
# Universidad del Azuay

## Facultad de Ciencias de la Administración

## Escuela de Ingeniería de Sistemas

"INVESTIGACION DEL PROCESO DE INGENIERÍA DE SOFTWARE BASADO EN COMPONENTES (ISBC) Y SU APLICACIÓN A UN CASO PRÁCTICO"

**Monografía de graduación previo a la obtención del título de**

**Ingeniera de Sistemas**

**Autor:** Mariela Susana Farfán Torres

Ruth Noemí Rojas

**Director: Ing. Pablo Pintado**

**Cuenca, Ecuador**

**2008**

# DEDICATORIA

## DEDICATORIA MARIELA

Quiero dedicar este trabajo en primer lugar a mis padres ya que sin su apoyo, su cariño y comprensión no habría podido salir adelante, a mis hermanos y a cada uno de mis amigos que me apoyaron y ayudaron en cada momento difícil.

## DEDICATORIA RUTH

Esta monografía la dedico a mis padres que con su apoyo siempre me impulsaron a seguir adelante en mis estudios, a mis hermanos y a todos mis compañeros que estuvieron a mi lado en estos cinco años de carrera.

# AGRADECIMIENTOS


## AGRADECIMIENTO MARIELA

Mis agradecimientos son a Dios por la vida que me da, a mis padres que supieron dirigir cada uno de mis pasos, al Ing. Pablo Pintado que nos brindo su apoyo y sus conocimientos para poder realizar este trabajo y a cada persona que con su apoyo nos dieron fuerzas para culminar nuestro trabajo.

## AGRADECIMIENTO RUTH

En primer lugar agradezco a Dios, a mis padres y mi familia por todo el cariño, apoyo tanto económico como moral que siempre me dieron en todos estos años de carrera.

GRACIAS

# INDICE DE CONTENIDO

# INDICE DE GRAFICOS

# INDICE DE TABLAS

# RESUMEN

El desarrollo de software basado en componentes se ha convertido actualmente en uno de los mecanismos más efectivos para la construcción de grandes sistemas y aplicaciones de software, esta monografía está orientada a la investigación del uso e implementación de los componentes de software, el mismo que va a ser demostrado mediante un caso práctico que estará orientado a un centro educativo.

Estos componentes nos permitirán realizar el software reutilizando código que ya ha sido realizado, el mismo que solamente será implementado en el caso práctico.

# ABSTRACT

The software development based on components has now become one of the most effective mechanisms for the construction of large systems and applications software, this monograph is aimed at investigating the use and implementation of software components, which will to be demonstrated through a case study that will focus on a center educativo.Estos components will enable us to make reusing software code that has already been done, which will be implemented only in the case study.

# INTRODUCCION

La necesidad de realizar software en un menor tiempo ha conducido al personal a hacer uso de código previamente desarrollado, favoreciendo así al avance de lo que se conoce como Desarrollo de Software Basado en Componentes (DSBC).

Esta nueva disciplina se apoya en componentes software ya desarrollados, que son combinados adecuadamente para satisfacer los requisitos del sistema. Construir una aplicación se convierte por tanto en la búsqueda y ensamblaje de piezas prefabricadas, desarrolladas en su mayoría por terceras personas, y cuyo código no puede modificarse. Bajo este nuevo planteamiento, cobran especial interés los procesos de búsqueda y selección de los componentes apropiados para construir las aplicaciones. En este sentido, la tecnología de componentes ofrece ya soluciones robustas para muchos de los problemas que se plantean en la construcción de grandes sistemas de software.

Por lo general se han venido obviando muchos de los aspectos de calidad que intervienen a la hora de seleccionar componentes y ensamblarlos para construir aplicaciones que satisfagan los requisitos del cliente, por un lado estos aspectos pueden afectar a varias partes del sistema, y por ello tendrán prioridad si entran en conflicto con los requisitos funcionales de la aplicación.

Este trabajo se enfoca en la aplicación de estos componentes, para demostrar el DSBC y satisfacer las necesidades del cliente que en este caso será un centro educativo.

# CAPITULO 1: CONCEPTOS PREVIOS

## INTRODUCCION

En el siguiente capítulo se desarrollaran algunas definiciones sobre las cuales se basara el desarrollo de los siguientes capítulos.

Se tratara de aclarar algunas de las definiciones para la mejor comprensión del tema a desarrollar.

El concepto más importante que ha cambiado y sigue cambiando los procesos de ingeniería y reingeniería, es el concepto de "componente". Este concepto surge por la necesidad de reutilizar partes de software existentes que podían ser utilizadas para generar nuevas extensiones de aplicaciones, o para la generación de aplicaciones completas. Pero esto requeriría un gran esfuerzo, porque se tenía que localizar las partes reutilizables y almacenarlas en repositorios especiales que más tarde pudieran ser consultadas en fase de diseño.

**1.1 Definición 1 (Componente Szyperski, [Szyperski, 1998]):** 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.

Un componente de 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 descendemos en el nivel de detalle, se dice que un componente es de grano fino.

**1.2 Definición 2 (Componente IBM, [IBM-WebSphere, 2001]):** 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.

**1.3 Definición 3 (Componente SEI, [Brown, 1999]):** 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.

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.
En [Cheesman y Daniels, 2001] se 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 componentes.

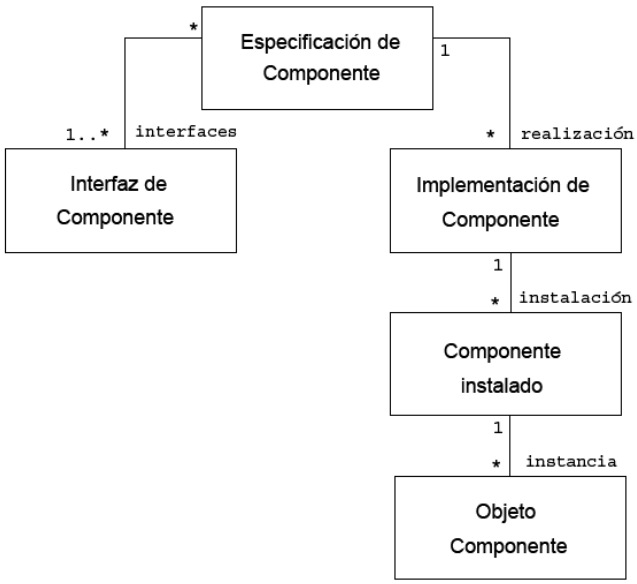



***Figura 1 FORMAS DE COMPONENTES***

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.

**1.4 Definición 4 (Componente EDOC, [OMG, 2001]):** Un componente es algo que se puede componer junto con otras partes para formar una composición o ensamblaje.

**1.5 Definición 5 COTS**: Un componente COTS es una unidad de elemento software en formato binario, utilizada para la composición de sistemas de software basados en componentes, que generalmente es de grano grueso, 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. Software que existe a priori, posiblemente en repositorios; está disponible al público en general; y puede ser comprado o alquilado.

**1.6 Definición 6 NDI:** Software desarrollado —inicialmente sin un intereses comercial— por unas organizaciones para cubrir ciertas necesidades internas, y que puede ser requerido por otras organizaciones. Por tanto es un software que existe también a priori, aunque no necesariamente en repositorios conocidos; está disponible, aunque no necesariamente al público en general; y puede ser adquirido, aunque más bien por contrato.

**1.7 Definición 7 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.

**1.8 Definición 8 ASSETS:** Recopilar y catalogar “módulos software” de aplicaciones para su reutilización, conocidos generalmente como “artefactos” o “módulos software”.

**1.9 Definición 9 REUTILIZACION:** 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.

**1.10 Definición 10 ISBC:** Es un proceso que concede particular importancia al diseño y la construcción de sistemas basados en computadoras que utilizan componentes de software reutilizables.

- Clements define a la ISBC como: *“la ISBC esta cambiando la forma en que se desarrollan los grandes sistemas de software. La ISBC encarna la filosofía de comprar,*

*no construir. En la misma forma como las primeras subrutinas liberaron al programador de pensar acerca de los detalles. La ISBC cambió el interés de programar software por el de componer sistemas de software. La implementación ha dado paso a la integración como centro del enfoque. En sus cimientos está la suposición de que existe suficiente base común entre muchos grandes sistemas de software para justificar el desarrollo de componentes reutilizables para explotarla y satisfacerla".*

Antes de continuar explicando el proceso de selección de componentes dentro de la ISBC vamos a dar el concepto de la palabra componentes que tanto se usa desde el inicio hasta el fin de esta exposición, **reutilización:** a la reutilización se la podría tomar como el paso más grande que se debe dar para tener una aproximación realista para llegar a los índices de productividad y calidad que la industria del software necesita según Brown y Wallnau nos dicen que un ***componente*** es la parte importante, casi independiente y reemplazable de un sistema que satisface una función clara en el contexto de una arquitectura bien definida.

Otros conceptos que también vamos a estar usando son los de: ***componente del software en ejecución*** paquete dinámico de unión de uno o más programas gestionados como unidad y a los cuales se tiene acceso por medio de interfaces documentales que se pueden descubrir en la ejecución, ***componente de software*** unidad de composición que sólo tiene dependencias de contexto explicitas y especificadas en forma establecida, ***componentes cualificados*** evaluados por ingenieros de software para garantizar que no solo la funcionalidad, sino el desempeño, la fiabilidad, la facilidad de uso y otros factores de calidad concuerdan con los requisitos del sistema o producto que se construirá, ***componentes adaptados*** adecuados para modificar características que no se requieren o indeseables, ***componentes actualizados*** sustituyen el software existente conforme están disponibles las nuevas versiones de los componentes. Estos conceptos también dados por Brown y Wallnau.

## CONCLUSION

Luego del desarrollo de este capítulo se han aclarado muchas de las definiciones que se van a utilizar en los capítulos siguientes.

Además han ayudado a la mejor comprensión y al conocimiento sobre el tema general de esta monografía.

# CAPITULO 2: INGENIERIA DE SOFTWARE BASADA EN COMPONENTES COTS

## INTRODUCCION

La reutilización de software se ha convertido en una actividad muy práctica que ha influido en la reducción de los costes, tiempos y esfuerzos en el proceso 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 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 hemos podido comprobar un aumento en el uso de componentes comerciales en prácticas de reutilización de software.

**2.1 DEFINICION:** 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 si (aunque un término no implica los otros), estando muy presentes en estos últimos años en ISBC.

Los componentes COTS (*Commercial-Off-The-Shelf*) se los denomina como una clase especial de componentes de software, normalmente de grano grueso, que presentan las siguientes características:

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

La reutilización de software se ha convertido en una actividad muy práctica que ha influido en la reducción de los costes, tiempos y esfuerzos en el proceso 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 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 hemos podido comprobar un aumento en el uso de componentes comerciales en prácticas de reutilización de software.

### 2.2 Limitaciones del desarrollo de software con componentes COTS

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.

### 2.3 Características de un componente comercial

Existen dos componentes más comunes en la literatura de COTS.

- ✓ Un componente COTS suele ser de grano grueso y de naturaleza de “caja negra” no existe la 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.
- ✓ 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

### 2.3.1 Ventajas

- ✓ Puede ser mucho más barato comprar un producto comercial, donde el costo de desarrollo ha sido amortizado por muchos clientes, que intentan desarrollar un nuevo software. 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.
- ✓ 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.

### 2.3.2 Desventajas

- ✓ Un componente comercial no tiene 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. También existen otros conceptos en cuestión de términos de caja negra, como ser: *Caja Blanca* donde se permite el acceso al código fuente de un componente para que sea reescrito y pueda operar con otros componentes. C*aja Gris*, donde el código fuente del componente no se puede modificar, pero proporciona su propio lenguaje de extensión o API. 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).
- ✓ 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 (las especificaciones).

| Ventajas | Desventajas |
|---|---|
| ✓ Disponibilidad inmediata; retorno de inversión inmediato | ✓ Licencias, propiedad intelectual |
| ✓ Evita costos de desarrollo y mantenimiento | ✓ Costos de licenciamiento y soporte periódicos |
| ✓ Funcionalidad muy rica | ✓ Demasiada funcionalidad compromete uso y rendimiento |
| ✓ Nuevas versiones continuamente | ✓ No existe control sobre las nuevas versiones |
| ✓ Organización dedicada al soporte | ✓ Dependencia del fabricante |
| ✓ Independencia de Hardware y Software | ✓ La integración no es trivial, los fabricantes no la garantizan |
| ✓ Reutilización (integración) de software | |
| ✓ Ciclos de desarrollo mas cortos | |
| ✓ Mejor ROI | |
| ✓ Funcionalidad mejorada (con mejores practicas) | |

***Tabla 1 VENTAJAS Y DESVENTAJAS DE COTS***

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 [Wallnau et al, 2002] [Dong et al., 1999] 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 software comerciales.

## 2.4 Sistemas basados en 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 siguiente se muestran algunas de las actividades en el desarrollo de software basado en componentes COTS [Meyers y Oberndorf, 2001]. 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 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 2 SISTEMAS BASADO EN COMPONENTES***

## 2.5 Ingeniería de requisitos y componentes COTS

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 lleva 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 esta 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 [Boehm, 1988] es el más utilizado en la construcción de sistemas basados en componentes COTS [Boehm, 2000] [Maiden y Ncube, 1998]

[Carney, 1999] [Saiedian y Dale, 2000]. 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 [Meyers y Oberndorf, 2001]:

— Análisis de la literatura 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 específicas 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 de [Lawlis et al., 2001], 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) [Maiden y Ncube, 1998] y ACRE (Acquisition of Requirements) [Maiden et al., 1997]. El primero utiliza un método basado en plantilla a tres niveles para la recogida de requisitos[15] 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) [Kontio et al., 1996], 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 [Franch, 1998], citado anteriormente.

Como adelantamos, 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 [Botella et al., 2001], NoFun permite definir diferentes tipos de componentes y relaciones entre componentes y subcomponentes, y adopta el estándar [ISO/IEC-9126, 1991] para tratar atributos de calidad.

También están los métodos relacionados con atributos de calidad para la selección de componentes COTS. En esta línea están conjuntamente los trabajos [Botella et al., 2002] y [Franch y Carvallo, 2002] donde se proponen unos modelos de calidad y de certificación de componentes COTS basados en el modelo de calidad de ISO [ISO/IEC-9126, 1991] y en el lenguaje NoFun como notación estructurada para la formalización de estos modelos.

Aquí también destacamos el trabajo [Bertoa y Vallecillo, 2002] que propone un modelo de calidad para componentes COTS inspirado en el modelo de ISO [ISO/IEC-9126, 1991] y que clasifica las características de calidad de un producto software.

Otros trabajos se centran en métodos para la selección de componentes COTS a partir de requisitos extra-funcionales. En esta línea está el trabajo [Rosa et al., 2001] que realiza un estudio formal en Z del proceso de desarrollo de software basado en componentes COTS considerando requisitos extra-funcionales. El estudio se centra en las fases de selección de componentes desde librerías COTS y ofrece una solución ante la dificultad de tratar los requisitos extra-funcionales en los procesos de selección.

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 destacamos el modelo COCOTS [COCOTS, 1999], basado en el modelo de construcción de software [COCOMO-II, 1999].

### 2.6 Propiedades de un Sistema Basado en Componentes COTS

Un sistema software basado en componentes COTS puede tener las siguientes propiedades:

- ✓ **Adaptable:** Las actualizaciones de la configuración de un componente es una tarea frecuente. 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.

- ✓ **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, 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.-

- ✓ **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 este sea extendido e integrado.

- ✓ **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.

## CONCLUSION

La falta de una información de especificación puede acarrear ciertos problemas al desarrollador que utiliza el componente COTS, como 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 software comerciales.

## CAPITULO 3: INGENIERIA DE SOFTWARE BASADA EN COMPONENTES (ISBC)

### INTRODUCCION:

En este capitulo mostraremos como es el proceso de la Ingeniería de Software Basada en Componentes (ISBC). Cuales son las partes que integran la ISBC como ser: la Ingeniería de Dominio y el Desarrollo Basado en componentes, los pasos que se deben seguir dentro de cada uno de ellos; mostrando como se realiza la selección de los componentes mas adecuados para la aplicación software, las preguntas mas frecuentes que se pueden surgir en el proceso de selección.

Así mismo luego de seleccionar a los componentes más adecuados se detallara como se realiza el proceso de calificación, adaptación y composición de los mismos que entren el marco de los requerimientos del sistema a crearse.

Mostraremos diferente modelos que nos pueden ayudar para que los componentes puedan funcionar en conjunto y también daremos unas pautas de cómo podríamos recuperar componentes que ya fueron creados con anterioridad y clasificarlos para nuevas aplicaciones en un futuro próximo. Y se pondrá en consideración el tema de las fábricas de Software para las personas que lean esta monografía y estén interesados en este tema.

Hablaremos de un tema que a todos nos interesa que es la economía, lo cuan costoso o beneficioso que puede ser utilizar la ISBC.

Y para concluir con este capítulo se mostrará tanto beneficios como desventajas que puede tener la ISBC.

Al hablar de la Ingeniería de Software Basada en Componentes (ISBC) podemos decir que desde siempre el concepto de la reutilización ha estado presente en los sistemas de computación, en la actualidad con el afán de construir sistemas más rápidos este conocimiento se ha explotado más ampliamente. El desarrollo de software basado en componentes permite reutilizar piezas de código preelaborado que permiten realizar diversas tareas, conllevando a diversos beneficios como las mejoras a la calidad, la reducción del ciclo de desarrollo y el mayor retorno sobre la inversión. La reutilización tiene ciertos beneficios como dificultades, entre las ventajas se podría decir que están: **Mejora de la productividad** *(disminución tiempo de desarrollo; disminución de costes)*. **Mejora de la calidad del software** *(mayor fiabilidad, mayor eficiencia)*.

Y entre las dificultades podemos mencionar: **En muchas empresas no existe plan de reutilización, Escasa formación, Resistencia del personal, Pobre soporte metodológico:** *uso de métodos que no promueven la reutilización, necesarios métodos para*: desarrollo para reutilización, desarrollo con reutilización.

Se puede reutilizar más que código fuente, se puede obtener beneficios mayores al reutilizar diseños y documentación asociada al código fuente reutilizable. Se puede reutilizar cualquier producto software obtenido en el ciclo de vida del software, con independencia de su nivel de abstracción como ser *especificaciones, diseños, código, pruebas, documentación, etc,* a estos elementos reutilizables se les llama *Asset o elemento de software reutilizable.*

**3.1 Tipos de Assets reutilizables:** Un asset puede encapsular cualquier abstracción útil producida durante el desarrollo de software y pueden ser:

- Planes de proyecto.
- Estimaciones de coste.
- Arquitectura.
- Especificaciones y modelos de requisitos.
- Diseños.
- Código fuente.
- Documentación de usuario y técnica.
- Interfaces hombre-máquina.
- Datos.
- Casos de prueba.

**3.2 Niveles de Reutilización:** pueden haber varios niveles de los objetos que pueden ser reutilizables, entre estos están:

- De código:

    *Librerías de funciones, editores, inclusión de ficheros, mecanismos de herencia en POO, componentes, etc.*

- De diseños

*No volver a inventar arquitecturas*

Por ej. patrones de diseño

Por ej. patrones arquitectónicos (C/S, pipeline, OO, etc.)

- De especificaciones

*Reutilización de las abstracciones del dominio*

*Debe estar asociada a la generación (semi)automática de los elementos de diseño e implementación.*

Con la explicación antes dada queremos dar un preámbulo de las ideas que se tiene en el mercado con lo referente al tema de reutilización de ciertos elementos dentro de la Ingeniería del Software basada en Componentes y por lo tanto en este trabajo vamos a detallar como la ISBC toma determinados componentes que tienen una funcionalidad independiente para hacerlos funcionar en un sistema integrado, para lograr esto la ISBC abarca dos actividades de ingeniería paralelas: *la ingeniería de dominio* y *el desarrollo basado en componentes*. La ingeniería de dominio explora un dominio de aplicación con el propósito único de encontrar componentes funcionales de comportamiento y de datos adecuados para la reutilización. A estos componentes se los ubica en unas librerías de reutilización. Por otro lado el desarrollo basado en componentes consigue los requerimientos de los clientes seleccionando un modelo arquitectónico adecuado para cumplir con los objetivos del sistema a construir.

A la ISBC se la puede asimilar con la Ingeniería del Software Orientada a Objetos convencional, ya que la ISBC comienza el proceso cuando se estable los requisitos que se necesitaran para el sistema que se va a construir en el que se aplica las técnicas convencionales de investigación, se establece un diseño arquitectónico, pero en el momento de ir hacia tareas de diseño más detalladas, se examina los requisitos para determinar qué elementos están directamente dispuestos para la composición y no para la construcción en este momento se puede decir que el desarrollador o desarrolladores se deben plantear las siguientes preguntas:

- ¿Hay componentes comerciales en línea (CDL) disponibles para implementar los requisitos?

- ¿Hay disponibles componentes reutilizables desarrollados internamente par implementar los requisitos?
- ¿Las interfaces para los componentes disponibles son compatibles dentro de la arquitectura del sistema que se construirá?

Tal vez los desarrolladores quieran modificar o eliminar aquellos requisitos del sistema que no sea posible implementar con CDL, si estos no se modificar o eliminar entonces se aplica los métodos de ingeniería del software en la construcción de aquellos nuevos componentes para satisfacer la aplicación que se está llevando a cabo.

Para los requisitos que se abordan con los componentes disponibles comienza un conjunto diferente de actividades de ingeniería del software: *cualificación, adaptación, composición y actualización.* Estas actividades vamos a detallar más cuando se hable del desarrollo basado en componentes.

### 3.3 Componente

Según Szyperski y Pfister dicen que: Un componente es una unidad 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.

#### 3.3.1 Características:

 Auto contenido

 Accesible solamente a través de su interfaz

 Inmutabilidad de sus servicios

 Documentación de sus servicios

 Reemplazable por otro componente

##### 3.3.1.1 Auto contenido

 Un componente no debe requerir la reutilización de otros componentes para cumplir su función.

 Si el componente requiere de otros, el conjunto de componentes o funciones, visto como un todo, debe ser considerado como un componente de más alto nivel.

#### 3.3.1.2 Es accesible sólo a través de su interfaz

 Es accedido a través de una interfaz claramente definida

 La interfaz debe ser independiente de la implementación física

 Debe ocultar los detalles de su diseño interno

#### 3.3.1.3 Sus servicios son inmutables

 Los servicios que ofrece un componente a través de su interfaz no deben variar

 La implementación física de estos servicios pueden ser modificadas, pero no deben afectar la interfaz.

#### 3.3.1.4 Está documentado

 Debe tener una documentación adecuada que facilite:

1. la recuperación del componente desde el repositorio,
2. la evaluación del componente,
3. su adaptación al nuevo ambiente y
4. su integración con otros componentes del sistema en que se reutiliza.

#### 3.3.1.5 Es reemplazable

 Puede ser reemplazado por una nueva versión o por otro componente que proporcione los mismos servicios.

**3.3.2 Meta-información:** Información adicional de un componente que suele hacerse pública. La idea es que con esta información un componente puede saber cómo utilizar otro componente

### 3.4 El proceso de ISBC

El proceso de ISBC no sólo se identifica los componentes candidatos sino que también cualifica la interfaz de cada componente, adapta los componentes para eliminar las equivocaciones arquitectónicas, ensambla los componentes conforme los requisitos del sistema cambian.

El proceso de ISBC toma el modelo de la ingeniería de dominio y la complementa con el desarrollo basado en componentes en la figura 1 se explica cómo se realiza esta acción.

### 3.5 Ingeniería del dominio

La ingeniería del dominio es identificar, construir, catalogar y diseminar un conjunto de componentes de software que sean aplicables para el software existente y futuro en un dominio de aplicación particular, con la finalidad de permitir a los ingenieros compartir dichos componentes para reutilizarlos.

La ingeniería del dominio incluye tres grandes actividades: análisis, construcción y diseminación.

#### 3.5.1 El proceso de análisis del dominio

Los pasos en el proceso se definen como:

1. Definir el dominio que se investigará.

2. Categorizar los elementos extraídos del dominio.

3. Recopilar una muestra representativa de las aplicaciones en el dominio.

4. Analizar cada aplicación en la muestra y definir las clases de análisis.

5. Desarrollar un modelo de análisis para las clases.

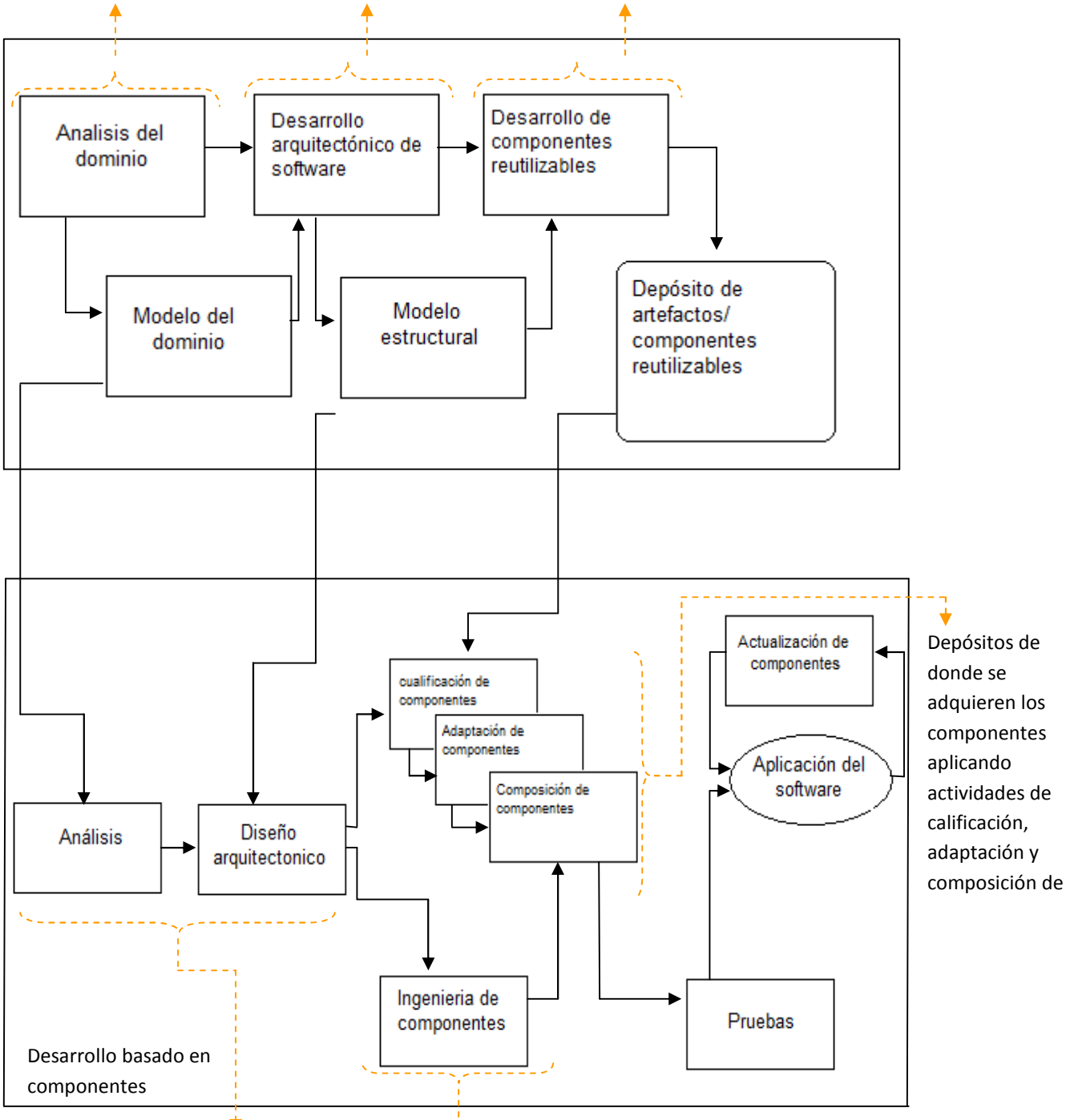
Hutchinson y Hindley nos dan unas preguntas que nos pueden servir de guía en el proceso de análisis de los componentes reutilizables más óptimos:

- ¿En las implementaciones futuras se requiere la funcionalidad del componente?
- ¿Cuán común es la función del componente dentro del dominio?

Crea un modelo de aplicación que se utiliza como base para analizar los requisitos del usuario

Una arquitectura genérica de software proporciona la entrada

Después de que los componentes reutilizables se han comprado, seleccionado de bibliotecas existentes o construidas están disponibles para los ingenieros.




Depósitos de donde se adquieren los componentes aplicando actividades de calificación, adaptación y composición de

Desarrollo basado en componentes

*Figura 2 MODELO DE PROCESO QUE APOYA LA ISBC*

El análisis y el diseño se pueden implementar en el contexto de un ADP. Un ADP implica que el modelo global del software se puede descomponer jerárquicamente

Se aplica cuando se requiere componentes especializados

- ¿Existe duplicidad de la función del componente dentro del dominio?
- ¿El componente depende del hardware? Si es así, ¿el hardware permanece invariable entre las implementaciones o las especificaciones del hardware pueden trasladarse hacia otro componente?
- ¿El diseño está lo suficientemente optimizado para la siguiente implementación?
- ¿Se pueden establecer parámetros respecto de un componente no reutilizable de modo que se vuelva reutilizable?
- ¿El componente es reutilizable en muchas implementaciones sólo con cambios menores?
- ¿Es factible la reutilización por medio de la modificación?
- ¿Un componente no reutilizable se puede descomponer para producir componentes reutilizables?
- ¿Cuán valida es la descomposición de un componente para la reutilización?

También existen otras propuestas para llevar a cabo el proceso de Ingeniería de Dominio: entre una de ellas se encuentra FODA (*Feature-Oriented Domain Analysis*)

### 3.5.2 FODA:

#### 3.5.2.1 Identificación del dominio y del alcance

La mayoría de las aplicaciones consisten en varios subsistemas o subproblemas distintos y reconocibles, aunque sólo algunos de ellos son económicamente reutilizables. Por lo tanto es importante decidir qué partes son válidas para un futuro tratamiento.

#### 3.5.2.2 Selección y análisis de ejemplos, necesidades y tendencias

Hay un delicado equilibrio entre la reutilización reactiva y proactiva. Un conjunto de elementos reutilizables debe anticiparse a las necesidades futuras. Dado que esto es difícil de hacer, y es la razón por la que construir software reutilizable es más caro que construir software convencional. El proceso de reutilización debe encontrar los elementos esenciales comunes y la variabilidad, así como dar prioridad a las partes que

deben ser tenidas en cuenta en la reutilización. La mayoría de las aproximaciones seleccionan ejemplos clave y extraen sus conjuntos de *features* esenciales. Los ejemplos relacionan el ámbito del dominio con la evaluación de las necesidades de los usuarios, el mercado, la tecnología y las tendencias del negocio.

#### 3.5.2.3 Identificación, recolección y agrupación de los conjuntos de *features*

Utilizando modelos de análisis, tablas y/o gráficos, las features que aparecen juntas (AND) o son variantes que seleccionar (OR, XOR) se estructuran en un marco de decisión, de esta forma la terminología del dominio es almacenada. En FODA un modelo de información describe las entidades de dominio y las relaciones entre ellas, un modelo de comportamiento describe las relaciones dinámicas entre las entidades del dominio. El modelo funcional de FODA, describe el flujo de datos entre las entidades. El modelo de features mantiene todos los modelos juntos estructurando y relacionando los conjuntos de features.

#### 3.5.2.4 Desarrollo de un modelo y una arquitectura de dominio o genéricos

A partir de estos conjuntos de features, un modelo de dominio resume las características esenciales de todas o algunas de las aplicaciones del dominio. También se desarrolla una arquitectura del sistema que relaciona los mecanismos principales las features, los subsistemas y las variantes. La arquitectura cubre los subsistemas y las aplicaciones resultantes, define los servicios principales, especifica las interfaces de forma precisa y sirve como modelo de referencia y como anteproyecto funcional.

#### 3.5.2.5 Representación de las partes comunes y la variabilidad

Se identifican los subsistemas, módulos y funciones genéricas, relacionándose entre ellas mediante generalizaciones, especializaciones o alternativas.

#### 3.5.2.6 Explotación de las partes comunes y la variabilidad

Se emplean notaciones y mecanismos para especificar diferentes clases de productos genéricos o parametrizados.

#### 3.5.2.7 Implementación, certificación y empaquetado de los elementos reutilizables

El subconjunto más importante de componentes reutilizables (assets) candidatos se implementan y se distribuyen como elementos reutilizables certificados, bajo una política de gestión de la configuración. El resto de elementos reutilizables serán implementados cuando se necesiten.

### 3.5.3 Funciones de caracterización

Aquí se requiere definir un conjunto de características del domino que comparta todo el software dentro de un dominio. Una característica del dominio define algún atributo genérico de todos los productos que existen dentro de él.

Un conjunto de características de domino de un componente reutilizable se puede representar como:

1. No es relevante si la reutilización es apropiada.
2. Relevante sólo en circunstancias inusuales.
3. Relevante el componente se modifica para usarlo a pesar de las diferencias.
4. Claramente relevante, y si el nuevo software no tiene esta característica, la reutilización será ineficiente pero tal vez sea posible.
5. Claramente relevante, y si el nuevo software no tiene esta característica, la reutilización será ineficiente y la reutilización sin dicha característica no se recomienda.

Cuando se requiere construir un nuevo software se comparan las características existentes contra las características que el nuevo sistema requiere y ver si la reutilización va a ser eficaz en el nuevo sistema. A pesar de que el nuevo sistema existe dentro del dominio de aplicación, los componentes reutilizables deben ser analizados para determinar su aplicabilidad.

El proceso de ISBC incluye dos subprocesos concurrentes: la ingeniería del dominio y el desarrollo basado en componentes. La finalidad de la ingeniería del dominio es identificar, construir, catalogar y diseminar un conjunto de componentes de software en un dominio de aplicación específico. Entonces el desarrollo basado en componentes califica, adapta e integra dichos componentes para emplearlos en un nuevo sistema. Además, el desarrollo basado en

componentes diseña los componentes nuevos que se basan en los requisitos personalizados de un sistema nuevo.

### 3.6 Modelado estructural y puntos de estructura

El modelado estructural es un enfoque de ingeniería del dominio basada en patrones que funciona bajo la premisa de que cualquier dominio de aplicación tiene patrones repetitivos (de función, datos y comportamiento) que tiene un potencial de reutilización.

McMahon describe un punto de estructura como una estructura distinta dentro de un modelo estructural. Los puntos de estructura tienen tres características distintas:

(1) Un punto de estructura es una abstracción que debe tener un número limitado de instancias. Además, la abstracción debe recurrir a través de las aplicaciones en el dominio. De otro modo no se justifica el costo de verificar, documentar y diseminar el punto de estructura.

(2) Las reglas que rigen el uso del punto de estructura deben comprenderse con facilidad. Además, la interfaz para el punto de estructura debe ser relativamente simple.

(3) El punto de estructura debe implementar la ocultación de información al aislar toda la complejidad dentro del mismo punto de estructura. Esto reduce las complejidades percibidas del sistema globales conjunto.

Con esto se puede definir puntos de estructura genéricos que trasciendan diferentes dominios de aplicación.

*Aplicación frontal*: la GUI (graphic user interface) que incluye todos los menús, paneles y entradas y ordena las funciones de edición.

*Bases de datos:* el depósito para todos los objetos relevantes respecto del dominio de la aplicación.

*Motor de cálculo:* los modelos numéricos y no numéricos que manipulan datos.

*Función de generación de informes:* la función que produce salidas de cualquier tipo.

*Editor de aplicaciones:* el mecanismo para personalizar la aplicación respecto a las necesidades de usuarios específicos.

### 3.7 Desarrollo basado en componentes

El desarrollo basado en componentes (DBC) es una actividad de ISBC que ocurre en paralelo con la ingeniería del dominio. Al aplicar los métodos de diseño de análisis y arquitectónico el equipo de software refina un estilo arquitectónico apropiado para el modelo de análisis creado para la aplicación que se construirá.

Una vez que la arquitectura se ha establecido, deben agregársela componentes que

1* estén disponibles en bibliotecas de reutilización.

2* sean diseñados para satisfacer las necesidades personales del cliente. Por lo tanto el flujo de tareas para el desarrollo basado en componentes tiene dos trayectorias paralelas.

Los componentes reutilizables están disponibles para su potencial integración en la arquitectura tienen que cualificarse y adaptarse. Si se requieren nuevos componentes es preciso diseñarlos. Entonces los componentes resultantes se componen en la plantilla arquitectónica y se prueban en forma minuciosa.

#### 3.7.1 Calificación, adaptación y composición de componentes

En la ingeniería del dominio proveemos la biblioteca de componentes reutilizables, pero esto no garantiza que todos los componentes se puedan integrar con facilidad, por esta razón se aplica una sucesión de actividades de desarrollo basado en componentes.

**3.7.1.1 Calificación de componentes:** esta actividad garantiza que el componente candidato realizará la función requerida, encajará adecuadamente en el estilo arquitectónico que especifica el sistema y mostrará las características de calidad (como por ejemplo desempeño, fiabilidad, facilidad de uso) que requiere la aplicación.

Entre los muchos factores considerados durante la calificación de componentes están: interfaz de programación de la aplicación; herramientas de desarrollo e integración que requiere el componente; requisitos de tiempo de ejecución, que incluyen uso de recursos como memoria o almacenamiento, tiempos o velocidad y protocolo de red; requisitos de servicio, que incluyen interfaces de sistema operativo y apoyo de otros componentes; características de seguridad, que incluyen controles de acceso y protocolos de autenticación; suposiciones de diseño anidado, que incluyen el empleo de algoritmos numéricos o no numéricos específicos; y manejo de excepciones.

**3.7.1.2. Adaptación de componentes:** la implicación de la integración fácil es que 1* se han implementado métodos consistentes de gestión de recursos para todos los componentes en la biblioteca, 2* existen actividades comunes como la gestión de datos para todos los componentes, y 3* se han implementado interfaces dentro de la arquitectura y con el entorno externo en una forma consistente.

Es posible que haya conflictos en una o más de las áreas indicadas, estos problemas usualmente se evitan utilizando una técnica de adaptación llamada *encubrimiento de componente*. Cuando un equipo de software tiene pleno acceso al diseño interno y el código de un componente se aplica el *encubrimiento de caja blanca*, el encubrimiento de caja blanca examina los detalles de procesamiento interno del componente y hace modificaciones en el código para eliminar cualquier conflicto.

El *encubrimiento de caja gris* se aplica cuando la biblioteca de componentes proporciona un lenguaje de extensión de componente que permite eliminar o enmascarar los conflictos.

El *encubrimiento de caja negra* requiere la introducción de pre y pos procesamiento en la interfaz del componente para eliminar o enmascarar los conflictos.

**3.7.1.3 Composición de componentes:** la tarea de composición de componentes ensambla componentes cualificados, adaptados y diseñados con el fin de agregárselos a la arquitectura establecida para la aplicación. Esto se logra estableciendo una infraestructura que una los componentes en un sistema operativo, esta proporciona un modelo para coordinar los componentes y los servicios específicos que permiten que los componentes se coordinen mutuamente y realicen tareas comunes.

Hay un conjunto de cuatro ingredientes arquitectónicos que debe estar presente:

**3.7.1.3.1 Modelo de intercambio de datos**: se deben definir mecanismos que permitan a los usuarios y aplicaciones interactuar y transferir datos, no sólo transferencia de datos humano-software y componente, sino también la transferencia entre recursos del sistema.

**3.7.1.3.2 Automatización:** se deben implementar varias herramientas, macros y guiones para facilitar la interacción entre componentes reutilizables.

**3.7.1.3.3 Almacenamiento estructurado:** los datos heterogéneos que contienen un documento compuesto deben estar organizados y ofrecer acceso como una sola estructura de datos y no como una colección de archivos separados.

**3.7.1.3.4 Modelo de objeto subyacente:** el modelo de objeto asegura que los componentes deben ser capaces de comunicarse a través de una red. Esto se logra si el modelo de objeto define un estándar para la interoperabilidad de los componentes.

- **OMG/CORBA:** CORBA es un distribuidor de objetos que permiten que los componentes reutilizables se comuniquen con otros componentes, sin importar su ubicación dentro de un sistema.

- **COM de Microsoft:** COM es un modelo de objeto para componentes que ofrece una especificación para utilizar componentes producidos por varias empresas dentro de una sola aplicación que corra bajo el sistema operativo Windows, incluye dos elementos: interfaces COM y un conjunto de mecanismos que registra y pasa mensajes entre interfaces COM.

- **Componentes Sun JavaBeans:** esta es una infraestructura de ISBC portátil e independiente de la plataforma que utiliza y desarrolla empleando el lenguaje de programación Java, incluye un conjunto de herramientas, llamado *Kit de desarrollo Bean,* que permite a los desarrolladores: 1* analizar como funcionan los Beans existentes, 2* personalizar sus comportamiento y apariencia, 3* establecer mecanismos para coordinación y comunicación, 4* desarrollar Beans personalizados para usarlos en una aplicación específica, y 5* probar y evaluar el comportamiento Bean.

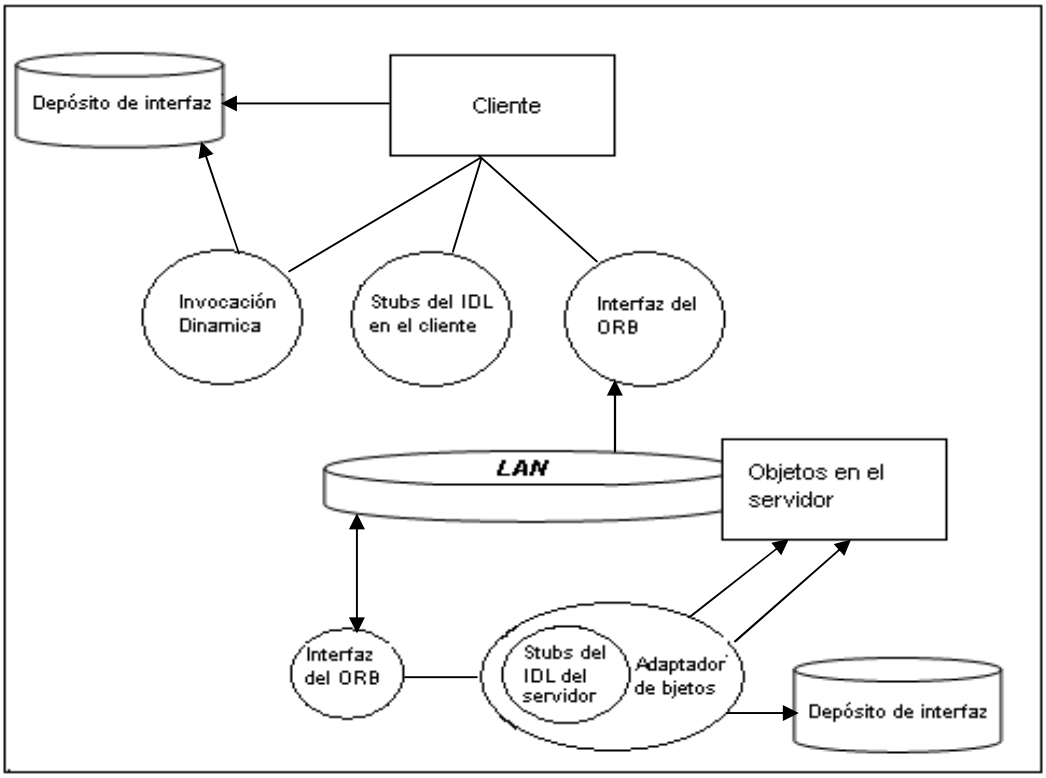



*Figura 3 ESTRUCTURA BÁSICA DE UNA ARQUITECTURA CORBA*

*Este es un sistema cliente-servidor, el objeto servidor se define usando un lenguaje de descripción de interface que permite que el ingeniero defina objetos, atributos, métodos y mensajes que se requieren para invocarlo. El cliente crea Stubs del IDL para comunicarse con el método residente en el servidor, estos proporcionan la compuerta a través de la que se acomodan las peticiones de objetos a lo largo del sistema cliente-servidor.*

#### 3.7.1.4 CORBA

CORBA es una especificación la cual define una infraestructura para la arquitectura OMA (Object Management Arquitecture) de OMG(Object Management Group), especificando los estándares necesarios para la invocación de métodos sobre objetos en entornos heterogéneos. La OMG es fundada en 1989, por American Airlines, Canon, Data General, HP, Philips Telecomunicaciones, Sun, 3Com y Unisys.

Las implementaciones de CORBA son conocidas como ORB (Object Common Broker). Los entornos heterogéneos son aquellos en los que las arquitecturas que constituyen el entorno pueden ser sistemas Microsoft Windows, nix de diferentes fabricantes. Y es más, dentro de la heterogeneidad también se incluyen los sistemas de comunicaciones (protocolos

de comunicación como TCP/IP, IPX) o los lenguajes de programación utilizados en las diferentes arquitecturas.

CORBA define su propio modelo de objetos, basado en la definición de las interfaces de los objetos mediante el lenguaje IDL.

De esta forma se logra una abstracción de la heterogeneidad que permite que el uso de CORBA no sea nada complejo. CORBA sigue una metodología concreta y fácil de seguir.

CORBA ha logrado parte de su éxito a la clara separación entre la interfaz de los objetos y la implementación de los mismos. Las interfaces se definen utilizando el lenguaje IDL, cuya principal característica es su alto nivel de abstracción, lo que le separa de cualquier entorno de desarrollo específico. Para la implementación de los objetos se puede utilizar cualquier lenguaje de programación que proporcione enlaces con el lenguaje IDL.

Para que un lenguaje de programación se pueda utilizar desde CORBA, debe tener definida la forma de enlazarse con IDL. De esta forma, y a partir de una especificación de las interfaces en IDL, se generan unos cabos (proxies) en el lenguaje elegido que permiten el acceso a la implementación de los objetos desde la arquitectura CORBA.

Dos aspectos de la arquitectura de CORBA sobresalen:

- ✓ Las implementaciones tanto del cliente como del objeto están aisladas del Object Request Broker (ORB) por una interfaz IDL, esto para garantizar la substitubilidad de las implementaciones para cada interfaz.
- ✓ Las solicitudes no pasan directamente del cliente al objeto, siempre son administradas por un ORB. Lo que hace que los detalles de la distribución sean transparentes para los clientes y para los objetos.

La arquitectura de administración de objetos propuesta por OMG define una arquitectura común que hace posible que exista un sistema unificado de computo basado en componentes heterogéneos que están interoperando entre si. Los servicios proporcionados por CORBA proporcionan servicios necesarios por casi todo sistema basado en objetos, las facilidades de CORBA permiten un acceso estandar a tipos de datos comunes y la funcionalidad necesaria para grupos de aplicaciones corporativos y específicos para las aplicaciones industriales.

#### 3.7.1.4.1 ¿Qué es un objeto CORBA?

Un objeto CORBA cumple con:

- ✓ *Encapsulamiento*: Un modulo de software encapsulado consiste de dos partes distintas; su interfaz que es lo que el módulo representa para el mundo exterior, y su implementación que es mantenida privada. La encapsulación también le permite a CORBA proporcionar la transparencia en la localización, ya que los clientes envían sus solicitudes a su ORB local, y no al objeto destino.
- ✓ *Herencia*: Es posible crear un nuevo objeto adaptando uno existente a crear uno partiendo de cero.
- ✓ *Polimorfismo*: Las operaciones pueden pertenecer a más de un clase de objeto.

#### 3.7.1.4 2 El IDL

El lenguaje de definición de interfaz o IDL (*Interface Definition Language*), es un lenguaje muy sencillo utilizado para definir interfaces entre componentes de aplicación. Es importante destacar que IDL sólo puede definir interfaces, no implementaciones. IDL, al especificar las interfaces entre objetos CORBA, es el instrumento que asegura la independencia del lenguaje de programación utilizado.

En el momento de construir un objeto CORBA el primer paso es definir cual va a ser la funcionalidad que va a proporcionar, para de esta manera poder escribir la interfaz en IDL. Toda la información necesaria para construir un cliente del objeto es proporcionada por la interfaz. Se debe escoger un lenguaje de programación que facilite la implementación de la interfaz de cada objeto.

#### 3.7.1.4.3 El ORB

Una parte fundamental de la arquitectura CORBA es el ORB, componente software cuyo fin es facilitar la comunicación entre objetos. El ORB se encarga de enviar las peticiones a los objetos y retornar las respuestas a los clientes que invocan las peticiones.

La principal característica del ORB es la transparencia, cómo facilita la comunicación cliente/servidor. Generalmente, el ORB oculta lo siguiente:

- ✓ Ubicación de los objetos. El cliente no sabe dónde se encuentra el objeto destino. Puede residir en un proceso diferente en otra máquina a través de la red, o dentro del mismo proceso
- ✓ Implementación de los objetos. El cliente no sabe cómo esta implementado el objeto remoto, en qué lenguaje de programación o de *scripts* está escrito, o el sistema operativo y el hardware, sobre el que se ejecuta.
- ✓ Estado de ejecución del objeto. Cuando el cliente lanza una petición sobre un objeto remoto, no necesita saber si el objeto está en ese momento en ejecución, y listo para aceptar peticiones. El ORB de forma transparente inicializa el objeto en caso de ser necesario, antes de enviarle la petición
- ✓ Mecanismos de comunicación de los objetos. El cliente no sabe qué mecanismos de comunicación (por ejemplo, TCP/IP, memoria compartida, llamada a método local) utiliza el ORB para enviar la petición al objeto y retorna la respuesta al cliente.

Estas características del ORB permiten a los desarrolladores de aplicaciones preocuparse más por las cuestiones propias del dominio de sus aplicaciones y desentenderse de las cuestiones de programación a bajo nivel del sistema distribuido.

La idea de un ORB es la siguiente, cuando un componente de aplicación quiere utilizar un servicio proporcionado por otro componente, primero debe obtener una referencia para el objeto que proporciona ese servicio. Después de obtenerla, el componente puede llamar a los métodos en ese objeto, accediendo así a los servicios proporcionados por éste; evidentemente el programador del componente cliente debe saber en tiempo de compilación que métodos están disponibles por un objeto servidor particular. La principal responsabilidad de ORB es resolver las peticiones por las referencias a objetos, posibilitando a los componentes de aplicación establecer conectividad entre ellos.
Cuando se crea un objeto CORBA también se crea una referencia para él. Cuando es utilizada por un cliente, la referencia siempre -durante toda la vida del objeto-, se refiere a dicho objeto para la que fue creada. En otras palabras, la referencia a un objeto siempre hace referencia a un único objeto. Las referencias a objetos son inmutables y opacas, de esta forma un cliente no puede manipular una referencia y modificarla. Las referencias a objetos pueden tener un formato estándar o propietario. Los clientes pueden obtener las referencias a objetos de muy diversas formas:

- ✓ En la creación de objetos. El cliente puede crear un nuevo objeto y conseguir así una referencia al objeto.
- ✓ A través del servicio de directorio. El cliente puede invocar a un servicio de búsqueda de cualquier tipo, con el fin de obtener referencias a objetos. Estos servicios no crean nuevos objetos, sino que almacenan referencias a objetos e información asociada (por ejemplo, nombres y propiedades) para los objetos existentes, y los proporcionan previa solicitud.
- ✓ Convirtiendo la referencia al objeto en una cadena y recuperándola. Una aplicación puede solicitar al ORB que convierta una referencia a un objeto en una cadena, y esta cadena almacenarla en un fichero o base de datos. Más tarde, la cadena puede ser recuperada y transformada nuevamente en una referencia a un objeto por el ORB.

#### 3.7.1.4.4 *Distribución de los objetos*

Para un cliente CORBA, una llamada a procedimiento remoto se ve exactamente igual a la llamada a un método local. Así, la naturaleza distribuida de los objetos CORBA es transparente a los usuarios de dichos objetos; los clientes no son conscientes de que están realmente tratando con objetos distribuidos en una red.

Debido a que la distribución de objetos supone una mayor probabilidad de fallo (caída de un servidor, caída de un segmento o enlace de la red…), CORBA debe ofrecer mecanismos para manejar dichas posibles situaciones. Para ello, proporciona un conjunto de excepciones, que pueden ser lanzadas por cualquier método remoto.

#### 3.7.1.4.5. *Referencias a objetos*

En una aplicación distribuida, hay dos métodos posibles para que el componente de una aplicación obtenga el acceso a un objeto en otro proceso.
Un método es conocido como paso por referencia. Mediante este mecanismo cuando un proceso invoca un método sobre un objeto en un proceso remoto, el método es ejecutado por dicho proceso remoto pues el objeto existe en la memoria y espacio de procesos de dicho proceso. El pasar un objeto por referencia significa que un proceso admite la visibilidad de uno de sus objetos (a través de las referencias a esos objetos) en otro proceso mientras conserva la propiedad de ese objeto.

El segundo método para pasar un objeto entre componentes de una aplicación se conoce como paso por valor. Mediante este mecanismo, el actual estado del objeto (así como los valores de sus atributos) es pasado al componente que lo solicita. Cuando los métodos de un objeto son invocados por un proceso remoto, son ejecutados por dicho proceso en vez de por aquel en el que reside el objeto original. Es más, puesto que el objeto es pasado por valor, el estado del objeto original no cambia; sólo la copia es modificada.

#### 3.7.1.4.6 *Los adaptadores de objeto*

El estándar CORBA describe un cierto número de adaptadores de objeto, cuya principal tarea es la de hacer de interfaz entre la implementación de un objeto y su ORB, puesto que se trata de mantener el ORB tan sencillo como sea posible. El ORB, en efecto, sólo ha de proporcionar una infraestructura de comunicación y activación para aplicaciones de objetos distribuidos.

Los adaptadores de objeto son, en otras palabras, un objeto interpuesto que usa la delegación para permitir realizar invocaciones sobre un objeto, incluso aunque el innovador no conozca la verdadera interfaz del objeto. Entre las responsabilidades de los adaptadores de objeto, cabe destacar las siguientes:

- ✓ La demultiplexación de solicitudes al servidor adecuado, es decir, la localización del servidor.
- ✓ El envío de la operación solicitada al servidor.
- ✓ La activación y desactivación de objetos.
- ✓ La generación de referencias a objetos.

Actualmente CORBA sólo proporciona un adaptador de objeto, conocido por adaptador de objeto básico. El adaptador de objeto básico suministra objetos CORBA con un conjunto común de métodos para acceder a las funciones del ORB y trata de ser un adaptador de objeto multipropósito que pueda soportar varios estilos e implementaciones de servidores, resultando ser muy básica en ciertas áreas. Esto ha supuesto baja eficiencia y baja portabilidad, pues cada vendedor ORB ha optado por cubrir dichas áreas con implementaciones propias. No obstante el OMG se ha percatado de este problema y está trabajando activamente en él.

#### 3.7.1.4.7 El modelo de comunicaciones CORBA

- ✓ ***Protocolos entre ORBs***

La especificación CORBA es independiente de los protocolos de transporte; el estándar CORBA especifica el conocido como GIOP (*General Inter-ORB Protocol*). GIOP especifica, a alto nivel, un estándar para la comunicación entre varios componentes CORBA ORBs.

GIOP, es sólo un protocolo general; el estándar CORBA también determina protocolos adicionales, que especializan GIOP para utilizar un protocolo de transporte en particular. Por ejemplo, los protocolos basados en GIOP existen para TCP/IP y DCE. Adicionalmente, los vendedores pueden definir y utilizar protocolos propietarios para la comunicación entre componentes CORBA.

El protocolo basado en GIOP más importante es el destinado a redes TCP/IP, conocido como IIOP (*Internet Inter-ORB Protocol*). Los vendedores tienen que implementar el protocolo IIOP para ser considerados conformes a CORBA, aunque pueden ofrecer además sus protocolos propietarios. Este requerimiento ayuda a asegurar la interoperabilidad entre los productos CORBA de diferentes vendedores pues cada producto conforme a CORBA 2.2 debe ser capaz de hablar el mismo lenguaje. Algunos vendedores han adoptado IIOP como protocolo nativo de sus productos en vez de utilizar un protocolo propietario; sin embargo, se permite que un ORB soporte cualquier número de protocolos, siembre y cuando IIOP se soporte, es decir, al comunicarse entre sí, los ORBs pueden negociar que protocolo utilizar. Adicionalmente algunos fabricantes están incorporando IIOP en productos como servidores de bases de datos o herramientas de desarrollo de aplicaciones o navegadores Web.

- ✓ ***CORBA y el modelo de trabajo en red***

Esencialmente, las aplicaciones CORBA se realizan sobre los protocolos derivados de GIOP como IIOP. Estos protocolos van sobre TCP/IP, DCE o cualquier otro protocolo de transporte que utilice la red. Las aplicaciones CORBA no están limitadas a utilizar únicamente uno de estos protocolos; la arquitectura de una aplicación puede ser diseñada para utilizar un puente que interconecte, por ejemplo, componentes de una aplicación basados en DCE con otros basados en IIOP. Es decir, máquinas que suplantan los protocolos de la red de transporte. CORBA es una arquitectura que crea otra capa (la capa del protocolo entre ORBs) que utiliza la

capa de transporte subyacente. Esto es también un punto determinante de la interoperabilidad entre aplicaciones CORBA, CORBA no dicta la utilización de una capa de transporte en particular.

- ✓ ***Clientes y servidores CORBA***

Tradicionalmente, en una aplicación cliente/servidor, el servidor es el componente o componentes que proporciona servicios a otros componentes de la aplicación. El cliente es el componente que hace uso de los servicios suministrados por un servidor o servidores.

La arquitectura de una aplicación CORBA no es diferente; generalmente, ciertos componentes de una aplicación proporcionan servicios que son utilizados por otros componentes de otra aplicación. El papel del cliente y servidor puede intercambiarse temporalmente, ya que un objeto CORBA puede participar en múltiples interacciones simultáneamente.

En una aplicación CORBA, cualquier componente que proporciona la implementación para un objeto es considerado un servidor. El hecho de ser un servidor CORBA significa que, el componente (el servidor), ejecuta métodos para un objeto particular, en nombre de otros componentes (los clientes).

- ✓ ***Stubs y skeletons***

Después de que el programador cree las definiciones de interfaz del componente utilizando IDL, dicho desarrollador procesa los ficheros IDL resultantes con un compilador IDL. El compilador crea los conocidos por *client stubs* y *server skeletons*. Los *client stubs* y *server skeletons* sirven como una clase de "pegamento" que conecta las especificaciones de interfaz IDL independientes del lenguaje con un código de implementación específico del lenguaje.

Los *client stubs* para una interfaz en particular se suministran para su inclusión con los clientes que utilizan dichas interfaces. El *client stub* para una interfaz en particular, proporciona una falsa implementación para cada uno de los métodos de dicha interfaz. En vez de ejecutar la funcionalidad del servidor, los métodos del *client stub* simplemente se comunican con el ORB para clasificar y desclasificar los parámetros.

Por otro lado, se tienen los *server skeletons*, proporcionando el esqueleto sobre el que se construirá el servidor. Para cada método en la interfaz, el compilador IDL genera un método vacío en el *server skeleton*. El desarrollador después proporciona una implementación para cada uno de esos métodos.

Los *stubs* y *skeletons* han sido generalizados a un DII (*Dynamic Invocation Interface*) y DSI (*Dynamic Skeleton Interface*), respectivamente. Ambas son interfaces proporcionadas directamente por ORB, y son independientes de las interfaces IDL de los objetos que están siendo invocados. Utilizando DII una aplicación cliente puede invocar peticiones, en cualquier objeto, sin tener conocimiento en tiempo de compilación de las interfaces del objeto. De forma semejante, DSI permite a los servidores ser codificados, sin tener *skeletons* para los objetos que están siendo invocados, siendo compilados estáticamente en el programa.

### 3.8 Ingeniería de componentes

La ISBC apoya al uso de componentes existentes, pero hay muchas ocasiones en que los componentes deben ser desarrollados e integrarlos con los CDL (componentes comerciales en línea) ya existentes y con los componentes de desarrollo propio. Puesto que los nuevos componentes se integran a la biblioteca propia de componentes reutilizables, deben diseñarse para su reutilización.

Los conceptos de diseño tales como abstracción, ocultación, independencia funcional, refinamiento y programación estructurada, junto con métodos orientados a objeto, pruebas de SQA y métodos de verificación de corrección, todos contribuyen a al creación de componentes de software reutilizables.

### 3.9 Análisis y diseño para la reutilización

En este paso la situación en discusión es extraer información a partir del modelo de requisitos en una forma que conduzca a la concordancia de especificaciones, si esta concordancia produce componentes que se ajustan con las necesidades de la aplicación actual, el diseñador puede extraer dichos componentes de una biblioteca de reutilización. Si no encuentra componentes de diseño, el ingeniero de software debe aplicar métodos de diseño convencional u Orientado a Objetos para crearlos, aquí se debe considerar el diseño para la reutilización (DPR).

El DPR requiere que el ingeniero de software aplique sólidos conceptos y principios de diseño de software, se deben considerar las características del dominio de la aplicación.

**3.9.1 Datos estándar:** se debe investigar el dominio de la aplicación e identificar las estructuras de datos globales, como ser, estructuras de archivos o una base de datos completa.

**3.9.2 Protocolo de interfaz estándar:** hay tres protocolos de interfaz: la naturaleza de las interfaces intramodulares, diseño de interfaces técnicas externas y la interfaz hombre-máquina.

**3.9.3 Plantillas de programa:** el modelo de estructura sirve como una plantilla para el diseño arquitectónico de un programa nuevo.

Una vez establecidas las interfaces, los datos estándar y las plantillas de programa, el diseñador tiene un marco de trabajo en el que puede crear el diseño.

### 3.10 Clasificación y recuperación de componentes

Cuando un ingeniero requiere elegir componentes tiene muchas opciones de donde escoger ¿cómo saber cuál es el mejor?, ¿cómo se describen los componentes de software en términos clasificables y sin ambigüedades?, se podría explorar ciertas tendencias para responder estas inquietudes que se presenta en este proceso.

**3.10.1 Descripción de los componentes reutilizables:** la mejor manera de describir un componente es utilizando el *modelo 3C* el cual consta de concepto, contenido y contexto.

El *contexto* de un componente de software es una descripción de lo que hace el componente. La interfaz del componente esta claramente descrita y la semántica identificada.

El *contenido* de un componente describe cómo se construye el concepto, es decir es información oculta para los usuarios habituales y que sólo la conocen las personas que requieren modificar o probar el componente.

El *contexto* sitúa un componente de software reutilizable en su dominio de aplicabilidad.

Estos métodos se pueden clasificar en: métodos de biblioteconomía y de ciencias de la comunicación, métodos de inteligencia artificial y sistemas de hipertexto, en gráfico 3 se presenta una taxonomía de los métodos de indexación en la biblioteconomía. Los vocabularios

controlados de indexación limitan los términos o sintaxis con que se clasifican un componente. Los vocabularios de indexación no controlados no ponen restricciones en la naturaleza de la descripción. La mayoría de los esquemas de clasificación para los componentes de software se incluyen en tres categorías:

**3.10.1.1 Clasificación enumerada:** los componentes se describen mediante una estructura jerárquica en la cual se definen las clases y los niveles variables de subclases de los componentes de software.

**3.10.1.2 Clasificación por facetas:** se analiza un área del dominio y se identifica un conjunto de características descriptivas básicas. Estas características llamadas facetas se clasifican según su importancia y se conectan con un componente.

Una faceta describe la función que el componente realiza, los datos que se manipulan, y otras características. El conjunto de facetas se denomina descriptor de facetas, porque se limitan a no más de siete u ocho facetas.

**3.10.1.3 Clasificación de valores y atributos:** un conjunto de atributos se define para todos los componentes en cierta área del dominio. Enseguida se asignan valores a dichos atributos en una forma muy similar a la clasificación por facetas. La clasificación de valores y atributos es similar a la clasificación por facetas, con las siguientes excepciones: 1* no se limita el número de atributos que se pueden utilizar, 2* no se asignan prioridades a los atributos, y 3* no se utiliza la función diccionario.

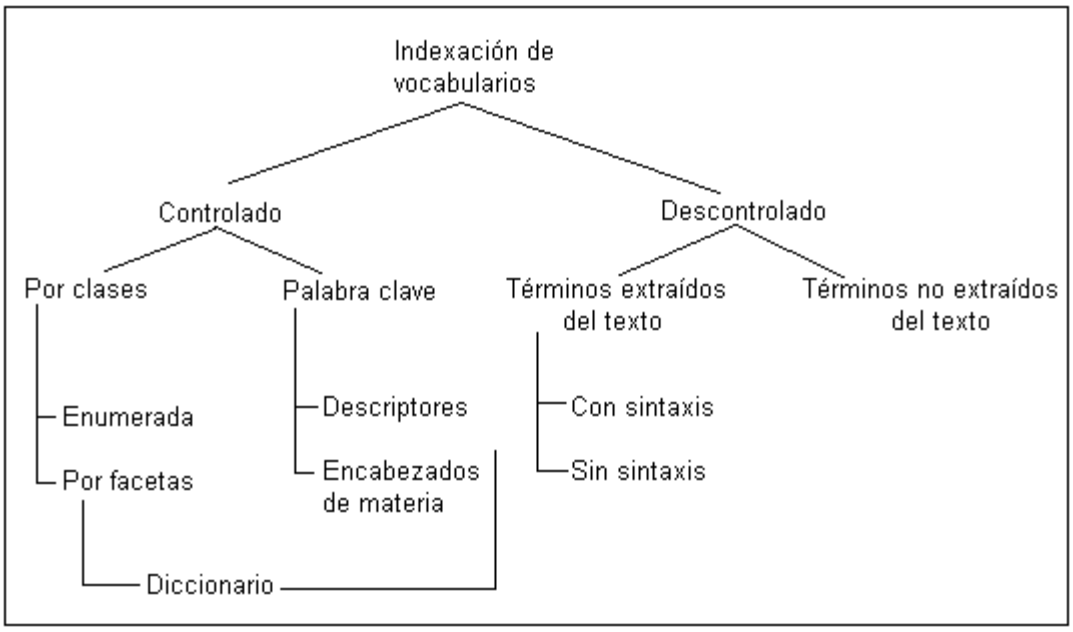



*Figura 4 TAXONOMÍA DE LOS MÉTODOS DE INDEXCIÓN*

### 3.10.2 El entorno de reutilización

A este debe apoyarla un entorno que incluya los siguientes elementos:

- ✓ Una base de datos de componentes capaz de almacenar componentes de software, así como la información de clasificación necesaria para recuperarlos.
- ✓ Un sistema de gestión de bibliotecas que ofrezca acceso a la base de datos.
- ✓ Un sistema de recuperación de componentes de software (por ejemplo, un distribuidor de objetos) que permita que una aplicación cliente recupere componentes y servicios del servidor de la biblioteca.
- ✓ Herramientas de ISBC que apoyen la integración de los componentes reutilizables en un nuevo diseño o implementación.

Todas estas están incorporadas en una biblioteca de reutilización.

La biblioteca de reutilización es un elemento de un depósito de software mayor y proporciona medios para el almacenamiento de componentes de software y una amplia gama de productos de trabajo reutilizables. La biblioteca contiene una base de datos y las herramientas necesarias para consultarla y recuperar componentes de ella.

## 3.11 Economía de la ISBC

Primero requiere entenderle lo que en realidad se puede reutilizar en un contexto de ingeniero del software y luego cuáles son en realidad los costos asociados con la reutilización, será posible desarrollar un análisis costo-beneficio para la reutilización de componentes.

### 3.11.1 Impacto sobre la calidad, la productividad y el costo

Numerosas evidencias mejoran la calidad del producto, la productividad de desarrollo y el costo global.

**3.11.1.1 Calidad:** en un entorno ideal, un componente de software que se desarrolle par reutilizarlo se verificaría como correcto y no contendría defectos. En realidad, la verificación formal no se lleva a cabo de manera rutinaria y existe la posibilidad de que ocurran defectos, y de hecho ocurren. Sin embargo, con cada reutilización los defectos se encuentran y eliminan, y, como resultado, mejora la calidad del componente.

**3.11.1.2 Productividad:** cada los componentes reutilizables se aplican a lo largo del proceso de software, se dedica menos tiempo a la creación de planes, modelos, documentos, código y datos que se requieren para crear un sistema fiable. Por lo tanto, se entrega al cliente el mismo nivel de funcionalidad con menos esfuerzo, lo que mejora la productividad.

**3.11.1.3 Costo:** los ahorros en el costo neto por la reutilización se estiman al proyectar el costo del proyecto si éste fuese desarrollado desde cero, C0, y luego se resta la suma de los costos asociados con la reutilización, Cr, y el costo real del software en el momento de la entrega, Ce.

El factor C0 se puede determinar al aplicar una o más de las técnicas de estimación, Cr, incluyen: análisis y modelo del dominio, desarrollo de arquitectura del dominio, aumento en la documentación, regalías y licencias para componentes adquiridos externamente, creación o adquisición y operación de un depósito de reutilización, y entrenamiento del personal en diseño y construcción para reutilización.

### 3.12 Análisis de costo empleando puntos de estructura

Una nueva aplicación, sistema o producto al definir una arquitectura del dominio y luego dotarla con puntos de estructura. Estos son o componentes reutilizables individuales o paquetes de componentes reutilizables.

Antes de proceder a la reutilización el gestor del proyecto debe comprender los costos asociados con la utilización de los puntos de estructura. Dado que todos los puntos de estructura tiene una historia, es posible recopilar datos de costos de cada uno. En un contexto ideal los costos de calificación, adaptación, integración y mantenimiento asociados con cada componente en una biblioteca de reutilización se mantienen para cada caso de utilización.

Realizando un caso práctico, considere una nueva aplicación, X, que requiere 60 por ciento de código nuevo y la reutilización de tres puntos de estructura: PE1, PE2 y PE3 cada uno de estos componentes reutilizables se ha utilizado en muchas otras aplicaciones, y están disponibles los costos promedio para cualificación, adaptación, integración y mantenimiento.

La estimación del esfuerzo necesario para entregar X requiere determinar lo siguiente:

esfuerzo global = Enuevo + Ecalif + Eadapt + Eint

donde

Enuevo = esfuerzo requerido para diseñar y construir nuevos componentes de software.

Ecalif = esfuerzo requerido para calificar PE1, PE2 y PE3

Eadapt = esfuerzo requerido para adaptar PE1, PE2 y PE3

Eint = esfuerzo requerido para integrar PE1, PE2 y PE3

El esfuerzo requerido para cualificar, adaptar e integrar PE1, PE2 y PE3 se determina al tomar el promedio de los datos históricos recopilados para la cualificación, adaptación e integración de los componentes reutilizables en otras aplicaciones.

Con toda esta explicación podemos decir que los componentes reutilizables deben diseñar en un entorno que establezca estructuras de datos estándar, protocolos de interfaz y arquitecturas de programas para cada dominio de la aplicación.

### 3.13 Fábricas de Software

Las Fábricas de Software son una iniciativa propuesta por Microsoft que plasma la necesidad y provee de los medios para hacer la transición desde el 'hacer a mano' hacia la fabricación o manufactura.

Una fábrica de software es una línea de producto que configura herramientas de desarrollo extensibles con contenido empaquetado y guías, cuidadosamente diseñadas para construir tipos específicos de aplicaciones, es decir es un ambiente de desarrollo configurado para soportar el desarrollo acelerado de un tipo específico de aplicación. Una fábrica de software contiene tres ideas básicas:

- **Un esquema de fabricación**. La analogía de esto es una receta. En ella se listan ingredientes, como proyectos, código fuente, directorios, archivos SQL y archivos de configuración, y explica cómo deberían ser combinados para crear el producto. Especifica qué Lenguajes de Dominio Específico (DSL) pueden ser usados y describe cómo los modelos basados en estos DSLs pueden transformarse en código y otros artefactos, o en otros modelos. Describe la arquitectura de la línea de producto, y las relaciones clave entre componentes y frameworks que la componen.

- **Una plantilla de fábrica de software**. Esta es una gran bolsa de supermercado que contiene los ingredientes listados en la receta. Provee los patrones, guías, plantillas, frameworks, ejemplos, herramientas personalizadas como herramientas para edición visual de DSLs, scripts, XSDs, hojas de estilos, y otros ingredientes para construir el producto.

- **Un ambiente de desarrollo extensible**. Un ambiente como Visual Studio Team System es la cocina en la cual la comida es cocinada. Cuando se configura con una plantilla de fábrica de software, Visual Studio Team System se convierte en una fábrica de software para la familia de productos.

Las fábricas de software son posibles hoy en día y representan el intento de aprender de otras industrias que encaran problemas similares, y aplican patrones específicos de automatización a tareas de desarrollo manual existentes. Las fábricas de software vuelven más rápida, barata y fácil la construcción de aplicaciones, concretando así la visión de la industrialización del software moderno.

### 3.14 Beneficios del Desarrollo de Software basado en Componentes

Según todo lo expuesto anteriormente podemos decir que la ISBC tiene ciertos beneficios como ser:

- **Reutilización del software**. Nos lleva a alcanzar un mayor nivel de reutilización de software.
- **Simplifica las pruebas**. Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- **Simplifica el mantenimiento del sistema**. Cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- **Mayor calidad**. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

De la misma manera, el optar por comprar componentes de terceros en lugar de desarrollarlos, posee otras ventajas:

- **Ciclos de desarrollo más cortos**. La adición de una pieza dada de funcionalidad tomará días en lugar de meses ó años.
- **Mejor ROI**. Usando correctamente esta estrategia, el retorno sobre la inversión puede ser más favorable que desarrollando los componentes uno mismo.
- **Funcionalidad mejorada**. Para usar un componente que contenga una pieza de funcionalidad, solo se necesita entender su naturaleza, más no sus detalles internos. Así, una funcionalidad que sería impráctica de implementar en la empresa, se vuelve ahora completamente asequible.

## CONCLUSION

Podemos decir que la Ingeniería de Software Basada en Componentes (ISBC) nos da los pasos o pautas que el Ingeniero en Sistemas necesita para escoger los mejores componentes para desarrollar un sistema, en el desarrollo de un sistema pueden surgir varias dudas que con la explicación antes dada se las puede aclarar y ayudar a la selección de estos componentes.

Con los modelos damos una opción para que el ingeniero se ayude en la selección de los objetos que desea incluir en su programa, damos una alternativa para que el desarrollador de programas software una vez terminada su aplicación pueda guardar lo desarrollado por él para una nueva aplicación si así lo requiriera. Demostramos los beneficios de utilizar Ingeniería de Software Basada en Componentes, así como los costos que tendría al optar por unir componentes antes que desarrollarlos nuevamente.

# CAPITULO 4: COMPARACION DE ISBC VERSUS COTS

## INTRODUCCION

En los capítulos anteriores hemos mostrado dos técnicas para escoger los mejores componentes dentro de una aplicación software, cada uno tiene diferentes pasos que seguir para la elección de los componentes que mas le convienen al Ingeniero de Sistemas en el desarrollo de un programa software. Se podría decir que en ciertos casos una aplicación podría resultar más fácil que la otra y para otro tipo de programa la técnica que tiene mas pasos le conviene mejor para la elección de los componentes más adecuados.

En la técnica COTS tomamos los componentes que necesitamos y tratamos de implementarlos en nuestro sistema, sino es compatible se lo descarta y se busca otro componente que cumpla con las características que se requiere para la aplicación que estamos desarrollando.

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.

Por otro lado la técnica ISBC nos da unos pasos que el Ingeniero en Sistemas debe seguir para la selección del componente mas adecuado de acuerdo a lo que él este desarrollando, tratando de identificar, construir, catalogar y diseminar un conjunto de componentes de software que sean aplicables para el software que se este construyendo.

ISBC a diferencia de COTS estudia la compatibilidad, la interoperabilidad o la trazabilidad de los componentes durante el desarrollo del sistema.

| Ventajas de COTS | Ventajas de ISBC |
|---|---|
| Facilitar la compartición de productos del ciclo de vida. | Tiene herramientas que ayudan con la búsqueda del mejor componente |
| La “tolerancia a fallos” es una típica característica que va a depender de la arquitectura de la aplicación es decir de la metodología COTS, mientras que la “madurez” es más propia de los componentes. | Distribución de componentes (CORBA, Servicios Web |
| | Interoperabilidad (CORBA, Servicios Web). |
| COTS va directamente a la búsqueda y selección de componentes | ISBC se define el dominio, estructura y arquitectura |

***Tabla 3 VENTAJAS COTS Y VENTAJAS DE ISBC***

Como podemos ver tanto las ventajas de COTS como de ISBC son similares ya que los dos métodos tienen la misma finalidad (el de reducirnos tiempo de trabajo y esfuerzo en la realización del proyecto), pero en el momento de analizar más a fondo encontramos que la metodología COTS como dice la última ventaja va directamente a la búsqueda y selección de componentes mientras que la metodología ISBC realiza otros pasos antes de seleccionar los componentes primero define el dominio, la estructura y arquitectura de los componentes. Así mismo la metodología ISBC tiene herramientas que nos ayudan elegir dichos componentes como CORBA, COM y otras, pero en COTS tenemos que este tiene una característica típica de su metodología que es la tolerancia a fallos, mientras que los componentes presentan la madurez dentro de su estructura.

| Desventajas de COTS | Desventajas de ISBC |
|---|---|
| Necesidad de invertir antes de obtener resultados | Problemas de cualificación (sin errores) |
| Carencia de métodos adecuados | Descripción inadecuada |
| Necesidad de formar al personal | Sistema de clasificación pobre |
| Convencer a los “managers” | Problemas de integración / adaptación |
| Dificultad para institucionalizar los procesos | Falta de estandarización |

***Tabla 4 DESVENTAJAS COTS Y DESVENTAJAS DE ISBC***

En lo que respecta a las desventajas la metodología COTS puede tener ciertos inconvenientes con respecto a ISBC ya que antes de escoger los componentes ISBC tiene diferentes métodos para evaluar el uso de los componentes deseados y como se puede ver en el cuarto punto muchas de las veces es complicado convencer a los jefes de optar por este método.

A nuestro criterio y en el momento de la práctica, la técnica COTS resulta ser más fácil de implementar ya que la técnica ISBC como ya se dijo anteriormente toma los componentes y los lleva a la práctica, no así la ISBC que toma diferentes componentes que cumplen la misma función y los analiza para ver cual de ellos puede funcionar de la mejor manera en la nueva aplicación que se esta desarrollando, pero aun así no se garantiza que los componentes funcionen adecuadamente retrazando el tiempo de desarrollo del software por estar escogiendo al mejor de todos ellos, porque en la ISBC se tiene una biblioteca en donde se almacenan los componentes y podemos tener 100 o más componentes que realicen la misma función y hasta escoger el mejor gastamos tiempo de análisis o de desarrollo del mismo.

## CONCLUSION

Como ya hemos visto tanto la técnica ISBC como COTS nos ayudan a reutilizar componentes como ser: código fuente, o cualquier tipo de assets, pero los dos tienen diferentes maneras de llevar a cabo la selección mas optima de los componentes que nos ayudaran a desarrollar una aplicación, con la comparación dada entre las dos técnicas mostrando las ventajas y desventajas que se puede encontrar al optar por una de las dos, el usuario puede escoger cualquiera que se acople según sus necesidades y según cual sea la técnica que le de un soporte más amplio de los componentes que requiere.

# CAPITULO 5: PLANTEAMIENTO Y DESARROLLO DEL CASO PRÁCTICO

## INTRODUCCION

En los capítulos anteriores se ha desarrollado de manera teórica el proceso de selección de los componentes tanto con la aplicación de COTS como ISBC, pero para entender mejor la teoría antes expuesta vamos a demostrar con un ejemplo práctico la forma de selección de estos elementos usando una de las técnicas antes expuestas, usaremos distintos componentes ya previamente programados y los ensamblaremos en el caso práctico.

### 5.1 Método de aplicación

El caso práctico de nuestra investigación es un sistema de notas para los alumnos del centro educativo EDEN, el método que hemos decidido utilizar es COTS, porque a nuestro criterio lo vemos mas fácil de utilizar y de escoger los componentes, así como se lo explico en el capítulo 4 de Comparación de ISBC contra COST.

### 5.2 Planteamiento de la aplicación

Para nuestro sistema utilizamos la aplicación A2TManager que maneja las aplicaciones: Apache, MySQL y PHP.

### 5.3 Selección de la aplicación

Para el desarrollo del caso práctico elegimos un programa que nos permita utilizar el servidor Apache, la base de datos MySql y Php, de tal manera que nos facilite el manejo y desarrollo del programa, esta parte de la aplicación fue lo que vimos mas necesario de manejar ya que estas herramientas son más fáciles de controlar funcionando juntas que por separado, para este caso utilizamos varias herramientas hasta llegar a Apache Triad que era el que más satisfacía nuestras necesidades.

### 5.4 Adaptación de la aplicación

Para el uso de la aplicación probamos 3 programas diferentes, tuvimos una serie de inconvenientes, la primera aplicación que probamos no era compatible con el sistema operativo Windows Vista, la segunda aplicación era compatible, pero tuvimos el problema con la base de datos de MySQL, ya que al momento de crear la base de datos no se lograba conexión, este

problema nos retraso un poco, se intento arreglar el problema pero no tuvimos éxito, por ultimo intentamos con Apache Triad que es el programa que hemos utilizado para el caso práctico.


***Figura5 pantalla de aplicación de los servicios.***

## 5.5 Integración de los componentes al sistema

Antes de comenzar la aplicación necesitamos levantar el servicio de Apache llamado Apache2Triad PostgreSQL Service.

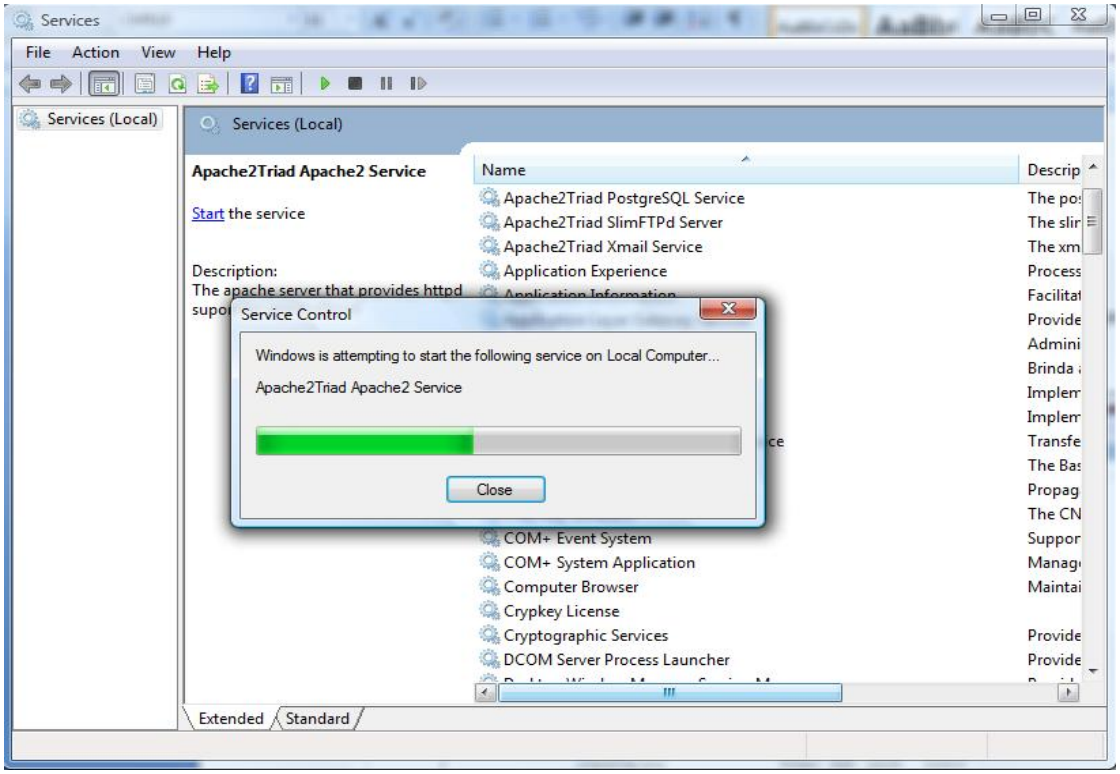



***Figura6 levantamiento del servicio Apache***

Para crear y administrar las bases de datos de una manera más sencilla utilizamos PhpMyAdmin, pero para empezar a utilizarlo necesitamos el password y el usuario, el mismo que será utilizado para la conexión con la base de datos, tenemos que pasar por la pantalla de confirmación de claves antes de ingresar al navegador.

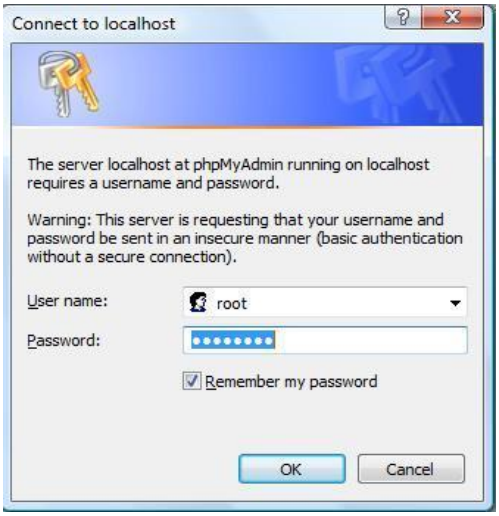



***Figura7 pantalla de confirmación de claves***

Una vez que ingresamos el usuario y la contraseña podemos hacer uso del navegador y usar phpMyAdmin, donde podemos administrar la base de datos con la cual trabajaremos.

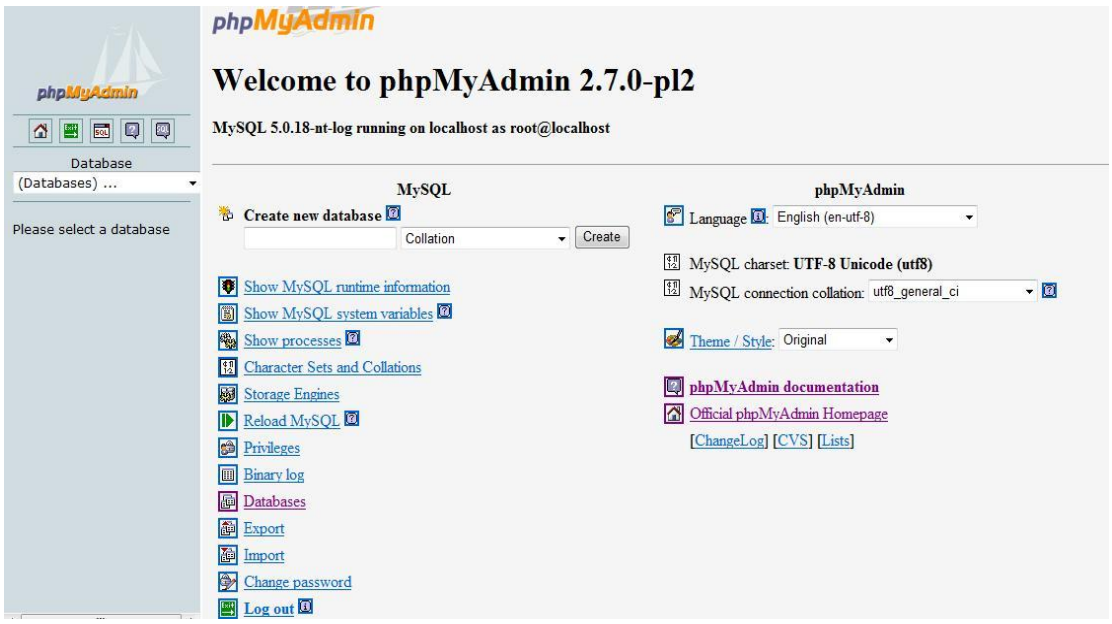



***Figura 8 Pantalla principal del administrador de la aplicación***

Al ingresar en las bases de datos podemos crear una nueva base de datos que para nuestro caso práctico se llamara “notas”.

Luego creamos las tablas que necesitamos y vamos a utilizar para el desarrollo de la aplicación, las mismas que son: materias, notas, alumnos y profesores, cuyas estructuras son las siguientes:

La estructura de la tabla de alumnos es:

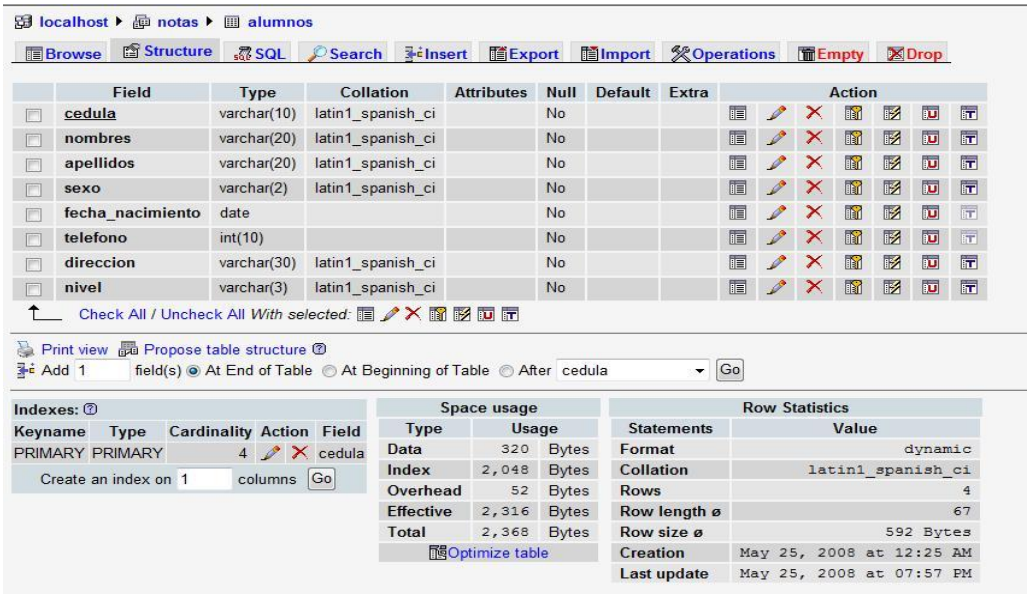



***Figura9 pantalla de la estructura de la tabla de alumnos***

La estructura de la tabla de profesores es:

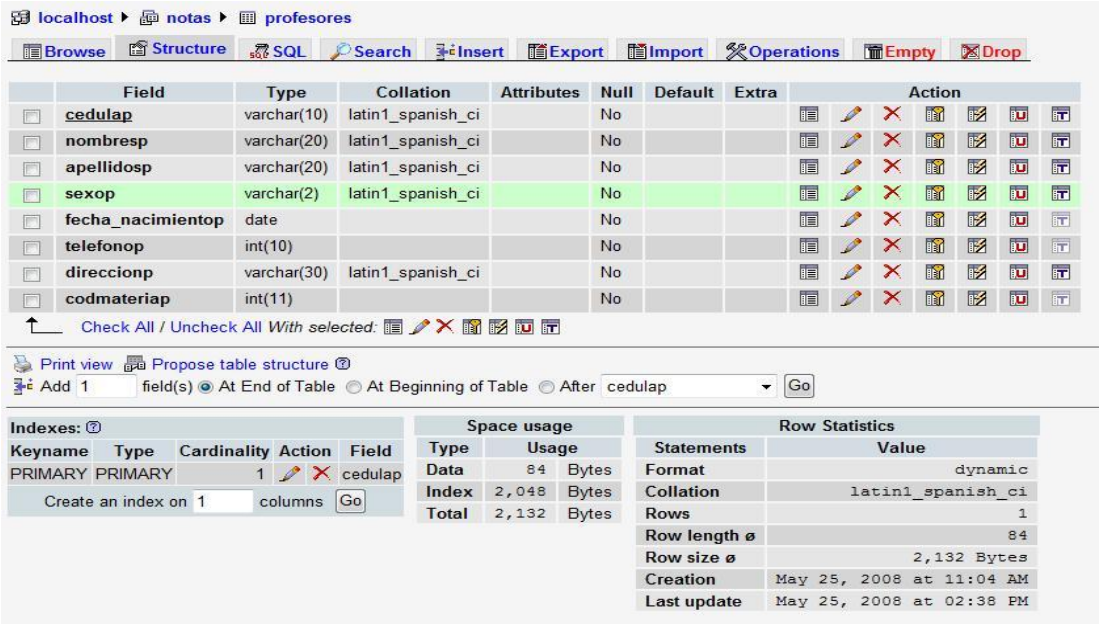



***Figura10 pantalla de la estructura de la tabla de profesores***

La estructura de la tabla de materias es:

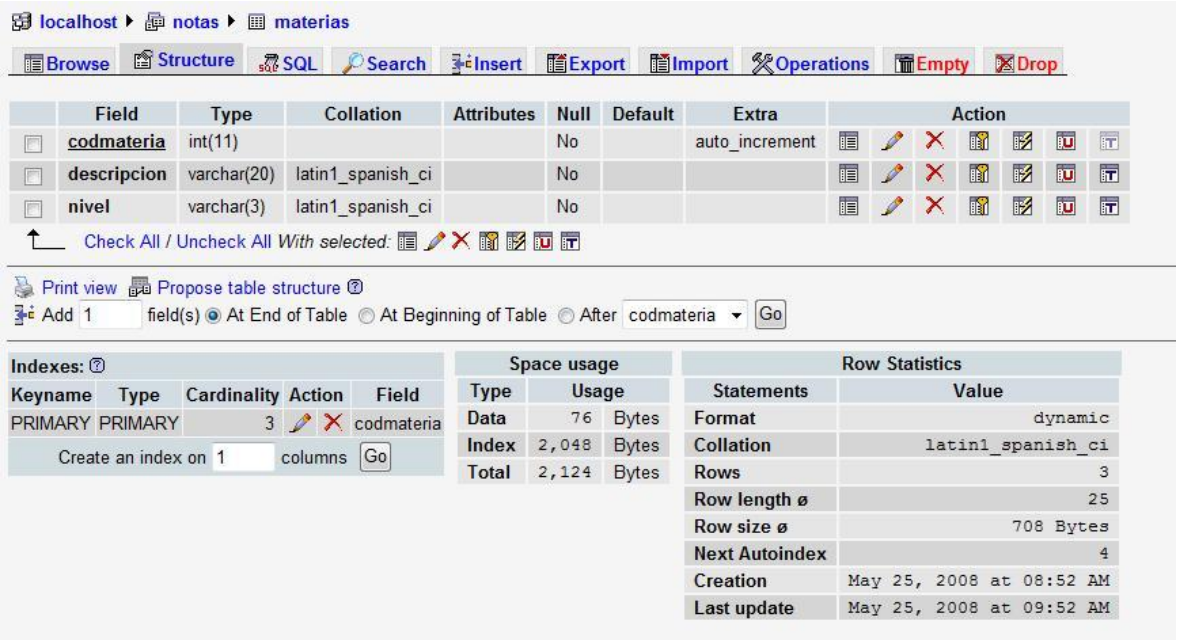



***Figura11 pantalla de la estructura de la tabla de materias***

La estructura de la tabla de notas es:

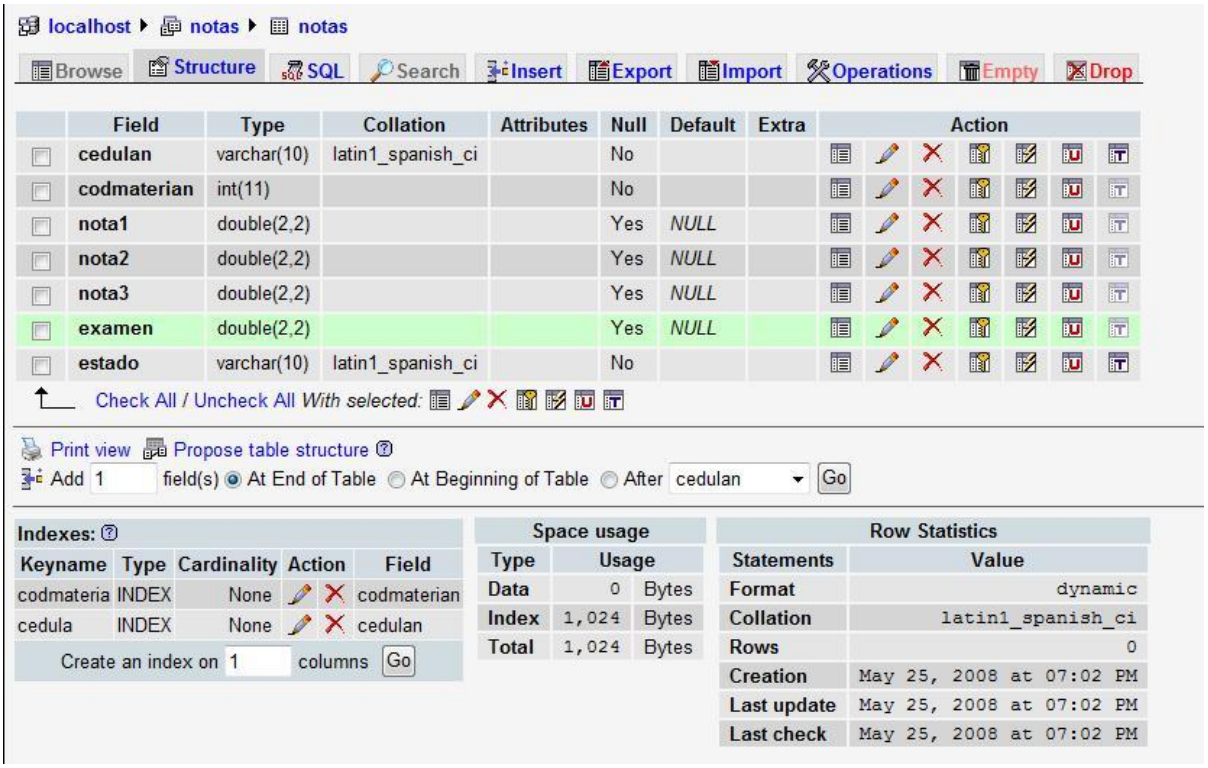



***Figura12 pantalla de la estructura de la tabla de notas***

### 5.6 Selección de los componentes

Para este proyecto seleccionamos unos componentes que cumplen con ciertos requerimientos que el programa y el cliente deseaban, que los encontramos ya desarrollados y los implementamos al proyecto.

Entre los componentes seleccionados tenemos:

#### 5.6.1 Selección de un componente Flash

Este es un flash que es la animación de las pantallas, este componente es utilizado tal cual es su programación, sin cambios en el código fuente más que el nombre del centro educativo al cual esta siendo aplicado este proyecto. El método que usamos es el abierto.

Este componente fue evaluado de manera sencilla, primeramente lo evaluamos visualmente, ya que queríamos que vaya acorde al tema y con colores no tan llamativos, además por su adaptación a nuestra aplicación.

Para su selección lo comparamos con otros Flash, pero al momento de implementarlo tomamos mucho en cuenta que al ser para un centro educativo tenía que ser un poco más llamativo pero al mismo tiempo no tenía que ser con colores demasiado exagerados, por lo cual decidimos seleccionar el que está en la imagen siguiente:


***Figura13 componente flash del proyecto***

#### 5.6.2 Selección del Menú

Uno de los componentes que el programa requería es un menú, el mismo que lo hemos buscado en el Internet, el componente se utilizo sin realizar muchos cambios en la programación haciéndonos mucho más fácil y rápido para el usuario, este elemento fue usado en su esencia como se indico sin cambios mas que los nombres principales y en el submenú se como se lo observa en el siguiente gráfico: (método usado abierto).

Para la selección de este componente, evaluamos primeramente su compatibilidad con el sistema operativo, además con el lenguaje en el que estamos programando, también lo elegimos por su diseño y su forma de adaptación, lo que hicimos también fue hacer cambios de colores para al final llegar a la selección del menú que implementaremos en la aplicación el mismo que es la 4ta opción.

Lo comparamos con otros menús que estaban desarrollados en el mismo lenguaje, pero la elección fue por la programación, su diseño y su posición, en este menú solamente teníamos que cambiar los nombres que se iban a mostrar y las rutas de las paginas a las cuales se iba a hacer el link, además nos su forma vertical, su color y su diseño.


*Figura14 menú opcion1*


*Figura15 menú opción 2*


*Figura16 menú opción 3*


*Figura17 menú opción 4*

### 5.6.3 Selección de la Validación de Cedula

Se lo evaluó según los requerimientos de nuestra aplicación, la validación de cedula tenía que estar sin errores, además según los números de cedulas existentes en nuestro país, para los otros campos elegimos una validación más sencilla en la que separamos el numero de la cedula en dos campos, los 9 primeros en un campo y el ultimo en un solo campo, las otras validaciones que implementamos simplemente permiten dar a conocer al usuario que no puede pasar a otra pantalla sin llenar todos los campos. Método usado abierto.

La elección que hicimos fue evaluando 2 validaciones, el la una se evaluaban los componentes por separado como se indico anteriormente, y en la segunda en un solo campo, el problema suscito cuando al momento de implementar esta validación nos daba algunos errores, ya que no siempre valido correctamente el campo, al principio también tuvimos algunas complicaciones ya que declaramos un campo entero de 9 dígitos, pero al momento de declararlo como entero los números que comenzaban con 0 no eran validados ya que no se tomaba en cuenta el número 9, por lo cual luego tuvimos que declararlo como VARCHAR, luego de eso pudimos implementar sin ningún problema la validación.

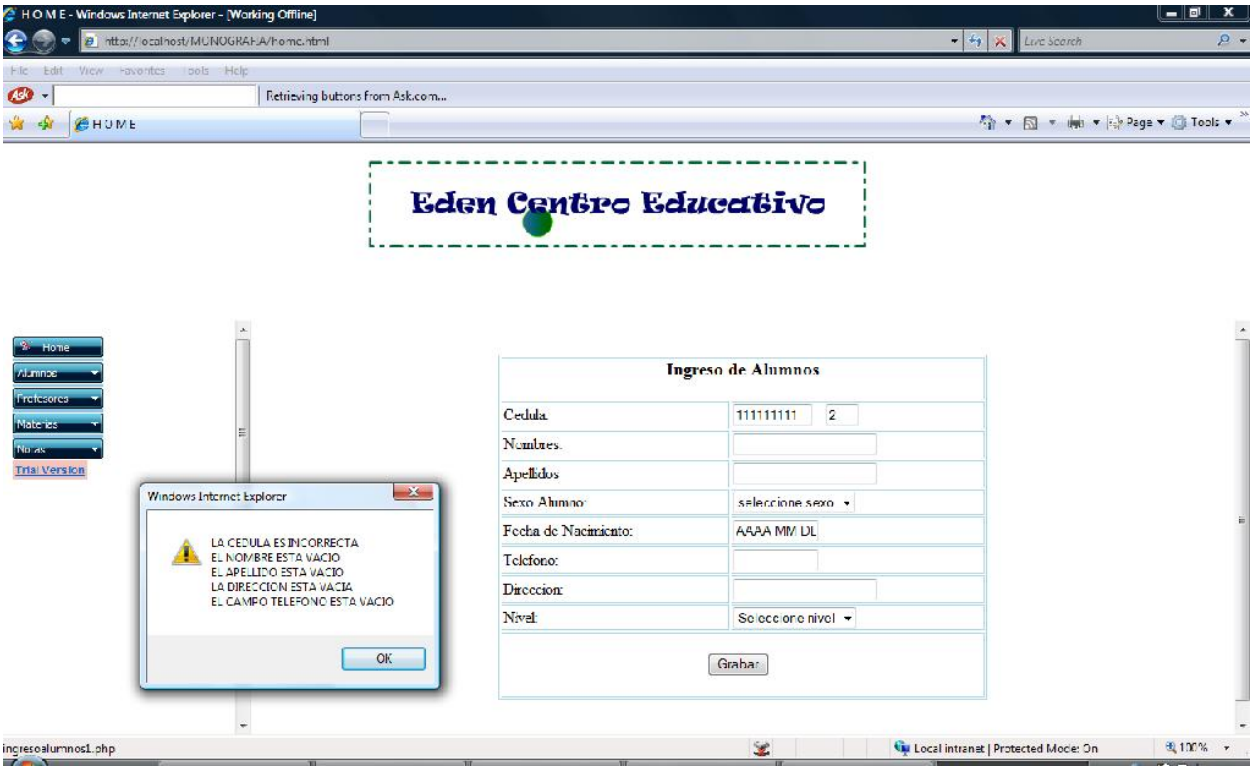



***Figura18 pantalla de ingreso de alumnos con las validaciones***

### 5.6.4 Selección de Componente para realizar reportes

Como último componente tenemos los reportes, para el uso de este componente elegimos la forma más fácil para el usuario, que le permita leer de una forma más clara y sea sencillo de hacerlo, por lo cual al elegir la opción de reporte solo va a salir una pantalla que va a indicar si quiere solamente abrirlo o si desea guardar el reporte, de esa manera no le complicamos al usuario y además fue una manera sencilla de poder adaptar este componente, este componente se implementan tal cual se encuentra programado, se utiliza unas librerías a las cuales no se les realiza cambio alguno solamente se las llama. El método que utilizamos es el abierto

Para la elección lo comparamos con otras maneras de realizar el reporte, intentamos hacerlo enviándolo a un pdf, pero se nos hizo un poco más complicado, ya al implementarlo no funcionaba, por lo cual intentamos hacerlo enviándolo a un archivo .doc, este fue más fácil su adaptación e implementación, uno de los problemas que encontramos fue que al principio al momento de enviar el reporte no podíamos visualizar los valores guardados en la base de datos, el problema se encontraba en el modo de mandar a escribir en el archivo, se hizo la búsqueda normal en la base de datos, pero al momento de mandarlo al archivo de texto utilizábamos la sentencia OUTPUT en lugar de ECHO que utilizamos para la impresión en pantalla, luego de haber solucionado este problema pudimos desarrollar tranquilamente los reporte.

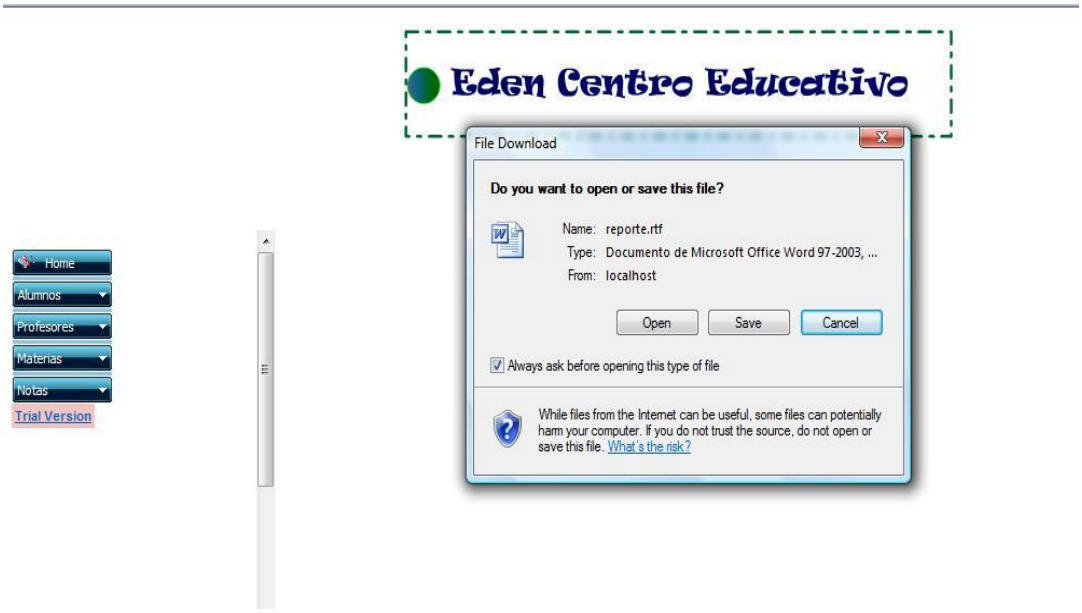



***Figura19 pantalla del reporte. doc formato como se guardan los reportes***

## CONCLUSION

Con la aplicación de este caso práctico hemos dado a conocer el uso de la técnica COTS, usando ciertos componentes y ensamblándoles, demostrando que el uso de componentes ya programados nos facilita la realización de un sistema cualquiera.

## CONCLUSIONES FINALES

Como hemos visto en toda esta investigación el uso de el método sea ISBC o COTS como se lo hizo en este proyecto nos simplifica muchas de las tareas que en tiempos anteriores nos hubieran ocupado mas tiempo del que un proyecto requiere, con todo este tiempo que nos ahorramos implementando componentes ya funcionando y a los cuales se les realiza cambios pequeños se puede realizar tareas como de prueba o de cambios generales que en otros momentos nos tomaría mas del tiempo prometido al cliente.

Con esto decimos que la Ingeniería de Software basada en componentes es una herramienta que le ayuda demasiado al ingeniero en sistemas y es una metodología que esta en auge por el tiempo que nos ahorra.

Esperamos que con este trabajo les sirva para ampliar sus conocimientos y eliminar ciertas dudas sobre este tema.

Gracias.

## GLOSARIO

| | |
|---|---|
| **COTS** | Commercial-Off-The-Shelf |
| **ISBC** | ingeniería de software basada en componentes |
| **ORB** | Object Request Broker |
| **PORE** | Procurement-Oriented Requirement Engineering |
| **ACRE** | Acquisition of Requirements |
| **OTSO** | Off-The-Shelf Option |
| **FODA** | Feature-Oriented Domain análisis |
| **GUI** | Graphic User Interface |
| **DBC** | Desarrollo Basado en Componentes |
| **CORBA** | Common Object Request Broker Architecture |
| **OMG** | Object Management Group |
| **ORB** | Object Common Broker |
| **IDL** | Interface Definition Language |
| **GIOP** | General Inter-ORB Protocol |
| **IIOP** | Internet Inter-ORB Protocol |
| **DII** | Dynamic Invocation Interface |
| **DSI** | Dynamic Skeleton Interface |
| **CDL** | Componentes Comerciales en Línea |
| **DSL** | Lenguajes de Dominio Específico |

## BIBLIOGRAFIA


http://www.escet.urjc.es/~aisw/docs/isiiisbc.pdf

http://www.lcc.uma.es/~canal/papers/tesis/canal_tesis.pdf

http://wiki.squeak.org/squeak/2519

http://www.ramonmillan.com/tutorialeshtml/corba.htm

http://www.programatium.com/manuales/corba/9.htm

http://www.tutorial-enlace.net/

http://dis.um.es/~lopezquesada/documentos/Tema%2012.ppt#285,9,

canal_tesis PENDIENTE.pdf

01CD_DBC_0708_T1_1_IntroDBC1.pdf

ISBC modelos.pdf

memoria-completa.pdf

Trabajos de compañeros de ciclos anteriores.

INGENIERIA DE SOFTWARE BASADA EN COMPONEN de Xavier Bermeo, Santiago Capelo y Jonny Delgado.

IngenieriaSoftwareBasadoComponentes-ok de Xavier Auquilla, Carlos Sampedro y Adrián Paguay.

TRABAJO de Johanna Arias, Anita Moncayo y Tatiana Vasquez.

# D I S E Ñ O  D E  M O N O G R A F I A

## 1. Titulo del Proyecto

INVESTIGACION DEL PROCESO DE INGENIERÍA DE SOFTWARE BASADO EN COMPONENTES (ISBC) Y SU APLICACIÓN A UN CASO PRÁCTICO.

## 2. Selección y Delimitación del Tema

***Contenido*:** Esta monografía se la ejecutará dentro del área de Ingeniería de Software, la misma se refiere a una exploración de los componentes funcionales, comportamiento y datos para la reutilización. Tomando en cuenta los diferentes aspectos que supone el uso de los componentes como:

- Los requisitos del cliente
- Selección de la arquitectura
- Selección de los componentes para la reutilización
- Calificar los componentes con relación a la arquitectura y funcionalidad definida (que aspectos deben tomar en cuentas para calificar o valorar los mismos)
- Adaptar los componentes (para integrarlos
- Integrar los componentes
- Pruebas.

Al ser explorados estos aspectos que un componente debe cumplir, se garantiza que el componente realizará la función requerida, suministrará la información útil acerca de la operación, con lo cual podremos tener un soporte de diferentes sistemas operativos, diversos lenguajes y alcanzar una gran cantidad de servicios.

***Tiempo*:** La monografía nos llevará un plazo no mayor a 9 semanas.

## 3. Descripción del Objetivo de Estudio

Cada vez las nuevas tendencias en desarrollo de software es simplificar las tareas que se realizan en el afán de llegar a completar en el menor tiempo posible el adelanto de un proyecto, con lo cual los Ingenieros en Sistemas optan por el uso de ciertos componentes en sus proyectos, tal es el caso de las páginas Web en las cuales se unen diversos elementos encontrados como ser menús, diversas validaciones, animaciones, etc, los cuales funcionan por si solos y pueden ser reutilizados en diversos casos.

Por lo que hemos decidido realizar un estudio de la ISBC (Ingeniería de Software Basada en Componentes) que supone la utilización de "partes software" bien especificadas, con interfaces descritas como contratos, con arquitecturas de software y estándares para hacer el diseño genérico de una aplicación para ingreso de notas dirigida a un centro educativo.

## 4. Introducción

En este momento la necesidad de desarrollar software con mucha más rapidez y con un óptimo funcionamiento que cumpla con los requerimientos del cliente, ha llevado a la Ingeniería de Software Basada en Componentes a establecer reglas o pasos que se debe seguir para elegir el componente más adecuado dependiendo de la aplicación que estemos realizando.

Con la ayuda del lenguaje JavaScript, Php, Html y la base de datos MySql, vamos a demostrar el uso de estos componentes dentro de los pasos que nos da la Ingeniería de Software Basada en Componentes, la misma que tiene diferentes etapas en las que podemos identificar al componente que mas nos conviene en el proyecto que estemos realizando, tal es el caso de: *la etapa de búsqueda del mejor componente* que identifica las propiedades (como ser los servicios que proporciona) o el uso de estándares, la calidad que son difíciles de aislar y aspectos no técnicos como la cuota de mercado de un vendedor o el grado de madurez del componente dentro de la organización.

## 5. Situación Actual y Futura

***Situación Actual:*** Hoy en día los desarrolladores de software están optando por un método que les permita realizar software en el menor tiempo posible, pero que al mismo tiempo cumpla con las exigencias del cliente, por este motivo la Ingeniería de Software Basada en Componentes esta

siendo explotada en su estudio para poder llegar a comprender mejor como unir estos diversos componentes encontrados y cuales son los mejores para satisfacer las necesidades planteadas entre cliente y desarrollador.

En nuestro medio cada día se realizan nuevas herramientas que ayuden al ingeniero a realizar de una manera más rápida su proyecto ahorrando muchos pasos con los que nos permite utilizar ese tiempo para plasmar otras tareas, por lo cual, la Ingeniería de Software Basada en Componentes esta cumpliendo un rol muy importante en el desarrollo de diversas aplicaciones.

***Situación Futura:*** Una vez terminada esta monografía se desea dar a conocer a los desarrolladores el uso de los componentes a través de la investigación detallada de las propuestas que nos da la Ingeniería de Software Basada en Componentes, dar una visión diferente a los Ingenieros en Sistemas con respecto al tema de desarrollo de software reutilizando los mejores componentes que se encuentran en otros repositorios, que cumplan con los requerimientos del cliente, además se desea demostrar este concepto mediante la implementación de un caso práctico que estará orientado a un centro educativo, este sistema será utilizado para el ingreso de notas de los alumnos.

# 6. Objetivos

## 6.1 Objetivo General

Demostrar la Ingeniería de Software Basado en Componentes mediante el desarrollo de un sistema para la gestión de notas para un Centro Educativo.

## 6.2 Objetivos Específicos

- Investigar el proceso ISBC de Ingeniería de Software.
- Plantear un proyecto ISBC.
- Desarrollar un caso práctico de ISBC

## 7. Componentes

Los componentes que vamos a utilizar para el caso práctico son:

Un menú en lenguaje JavaScript.

Función para validar:

- cedula
- fecha

Manejo de Base de Datos:

- MySql

Manejo de Dreamweaver o Delphi y Php

## 8. Procedimientos Metodológicos

- Investigación de la Ingeniería de Software Basada en Componentes.
- Desarrollo de la investigación.
- Desarrollo de productos: Caso Práctico
- Pruebas y corrección de errores (retroalimentación)

## 9. Contenidos

### 9.1 Marco Teórico

La Ingeniería de Software basada en componentes es un proceso que se centra en el diseño y construcción de sistemas basados en computadora que utilizan "componentes" de software reutilizables.

ISBC está basado en el ensamblaje e integración de componentes software ya existentes, compuesto por diferentes etapas.

Sistemas informáticos complejos y de alta calidad deben ser desarrollados en periodos de tiempo muy cortos.

El desarrollo de software basado en componentes se ha convertido en uno de los mecanismos más efectivos para la construcción de grandes sistemas y aplicaciones de software.

Cuando la mayor parte de los aspectos funcionales de esta disciplina comienzan a estar bien definidos, la atención de la comunidad científica comienza a centrarse en los aspectos extra funcionales y de calidad, como un paso hacia una verdadera ingeniería. En este trabajo se dan a conocer los aspectos de calidad relativos a los componentes software y a las aplicaciones que con ellos se construyen, con especial énfasis en los estándares internacionales que los definen y regulan, y en los problemas que se plantean en este tipo de entornos.

La creciente necesidad de realizar sistemas complejos en cortos periodos de tiempo, a la vez que con menores esfuerzos tanto humanos como económicos, está favoreciendo el avance de lo que se conoce como Desarrollo de Software Basado en Componentes (DSBC).

Esta nueva disciplina se apoya en componentes software ya desarrollados, que son combinados adecuadamente para satisfacer los requisitos del sistema.

Desarrollar una aplicación se hace más sencillo, ya que se puede realizar las búsquedas necesarias y reutilizar el código fuente que cumpla con las necesidades.

En este sentido, la tecnología de componentes ofrece ya soluciones robustas para muchos de los problemas que se plantean en la construcción de grandes sistemas de software. Los principales esfuerzos de la comunidad de software en estos temas se han basado hasta ahora en los aspectos funcionales de los componentes, es decir, en la funcionalidad que ofrecen. Sin embargo, por lo general se han venido obviando muchos de los aspectos de calidad que intervienen a la hora de seleccionar componentes y ensamblarlos para construir aplicaciones que satisfagan los requisitos del cliente, estos requisitos pueden afectar a varias partes del sistema, y por ello tendrán prioridad si entran en conflicto con los requisitos funcionales. Además, el cuidadoso análisis de los atributos de calidad puede mejorar el proceso de discriminación de los componentes candidatos que cumplan el núcleo de requisitos funcionales.

La reutilización de componentes de software es un proceso inspirado en la manera en que se producen y ensamblan componentes en la ingeniería de sistemas físicos. La aplicación de este concepto al desarrollo de software no es nueva. Las librerías de subrutinas

especializadas, comúnmente utilizadas desde la década de los setenta, representan no de los primeros intentos por reutilizar software.

Existen variadas definiciones del término Reutilización de Software. Algunas de estas definiciones son las siguientes:

• Según Somerville [1], la reutilización es un enfoque de desarrollo [de software] que trata de maximizar el uso recurrente de componentes de software existentes.

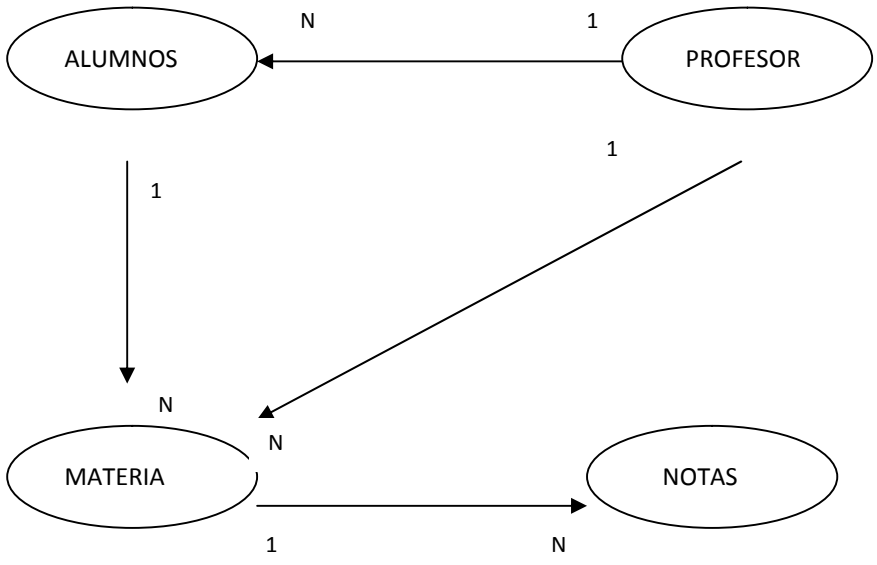
• Según Sametinger [2], la reutilización de software es el proceso de crear sistemas de software a partir de software existente, en lugar de desarrollarlo desde el comienzo.

• Según Sodhi et al. [3], la reutilización de software es el proceso de implementar o actualizar sistemas de software usando activos de software existentes.

Objetivos:

- Desarrollar sistemas a partir de componentes ya construidos
- Desarrollar componentes como entidades reusables
- Mantener y evolucionar el sistema a partir de la adaptación y reemplazo de sus componentes.

## 9.2 Diseño



### 9.3 Implementación

Este punto será demostrado al finalizar la monografía, hemos planteado un caso práctico para implementarlo en un centro educativo, el mismo que será utilizado para el ingreso de notas de alumnos.

## 10. Recursos Humanos y Técnicos

***Recursos Humanos:***

El proyecto se lo realizará con la participación de:

Investigadores y Desarrolladores: Mariela Farfán y Ruth Rojas

Asesores del Proyecto: Ing. Pablo Pintado

***Recursos Materiales:***

Hardware: El hardware a utilizar es el siguiente:

2 computadoras Intel Pentium 4

CPU 2.66 GHz, 512 MB de RAM

Software: Dentro del software a utilizar estan:

Sistema Operativo Windows XP

Base de Datos MySql Administrador

Editor Dreamweaver

Lenguaje JavaScript y Php

## 11. Cronograma de actividades

Para una correcta realización de la presente monografía se seguirá el orden del siguiente cronograma

El tiempo viene dado en semanas.

| **ACTIVIDADES** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|
| 1. Investigación sobre ISBC | | | | | | | |
| 2. Desarrollar de caso practico | | | | | | | |
| 3. Probar y corregir errores | | | | | | | |
| 4 Implementar el caso practico | | | | | | | |

## 12.Conclusiones

La ingeniería de software basado en componentes se convirtió en el pilar de la Revolución Industrial del Software y se proyecta hoy en día en diversas nuevas formas de hacer software de calidad con los costos más bajos del mercado y en tiempos que antes eran impensables.

Con esto nos podemos dar cuenta que las ventajas son mas grandes que las desventajas que puede tener la reutilización de componentes obteniendo beneficios como mayor disponibilidad o retorno de inversión inmediata, mejora el proceso de negocio, nos evita costos de desarrollo y mantenimiento, se pueden realizar nuevas versiones continuamente y la independencia entre el Hardware y Software.

## 13. Recomendaciones

La ISBC nos propone diversas pautas que se debe seguir para tener un software de calidad en menor tiempo con costos menores en cuestión de desarrollo y mantenimiento pero debemos darnos cuenta que para que esto suceda el Ingeniero debe tomar en cuenta todos los procesos que supone la selección de los componentes más adecuados para la aplicación que desee desarrollar, que todos aquellos elementos que se requieran utilizar se acoplen adecuadamente, que cumplan con los requerimientos del cliente y que en el momento de ser calificados su funcionalidad y arquitectura cumpla con los aspectos que la aplicación requieren.

## 14. Bibliografía


- **www.google.com.ec**
- **http://xherrera334.blogspot.es/1193625420/**
- **www.altavista.com**
- **Libro Ingeniería del Software**

UNIVERSIDAD DEL AZUAY

Cuenca, 27 de Febrero del 2008

Señor Eco.
Luis Mario Cabrera
Decano de la Facultad de Ciencias de la Administración

Señor Decano:

Por medio de la presente, Mariela Farfán y Ruth Rojas egresadas de la escuela de Ingeniería de Sistemas, solicitamos la aprobación de la monografía cuyo tema es **INVESTIGACION DEL PROCESO DE INGENIERÍA DE SOFTWARE BASADO EN COMPONENTES (ISBC) Y SU APLICACIÓN A UN CASO PRÁCTICO.**

Seguras de contar con la favorable acogida a la presente, nos despedimos de usted,

Atentamente,

Mariela Farfán

Ruth Rojas

CERTIFICO .Que, El H. Consejo de Facultad en sesión del 3 de Marzo del 2008 conocio el informe del señor profesor de la Junta Academica de Ingeniero de Sistemas de las señoritas Mariela Farfan y Ruth Rojas y en base a esta , aprobo la denuncia de la Monografia con el tema INVESTIGACION DEL PROCESO DE INGENIERIA DE SOFTWARE BASADO EN COMPONENTES (ISBO) Y SU APLICACIÓN A UN CASO PRACTICO y se ratifica como Director al Ingeniero Pablo Pintado Zumba y como Miembro del Tribunal al Ingeniero Lennin Eraso el denunciante tienen un plazo maximo de cuatro meses contados a partir de la fecha de aprobación es decir hasta el 3 de Junio del 2008

Cuenca 4 de Marzo del 2008

UNIVERSIDAD DEL AZUAY
FACULTAD DE ADMINISTRACION
SECRETARIA

UNIVERSIDAD DEL AZUAY

Cuenca, 27 de Febrero del 2008

Señor Eco.
Luis Mario Cabrera
Decano de la Facultad de Ciencias de la Administración

Señor Decano:

Por medio de la presente, me permito informar que he procedido a revisar el diseño de monografía de los estudiantes Mariela Farfán y Ruth Rojas, egresados de la escuela de Ingeniería de Sistemas cuyo tema es **INVESTIGACION DEL PROCESO DE INGENIERÍA DE SOFTWARE BASADO EN COMPONENTES (ISBC) Y SU APLICACIÓN A UN CASO PRÁCTICO**, el mismo que cumple con los requisitos metodológicos y técnicos requeridos.

Por las consideraciones anotadas me permito, salvo mejor criterio, recomendar su aprobación

Atentamente,

Ing. Pablo Pintado