satisfecha	**No Satisfecha**	**No Satisfecha**	**Básico**
Satisfecha	**Satisfecha**	**No Satisfecha**	**Intermedio**
Satisfecha	**No Satisfecha**	**Satisfecha**	**Intermedio**
Satisfecha	**Satisfecha**	**Satisfecha**	**Avanzado**

Tabla 3. 5 Nivel de calidad del producto con respecto a las categorías satisfechas para el producto

- **Calidad básica:** Es la mínima calidad requerida. Se satisfacen las características: Cliente-Proveedor e Ingeniería.
- **Calidad intermedia:** Esta no sólo satisface las características de Calidad básica, sino que, además, satisface las características de Soporte y Gestión.
- **Calidad avanzada:** Satisface todas las características.

Por último, se debe realizar una "integración" de la medición del producto y de la medición del proceso para obtener la medición de la calidad sistémica. Los niveles de calidad sistémica se proponen en la Tabla 3.6.

Nivel de Calidad Producto	Nivel de Calidad Proceso	Calidad Sistémica
Básico	**-**	**Nulo**
Básico	**Básico**	**Básico**
Intermedio	**-**	**Nulo**
Intermedio	**Básico**	**Básico**
Avanzado	**-**	**Nulo**
Avanzado	**Básico**	**Intermedio**
Básico	**Intermedio**	**Básico**
Intermedio	**Intermedio**	**Intermedio**
Avanzado	**Intermedio**	**Intermedio**
Básico	**Avanzado**	**Intermedio**
Intermedio	**Avanzado**	**Intermedio**
Avanzado	**Avanzado**	**Avanzado**

Tabla 3. 6 Nivel de Calidad Sistémica Global a partir del nivel de Calidad del Producto

Como puede observarse en la Tabla 3.6, esta propuesta obedece a la necesidad de mantener un equilibrio entre las distintas dimensiones de la calidad de los sistemas de software, es por ello que, la Calidad del Producto de Software tiene igual peso que la Calidad del Proceso de Desarrollo de Software. Se considera que la aplicación del modelo permitirá ajustar con mayor precisión este "equilibrio". Finalmente, se tiene que la integración de las medidas de calidad de los sub-modelos mide la calidad sistémica como una balanza; es decir, si el nivel de calidad de uno de los sub-modelos es menor que el nivel del otro sub-modelo, entonces la balanza no estará estable y por ello se inclinará hacia el nivel de menor calidad. Esto se debe a la sencilla razón de que si la Calidad del Producto de Software o la Calidad del Proceso de Desarrollo no cumplen con las características necesarias para tener un nivel más alto de calidad, implicará

directamente que la calidad sistémica tampoco cumpla con las características necesarias para tener un nivel de calidad superior.

3.2 EVALUACIÓN

Según Gros se considera software educativo a cualquier producto basado en computadora con una finalidad educativa[29]. Así mismo, según Galvis, en el campo educativo suele denominar software educativo a aquellos programas que permiten cumplir y apoyar funciones educativas. En esta categoría entran tanto los que dan soporte al proceso de enseñanza y aprendizaje (un sistema para enseñar matemáticas, ortografía, contenidos o ciertas habilidades cognitivas), como los que apoyan la administración de procesos educacionales o de investigación (Ej. un sistema que permita manejar un banco de preguntas)[30]. El caso de estudio realizado está relacionado principalmente con la primera definición; es decir, con los materiales educativos computarizados que apoyan el proceso de enseñanza-aprendizaje.
Cuando se habla de calidad de software educativo, se requiere un producto que satisfaga tanto las expectativas de los docentes como de los usuarios, a un menor costo, libre de defectos y cumpliendo con ciertas especificaciones.

Según Gros, la calidad del software educativo está determinada no sólo por los aspectos técnicos del producto sino por el diseño pedagógico y los materiales de soporte. Este último aspecto es uno de los más problemáticos ya que existen pocos programas que ofrecen un soporte didáctico[31]. La evaluación de software educativo se ha centrado tradicionalmente en dos momentos:

1. Durante su utilización real por los usuarios, para juzgar su eficiencia y los resultados que con él se obtienen.
2. Durante el proceso de diseño y desarrollo, con el fin de corregir y perfeccionar el programa.

[29] GROS, B., BERNARDO, A., LIZANO, M., MARTINEZ, C., PANADÉS, M., RUIZ, I., Diseños y programas educativos, pautas pedagógicas para la elaboración de software. Editorial Ariel S.A., 1997, 149 pp.
[30] GALVIS, A., Ingeniería de software educativo 2da. reimpresión. Uniandes Colombia, 2000.
[31] GROS, B., BERNARDO, A., LIZANO, M., MARTINEZ, C., PANADÉS, M., RUIZ, I., Diseños y programas educativos, pautas pedagógicas para la elaboración de software. Editorial Ariel S.A., 1997, 149 pp.

En consecuencia, el tema de evaluación del software educativo ha sido estudiado y documentado por diversos autores del ámbito educativo, proporcionando medidas de evaluación en el área educativa y técnica. Destacan los métodos de evaluación de Galvis y de la Universidad Virtual de Michigan, que utilizan métodos cuantitativos de evaluación. Sin embargo, en vista de que gran parte de las propuestas sobre software educativo, son de índole cualitativa o necesitan adaptarse a medidas estándares de evaluación de software según las normas ISO/IEC 9126, surge la necesidad de la disponibilidad de un instrumento de medidas estándares de calidad para la evaluación de software educativo, que sea de utilidad tanto para los desarrolladores de software educativo como para los interesados en adquirir software comercial (por ejemplo, educadores e instituciones educativas).

3.2.1 ADECUACIÓN DE MOSCA PARA SOFTWARE EDUCATIVO

Dado que MOSCA es un modelo de especificación de la calidad de los sistemas de software, que además permite su medición, en este capítulo se hace una adecuación del mismo para software educativo.

Como un primer alcance, se decidió utilizar tanto la perspectiva producto como la perspectiva proceso. De la primera perspectiva, sólo la dimensión de la Efectividad del producto se utilizó, en virtud de que la aplicación desarrollada busca cubrir las necesidades de maestros de educación básica para la enseñanza de idiomas. De la perspectiva del proceso, la dimensión de la Efectividad del Proceso se utilizó, ya que es necesario conocer la calidad del proceso de desarrollo de software basado en componentes COTS que se propuso en el Capítulo 1.

La propuesta del modelo de evaluación de calidad del caso de estudio consiste, entonces, en un conjunto de categorías, características, subcaracterísticas y las métricas asociadas (ver Figura 3.6, Tabla 3.7 y Tabla 3.8). La estructura del modelo consta de cuatro niveles que se explican brevemente a continuación:

- **Perspectiva: Producto**

  Nivel 0: Dimensiones. Efectividad Producto.

  Nivel 1: Categorías. Se contemplan tres categorías:

  - **Funcionalidad (FUN):** Es la capacidad del producto del software para proveer funciones que cumplan con necesidades específicas o implícitas, cuando el software es utilizado bajo ciertas condiciones.
  - **Usabilidad (USA):** Esta categoría se refiere a la capacidad del producto de software para ser atractivo, entendido, aprendido y utilizado por el usuario bajo condiciones específicas.
  - **Fiabilidad (FIA):** La fiabilidad es la capacidad del producto de software para mantener un nivel especificado de rendimiento cuando es utilizado bajo condiciones especificadas.

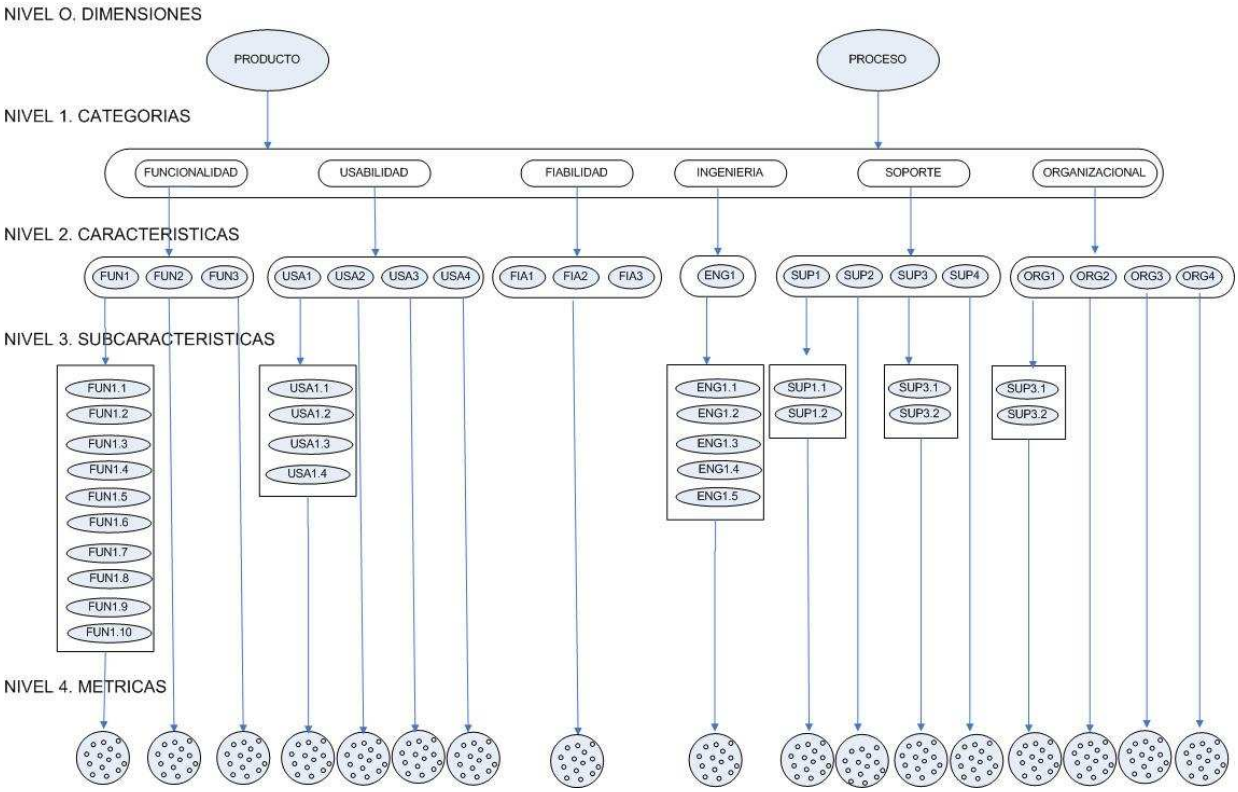

Figura 3 6 Propuesta del modelo de evaluación de software educativo

Como se indicó, MOSCA consta de seis categorías, de las cuales sólo se deben utilizar tres para la evaluación de software educativo. Debido a que la categoría de Funcionalidad siempre debe estar presente, en esta actividad se seleccionan dos categorías de las cinco restantes del modelo del producto (Usabilidad, Fiabilidad, Eficiencia, Mantenibilidad y Portabilidad). Se seleccionaron Usabilidad y Fiabilidad. La Usabilidad es seleccionada debido a que para que un software educativo motive al aprendizaje, es fundamental que el material educativo sea atractivo y de fácil manejo, debe generar actividades interactivas que motiven y mantengan la atención, actividades que deben ser variadas y que respondan a los diversos estilos de aprendizaje. Se seleccionó Fiabilidad debido a que es importante que el producto funcione bajo las condiciones establecidas y mantenga un nivel específico de rendimiento para garantizar un ambiente de aprendizaje adecuado.

Nivel 2: Características. Cada categoría tiene asociado un conjunto de características (10 en total). Una vez seleccionadas las categorías que están relacionadas con la evaluación de software educativo (Funcionalidad, Usabilidad y

Fiabilidad), se seleccionan las características asociadas a estas categorías en MOSCA, que están relacionadas con el área educativa. Se decidió seleccionar ciertas características asociadas a la efectividad del producto y no a la eficiencia del producto, debido a que, al adquirir un software comercial, no se tiene acceso a los documentos que permiten aplicar las métricas correspondientes a esta dimensión, por lo que no es posible evaluarla.

Nivel 3: Sub-características. Para algunas de las características se asocian un conjunto de sub-características. Para algunas características, tales como 'Ajuste a los propósitos' y 'Facilidad de comprensión del software', se agregó un conjunto de sub-características (14 en total) que añadieron el componente educativo a MOSCA.

Nivel 4: Métricas. Para cada característica se propone una serie de métricas utilizadas para medir la calidad sistémica. Es necesaria una selección de métricas adicionales relacionadas con Funcionalidad, Usabilidad y Fiabilidad, que permitan adaptar MOSCA en el área de software educativo.

CATEGORÍA	CARACTERÍSTICAS	SUBCARACTERÍSTICAS
FUNCIONALIDAD (FUN)	FUN.1 Ajuste a los propósitos	FUN.1.1 General FUN.1.2 Objetivos de aprendizaje FUN.1.3 Contenidos de aprendizaje FUN.1.4 Actividades de aprendizaje FUN.1.5 Ejemplos FUN.1.6 Motivación FUN.1.7 Retroalimentación FUN 1.8 Ayudas FUN.1.9 Evaluación y registro de datos FUN.1.10 Metodología de enseñanza
	FUN.2 Precisión FUN.3 Seguridad	
USABILIDAD (USA)	USA.1 Facilidad de comprensión	USA.1.1 General USA.1.2 Interactividad USA.1.3 Diseño de la interfaz USA.1.4 Guías didácticas

	USA.2 Capacidad de uso USA.3 Interfaz Gráfica USA.4 Operabilidad
FIABILIDAD (FIA)	FIA.1 Madurez FIA.2 Recuperación FIA.3 Tolerancia a fallas

Tabla 3. 7 Propuesta de categorías, características, sub-características y número de métricas del Producto para el modelo propuesto basado en MOSCA

- **Perspectiva: Proceso**

  Nivel 0: Dimensiones. Efectividad Proceso.

  Nivel 1: Categorías. Se contemplan tres categorías:

  - **Ingeniería (ING):** Consiste en procesos que directamente especifican, implementan o mantienen el producto de software, su relación con el Sistema y su documentación.
  - **Soporte (SOP):** Consta de procesos que pueden ser empleados por cualquiera de los procesos (incluyendo a los de soporte) en varios niveles del ciclo de vida de adquisición.
  - **Organizacional (ORG):** Agrupa los procesos que establecen las metas comerciales de la organización y desarrollan bienes (valores) de proceso, producto y recurso, que ayudarán a la organización a alcanzar sus metas en los proyectos. Esta categoría es muy importante, pues la meta que se desea alcanzar es un modelo de calidad para el desarrollo de software basado en componentes COTS.

  Nivel 2: Características. Cada categoría tiene asociado un conjunto de características (9 en total). Una vez seleccionadas las categorías que están relacionadas con la evaluación de software educativo (Ingeniería, Soporte y Organizacional), se seleccionan las características asociadas a estas categorías en MOSCA, que están relacionadas con el desarrollo de software basado en componentes.

  Nivel 3: Sub-características. Para algunas de las características se asocian un conjunto de sub-características. Para algunas características, tales como 'Desarrollo', se agregó un conjunto de sub-características (11 en total) a MOSCA.

Nivel 4: Métricas. Para cada característica se propone una serie de métricas utilizadas para medir la calidad sistémica. Es necesaria una selección de métricas relacionadas con el proceso de desarrollo de software basado en componentes.

CATEGORÍA	CARACTERÍSTICAS	SUBCARACTERÍSTICAS
INGENIERIA (ENG)	ENG.1 Desarrollo	ENG.1.1 Modularidad ENG.1.2 Simplicidad ENG.1.3 Cumplimiento de estándares ENG.1.4 Reusabilidad ENG.1.5 Mantenimiento
SOPORTE (SUP)	SUP.1 Aseguramiento de la Calidad	SUP.1.1 Correctitud SUP.1.2 Confiabilidad
	SUP.2 Documentación	SUP.2.1 Consistencia SUP.2.2 Cumplimiento de estándares
	SUP.3 Gestión de Configuración SUP.4 Auditoría	
ORGANIZACIONAL (ORG)	ORG.1 Establecimiento del Proceso	ORG.1.1 Efectivo ORG.1.2 Consistente
	ORG.2 Evaluación del Proceso ORG.3 Mejoramiento del Proceso ORG.4 Reuso	

Tabla 3. 8 Propuesta de categorías, características, sub-características y número de métricas del Proceso para el modelo propuesto basado en MOSCA

En resumen, la propuesta del modelo de evaluación de software educativo consta de un total de 2 dimensiones, 6 categorías, 19 características, 25 sub-características y sus respectivas métricas.

Una vez formulado el modelo, se diseñó el proceso a seguir para estimar la calidad del Software educativo según MOSCA.

3.2.2 RESULTADOS DE LA EVALUACIÓN

3.2.2.1 Evaluación del Producto

Para realizar la evaluación del producto, se aplicó un cuestionario al personal del Grupo Editorial Santillana, específicamente a 8 personas pertenecientes a los departamentos de: CAPSE (Capacitación a los Docentes Santillana Ecuador), Richmond Publishing (Área dedicada a la enseñanza del idioma Inglés), Editorial (Área dedicada a la realización de textos escolares y multimedia), Gerencia de Proyectos (Área encargada de la creación de nuevos proyectos educativos). El cuestionario se desarrolló a partir de las métricas que conforman el modelo propuesto (se puede encontrar el formato del cuestionario en la sección Anexos). A continuación se muestran los resultados obtenidos en la evaluación:

3.2.2.1.1 Funcionalidad

En la Figura 3.7 se muestran los porcentajes alcanzados por AMASOP, en cuanto a los requerimientos de calidad asociados a Funcionalidad.

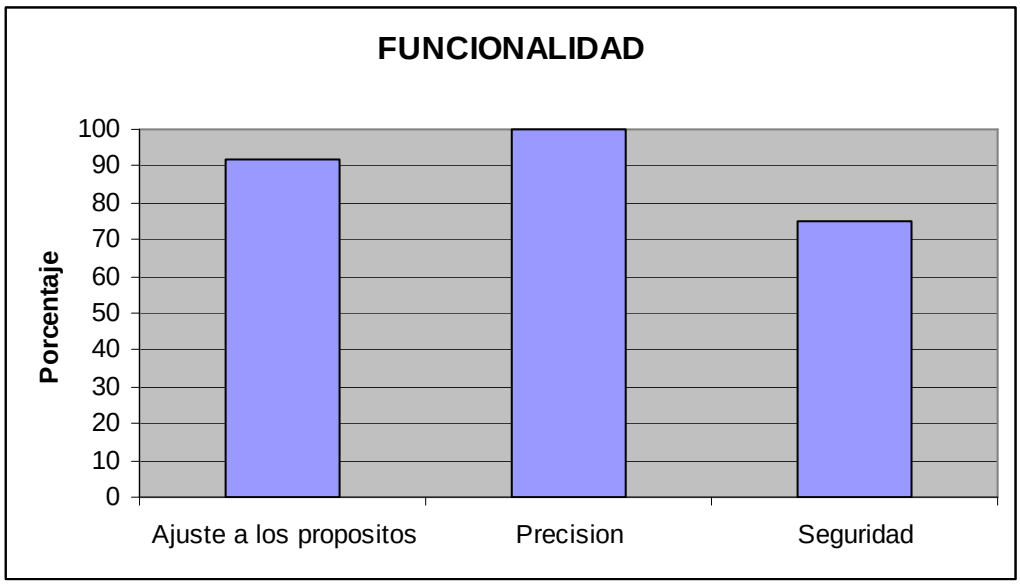

Figura 3 7 Porcentaje de satisfacción de AMASOP frente a las tres características de Funcionalidad

Como se puede observar en la Figura 3.7, AMASOP cumple con las características de 'Ajuste a los propósitos', 'Precisión' y 'Seguridad', por lo tanto se concluye que la

categoría Funcionalidad es satisfecha para el software evaluado; y se procede a continuación con el algoritmo del modelo ampliado de MOSCA.

3.2.2.1.2 Usabilidad

La Figura 3.8 muestra los porcentajes alcanzados por las características asociadas a la categoría Usabilidad.

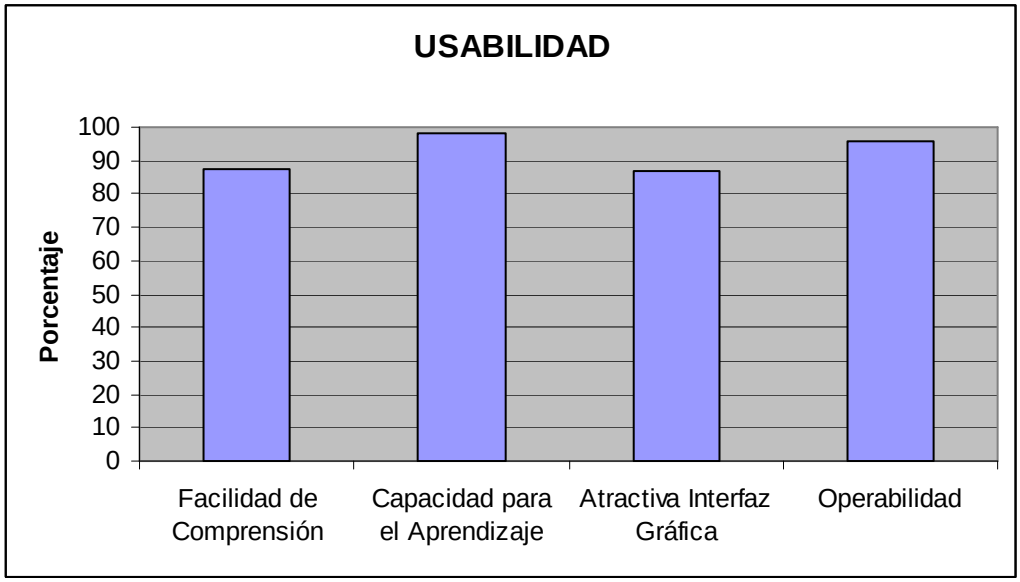

Figura 3 8 Porcentaje de satisfacción de AMASOP frente a las características de Usabilidad

Como se puede observar en la Figura 3.8, cumple con las cuatro características asociadas a Usabilidad, puesto que todas sobrepasan el 75% de la satisfacción. Por lo tanto se concluye que la categoría Usabilidad es satisfecha para AMASOP.

3.2.2.1.3 Fiabilidad

La Figura 3.9 muestra los porcentajes alcanzados por las características asociadas a la categoría Fiabilidad.

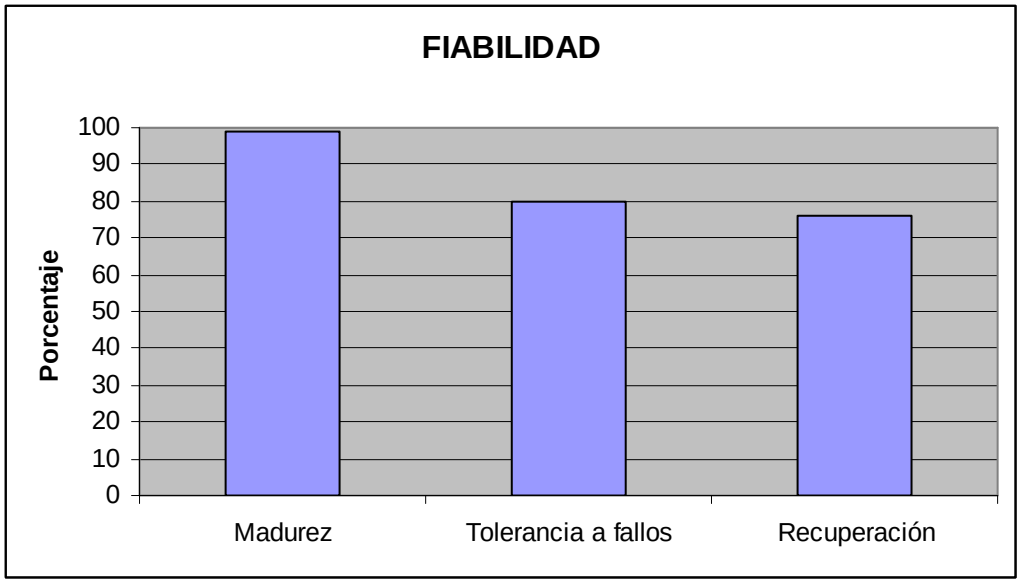

Figura 3 9 Porcentaje de satisfacción de AMASOP frente a las características de Fiabilidad

Como se puede observar en la Figura 3.9, AMASOP cumple con las características de Fiabilidad, aunque en la característica de Recuperación, AMASOP se encuentra en el límite permitido, se puede concluir que la categoría Fiabilidad es satisfecha.

Analizando los resultados de las Figuras 3.7, 3.8 y 3.9, se tiene la Tabla 3.9, la cual muestra que AMASOP presenta un nivel de calidad avanzado.

Software educativo	Categoría			Nivel de calidad
	Funcionalidad	Usabilidad	Fiabilidad	
AMASOP	Satisfecha	Satisfecha	Satisfecha	Avanzada

Tabla 3. 9 Resultados de la evaluació de AMASOP según el modelo propuesto

Como se podrá observar en la Tabla 3.9, para que un software educativo presente nivel de calidad avanzada, debe satisfacer las tres categorías, a saber, Funcionalidad, Usabilidad y Fiabilidad. El nivel de calidad Intermedia sólo será posible en el caso de que las categorías de Funcionalidad y Usabilidad sean satisfechas. Esto quiere decir que si el software educativo tiene las categorías de Funcionalidad y Fiabilidad satisfechas, el nivel de calidad será sólo de Básico. La exigencia de poseer al menos Funcionalidad y Usabilidad para poseer el nivel de calidad Intermedio, se debe a que el software educativo tiene que cumplir con los propósitos para el cual fue diseñado, debe ser fácil

de usar y poseer una interfaz adecuada a los propósitos y a la población de estudiantes para la que fue diseñada.

3.2.2.2 Evaluación del Proceso

En la evaluación del proceso de desarrollo de la aplicación, se obtuvieron los siguientes resultados, posterior a la aplicación de las métricas correspondientes a las características de las diferentes categorías.

3.2.2.2.1 Ingeniería

La Figura 3.10 muestra los porcentajes alcanzados por las características asociadas a la categoría Ingeniería.

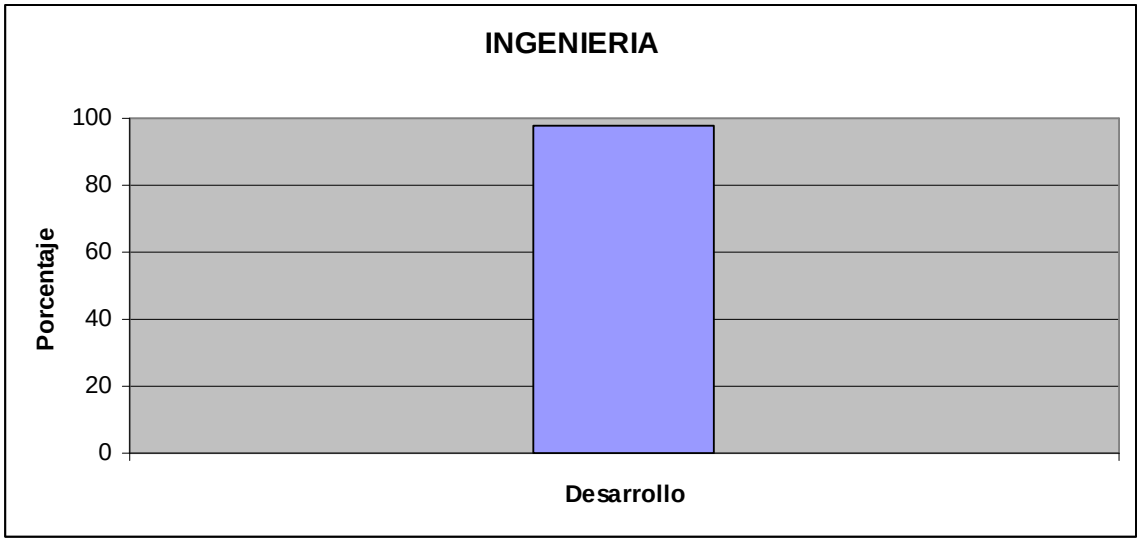

Figura 3 10 Porcentaje de satisfacción de AMASOP frente a las características de Ingeniería

Como se puede observar en la Figura 3.10, AMASOP cumple con la característica de Desarrollo, pues cumple con un porcentaje mayor al 90% de satisfacción, por lo que se puede concluir que, los procesos que directamente especifican, implementan o mantienen el producto de software, su relación con el sistema y su documentación, es decir, la categoría Ingeniería, es satisfecha.

3.2.2.2.2 Soporte

En la Figura 3.11 se muestran los porcentajes de satisfacción alcanzados por las características asociadas a la categoría Soporte.

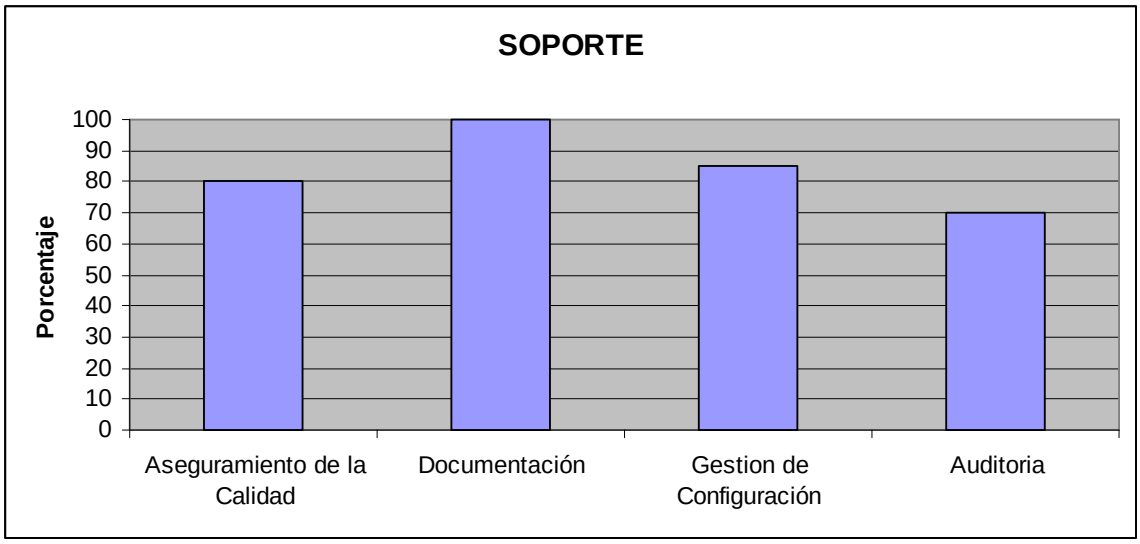

Figura 3 11 Porcentaje de satisfacción de AMASOP frente a las características de Soporte

Como se puede observar, la característica de Documentación alcanza un porcentaje de satisfacción del 100%, esto debido a que, para el desarrollo de la aplicación, se siguió exactamente el proceso propuesto para Desarrollo de Software Basado en Componentes COTS. La característica de Documentación, es seguida por la característica de Gestión de Configuración, la cual sobrepasa el 80% de satisfacción, seguida por la característica de Aseguramiento de la Calidad, la cual tiene un porcentaje de 80% de satisfacción, esto debido a que la calidad no puede ser completamente asegurada en ciertas fases del proceso de desarrollo, por el hecho de usar componentes de terceros, y de los cuales no existe documentación asociada con su calidad. Finalmente tenemos a la categoría de Auditoria, la cual alcanza un 70% de satisfacción, del mismo modo que la anterior categoría, esto es debido a que, por la existencia de componentes de Terceros, que son de naturaleza de caja negra, no se puede tener información relevante para procesos de auditoria. Sin embargo, aunque la característica de auditoria es la única que no cumple con al menos el 75% de satisfacción, podemos concluir que la categoría Soporte es satisfecha por la aplicación.

3.2.2.2.3 Organizacional

En la Figura 3.12 se muestran los porcentajes de satisfacción alcanzados por las características asociadas a la categoría Organizacional.

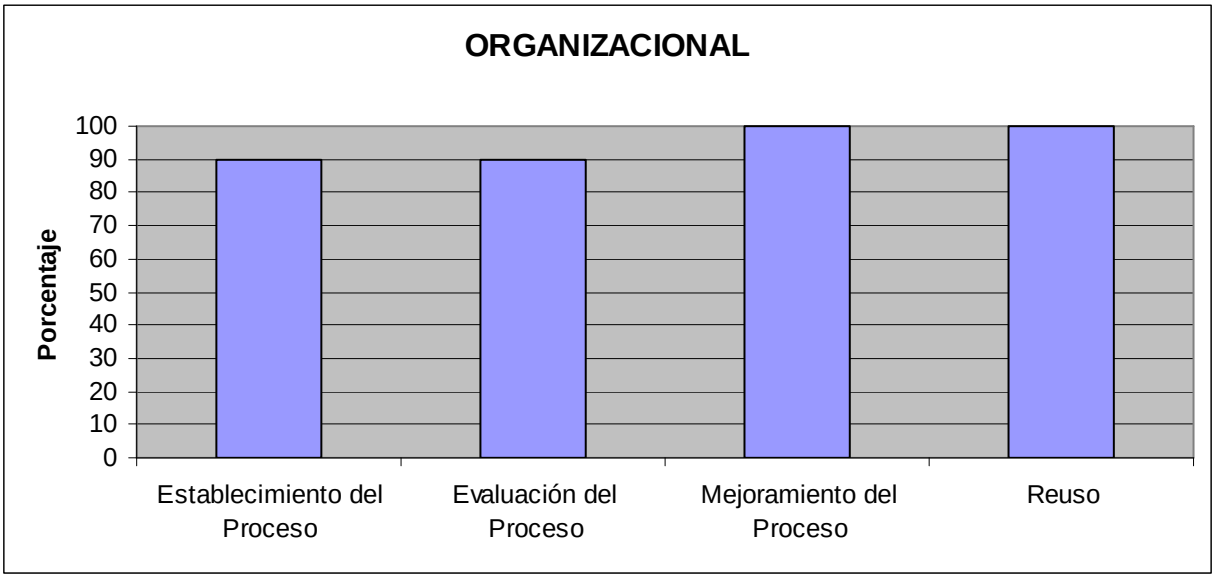

Figura 3 12 Porcentaje de satisfacción de AMASOP frente a las características de la categoría Organizacional

Con respecto a la categoría Organizacional, y como se puede observar en la Figura 3.12, las características de Mejoramiento del Proceso y de Reuso tienen un 100% de satisfacción, esto atendiendo a que, el uso de componentes comerciales, mejora en varios aspecto el proceso de desarrollo, como tiempos de entrega, facilidad de programación, disminución de líneas de código, etc. Las características de Establecimiento del Proceso y de Evaluación del Proceso, alcanzan un 90% de satisfacción, esto debido a que, se vio que en ciertas etapas del proceso propuesto, se hacen necesarias actividades extras, no contempladas en el proceso de desarrollo, y que deben ser cumplidas tanto por los proveedores como los adquirientes de componentes COTS, un ejemplo de estas actividades, es el hecho de tener una documentación precisa de los componentes que se ofrecen y se requieren.

Con los porcentajes obtenidos por las características mencionadas, se puede concluir que la categoría Organizacional es satisfecha.

Analizando los resultados de las Figuras 3.10, 3.11 y 3.12, se tiene la Tabla 3.10, la cual muestra que AMASOP presentan nivel de calidad avanzado en el desarrollo del proceso.

Proceso de Desarrollo de AMASOP	**Categoría**			**Nivel de calidad**
	Ingeniería	**Soporte**	**Organizacional**	
Desarrollo de Software basado en Componentes Comerciales COTS	Satisfecha	Satisfecha	Satisfecha	Avanzada

Tabla 3. 10 Resultados de la evaluación del proceso de desarrollo de AMASOP según el modelo propuesto

Como se puede observar en la Tabla 3.10, para que el proceso de Desarrollo de Software, presente nivel de calidad avanzada, debe satisfacer las tres categorías, a saber, Ingeniería, Soporte y Organizacional.

Por último, se realiza una "integración" de la medición del producto y de la medición del proceso para obtener la medición de la calidad sistémica. Como podemos ver en la Tabla 3.11, el nivel de calidad sistémica alcanzado por AMASOP es Avanzada.

Nivel de Calidad Producto	Nivel de Calidad Proceso	Calidad Sistémica
Avanzado	**Avanzado**	**Avanzado**

Tabla 3. 11 Nivel de Calidad Sistémica alcanzado por AMASOP

Se concluye entonces, que AMASOP es una aplicación de calidad tanto para el producto, como para el proceso de desarrollo.

CAPITULO 4 CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

Del trabajo realizado, y tomando en cuenta los objetivos planteados, se puede concluir que el Desarrollo de Software Basado en Componentes, es utilizado para reducir los costos, tiempos y esfuerzos en el desarrollo del software, a la vez que ayuda a mejorar la fiabilidad, flexibilidad y la reutilización de la aplicación final.

Tomando en cuenta que los componentes son partes de software que se pueden combinar con otros componentes para generar un conjunto aún mayor (por ejemplo otro componente, subsistema o sistema), se tiene que un componente juega el papel de una unidad software reutilizable que puede interoperar con otros módulos de software mediante sus interfaces. Un componente define una o más interfaces desde donde se puede tener acceso a los servicios que éste ofrece a los demás componentes.

Cabe recalcar que un componente COTS es una unidad de elemento de software en formato binario, utilizada para la composición de sistemas de software basados en componentes, que generalmente es de granularidad gruesa, que puede ser adquirido mediante compra, licencia, o ser un software de dominio público, y con una especificación bien definida que reside en repositorios conocidos.

Entre las ventajas de utilizar componentes COTS tenemos al factor económico, relacionado con el costo de desarrollo, y al hecho de que un componente COTS haya sido probado y validado por el vendedor y por otros usuarios del componente en el mercado, lo que suele hacer que sea aceptado como un producto mejor diseñado y más fiable que los componentes construidos por uno mismo. Otra ventaja es que el uso de un producto comercial permite integrar nuevas tecnologías y nuevos estándares más fácil y rápidamente que si se construye por la propia organización.

En estos últimos años se ha comprobado un aumento en el uso de componentes comerciales en prácticas de reutilización de software. Concretamente, estos componentes comerciales, están siendo considerados con mayor frecuencia para la

construcción de sistemas complejos, distribuidos y abiertos. Para la elaboración de estos sistemas, los ingenieros utilizan metodologías, procesos y técnicas de desarrollo basados en componentes, pero se hace necesario un método propio y dirigido estrictamente al Desarrollo de Software Basado en Componentes COTS, esto debido a que los sistemas de componentes COTS se construyen mediante la integración a gran escala de componentes adquiridos a terceras partes. La naturaleza de la caja negra de estos componentes, la posibilidad de que exista una incompatibilidad con la arquitectura, y su corto ciclo de vida, requiere una aproximación de desarrollo diferente.

Finalmente, es importante reiterar que para realizar un proceso de evaluación de calidad, se debe tomar en cuenta que la calidad del software no es algo que depende de una sola característica en particular, sino que obedece al compromiso de todas sus partes.

4.2 RECOMENDACIONES

Se recomienda distinguir que existen dos estilos de desarrollo de software relacionados con componentes, por un lado está el estilo de desarrollo de software basado en reutilización, donde las aplicaciones se construyen a partir de otras partes de software ya existentes y accesibles en repositorios conocidos; y por otro lado está el desarrollo de software de reutilización, donde se ponen en práctica procesos de ingeniería para la elaboración de partes eficientes de software que luego pueden ser utilizadas para la construcción de aplicaciones.

Se recomienda también definir un mecanismo para documentación de componentes comerciales, que sea útil para tareas de búsqueda, selección y ensamblaje de componentes. Para estas labores, es necesario que la documentación de los componentes contenga información del tipo funcional como signaturas, protocolos y comportamiento, y también información no funcional y no técnica del componente.

Se recomienda como trabajo futuro, la realización de herramientas que automaticen las operaciones de emparejamiento entre componentes a nivel de protocolos y a nivel semántico. Es muy importante contar con un soporte automatizado para las comprobaciones de compatibilidad y reemplazabilidad de servicios, ya que estas son dos aspectos claves en el DSBC.

Finalmente, se recomienda siempre englobar las dos tendencias actuales de modelos de calidad; estas son calidad del producto (software) y calidad del proceso, tomando en cuenta que la calidad del proceso garantiza la calidad del producto y consecuentemente no se pueden desligar.

BIBLIOGRAFÍA

TEXTOS

BOOCH, Grady; JACOBSON, Ivar; RUMBAUGH, James. El Lenguaje Unificado de Modelado, Addison-Wesley Iberoamericana España, S.A. Primera Edición, 2000.

BOOCH, Grady; JACOBSON, Ivar; RUMBAUGH, James. El Proceso Unificado De Desarrollo De Software, Addison-Wesley Iberoamericana España, S.A. Primera Edición, 2000.

BASS, L., CLEMENTS, P., y KAZMAN, R. Software Architecture in Practice. Addison-Wesley, 1998.

BROWN, A. Constructing Superior Software, Macmillan Technical Publishing, 1999.

BROY, M., DEIMEL, A., HENN, J., KOSKIMIES, K., PLASIL, F., POMBERGER, G., PREE, W., Stal, M., y Szyperski, C. What characterizes a (software) component? Software - Concepts and Tools, 1998.

CHEESMAN, J. y DANIELS, J. UML Components. A Simple Process for Specifying Component-Based Software. Addison-Wesley, 2001.

DE CESARE Sergio, LYCETT Mark, MACREDIE Robert D., Development of Component-Based Information Systems, S/E, 2005.

GALVIS, A., Ingeniería de software educativo 2da. reimpresión. Uniandes Colombia, 2000.

GROS, B., BERNARDO, A., LIZANO, M., MARTINEZ, C., PANADÉS, M., RUIZ, I., Diseños y programas educativos, pautas pedagógicas para la elaboración de software. Editorial Ariel S.A., 1997.

HEINEMAN, G. T. y COUNCILL, W. T. Component-Based Software Engineering. Putting the Pieces Together. Addison-Wesley, 2001.

JACOBSON, I., CHRISTERSON, M., JONSSON, P., y OVERGAARD, G. Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley Longman, S/E, 1992.

MEYERS, B. C. y OBERNDORF, P. Managing Software Acquisition: Open Systems and COTS Products. The SEI Series in Software Engineering, Addison-Wesley, 2001.

PRESSMAN, Roger S. Ingeniería del Software, McGraw-Hill e Interamericana de mexico, Sexta edición, 2005.

ROBERTSON, S. y ROBERTSON, J. Mastering the Requirements Process. Addison-Wesley, 1999.

SZYPERSKI, C. Component Software. Beyond Object-Oriented Programming., Addison-Wesley, 1998.

WALLNAU, K. C., HISSAM, S. A., y SEACORD, R. C. Building Systems from Commercial Components. The SEI Series in Software Engineering. Addison-Wesley, 2002.

DOCUMENTOS

BROOKS, F. No Silver Bullet: Essence and Accidents of Software Engineering. IEEE Computer, 1987.

CALLAOS, N. y B. CALLAOS, Designing with Systemic Total Quality, Conferencia Internacional de Sistemas de Información, Orlando, Florida, July 1996.

IEEE, IEEE Standard Glossary of Software Engineering Terminology. IEEE Std610.12, IEEE Computer Society Press, 1990.

IRIBARNE, L., TROYA, J. M., y VALLECILLO, A., Study of the RM-ODP Trading Function. Informe Técnico número TR-CBSE-01, Departamento de Lenguajes y Computación. Universidad de Almería, 2001.

JORDON, K. y DAVIS, A. Requirements Engineering Metamodel: An Integrated View of Requirements. En “Fifteenth Annual International Computer Software and Applications Conference”. IEEE Computer Society Press, 1991.

MAIER, M. W., EMERY, D., y HILLIARD, R. Software Architecture: Introducing IEEE Standard 1471. IEEE Computer, 2001.

SITIOS WEB

http://www.ibm.com/software/components
http://www.cotstrader.com/
http://acacia.ual.es/profesor/LIRIBARNE/research/Jis00ref05/
http://www.ual.es/~liribarn/Investigacion/JISBD2002.html
http://www.trusted-components.org
http://www.componentsource.com