UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 7

GUÍA PARA LA REALIZACIÓN DEL ANÁLISIS Y DISEÑO DE APLICACIONES CON ARQUITECTURA BASADA EN COMPONENTES.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autores: Lucelia Jiménez Pandiella

Andro Ernesto Viera Ojeda

Tutores: Ing. Rosalía Cué Delgado

Ing. Yovannys Sánchez Corales

Asesor: Dr. Denis Derivet Thaureaux

Ciudad de La Habana, Junio de 2007

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo en el cual hemos utilizado información y documentación que es propiedad de la empresa SOFTEL lo cual está sujeto a un acuerdo de confidencialidad. Ponemos a disposición de la Universidad de las Ciencias Informáticas (UCI) todo aquello que no comprometa dicho acuerdo.

Para que así conste firmamos la presente a los 28 días del mes de junio del año 2007.

Firma autora: Lucelia Jiménez Pandiella	Firma tutora: Ing. Rosalía Cué Delgado
Firma autor: Andro Ernesto Viera Ojeda	Firma tutor: Ing. Yovannys Sánchez Corales

"El hombre es el verdadero creador de su destino.
Cuando no está convencido de ello, no es nada en la vida."
Gustave Le Bon

DATOS DE CONTACTO

Ing. Rosalía L. Cué Delgado:
Graduada de Ingeniería en Sistemas Automatizados en el año 1995. Profesora Auxiliar Adjunto Facultad No. 7, en la disciplina de Ingeniería de Software. Posee 11 años de experiencia en el desarrollo de software desempeñando diferentes roles. En la actualidad se desempeña como Especialista Principal en Ingeniería de Requerimientos de la dirección de desarrollo de la empresa SOFTEL. Su correo electrónico es:
rosalia@softel.cu

Dr. Denis Derivet Thaureaux:
Graduado de Medicina en el año 1994. Especialista 1er Grado en Medicina General Integral en 1999. Experto Funcional del MINSAP. Profesor Instructor. Imparte asignaturas del Perfil de Salud en la Facultad 7. Cursa las maestrías de Gestión de proyectos Informáticos y Bioinformática. Posee 12 años de experiencia. Ha presentado ponencias en eventos nacionales e internacionales. Sus direcciones de correo electrónico son:
denis@softel.cu
derivet@infomed.sld.cu.

Ing. Yosvanys Sánchez Corales:

Profesor graduado de Ingeniero en Informática en el año 2005 en la CUJAE. Ha impartido las asignaturas Programación 3 e Inteligencia Artificial, Forma parte del proyecto de Atención Primaria de Salud de la facultad 7 y actualmente cursa diplomados para la matrícula en la maestría GPI (Gestión de Proyectos Informáticos). Su correo electrónico es:

yscorales@uci.cu

AGRADECIMIENTOS

Agradecemos en primer lugar a nuestro país por permitirnos estudiar esta carrera.

A nuestra tutora Rosalía Cué por toda la ayuda que nos ha brindado y por todo lo que nos ha enseñado.

A nuestros padres por su confianza y por su apoyo en todo momento.

A todos, muchas gracias; sin ustedes no hubiese sido posible la terminación de este trabajo.

DEDICATORIA

A mis padres, quienes me han brindado amor incondicional y siempre me han impulsado a superarme profesionalmente, ofreciéndome aliento constante para lograrlo.

A mi hermano, a quien quiero mucho.

A mi novio, que me ha apoyado en todo momento y me ha enseñado que en los momentos difíciles uno siempre se crece a pesar de las dificultades.

A mis amigos de la UCI, a los que quiero y nunca olvidaré.

Lucelia

DEDICATORIA

A mis padres queridos que son mi guía en la vida, que siempre han creído en mí y me han dado fuerzas para seguir adelante.

A mi abuela Mercedes que me quiere mucho y esperaba con anhelo este resultado.

A todos los que de una forma u otra me han apoyado para lograr este triunfo.

Andro Ernesto.

RESUMEN

En el presente trabajo de tesis, se propone una guía que contiene el flujo de actividades que deben seguir los desarrolladores de software para realizar un buen proceso de análisis y diseño de las aplicaciones, cuando la arquitectura con la que están trabajando es basada en componentes, independientemente de la metodología de desarrollo utilizada.

Para alcanzar esta meta, se estudia la arquitectura basada en componentes y los procesos asociados a la misma. Se analizan los principales proyectos productivos de la universidad de las Ciencias Informáticas (UCI), sobre todo aquellos que utilizan este patrón arquitectónico, para conocer si tienen definida una guía para el análisis y diseño de sus aplicaciones, si conocen los beneficios que les brinda esta arquitectura y si tienen claro el concepto de componentes. Para obtener esta información, se utilizaron entrevistas, realizadas a los arquitectos de estos proyectos. Con ellas se pudo conocer que se carece de una guía que explique cómo realizar este proceso. Muchos ni siquiera tienen definido claramente que es un componente. Aunque todos conocen el principal beneficio que brinda la utilización de esta arquitectura, que es la reutilización de los componentes ya existentes.

La guía propuesta en esta investigación, favorecerá el incremento de la eficiencia en las aplicaciones que se realizan en los proyectos productivos de la UCI. Con ella los arquitectos tendrán menos riesgos de fallar, se agotarán menos recursos humanos, técnicos y financieros. Además, los tiempos de desarrollo para la realización del proyecto podrán disminuirse.

Palabras Claves:
Arquitectura basada en componentes, análisis y diseño de aplicaciones.

TABLA DE CONTENIDO

INTRODUCCIÓN

En nuestro país se realizan grandes esfuerzos por impulsar la industria del software, para ello se hace necesario definir un conjunto de premisas que permitan dar el salto de la producción artesanal a la industrial. Entre ellas, se encuentra la selección de una arquitectura que nos garantice el esquema productivo de una fábrica de software. Es precisamente la arquitectura basada en componentes la que brinda la forma de industrializar el desarrollo mediante el ensamblaje de componentes de software prefabricados.

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. Este se caracteriza por describir, desarrollar y utilizar técnicas basadas en componentes software para la construcción de sistemas abiertos y distribuidos.

A medida que un sistema es más grande y complejo, su arquitectura crece en importancia. La Arquitectura de Software basada en componentes, permite crear sistemas fácilmente escalables, intuitivos de entender y promueve la reutilización de software, reduciendo costos y tiempos de desarrollo. Los componentes de software, en cierta medida, surgen de la necesidad de hacer un uso correcto del software reutilizable dentro de las aplicaciones. En cada iteración, se agregan nuevos componentes hasta consolidar la arquitectura final del sistema.

Con la creación de la Universidad de las Ciencias Informáticas (UCI), se dió un paso de avance para la creación de la infraestructura de la industria cubana del software. En la actualidad, en ella se desarrollan una gran cantidad de proyectos productivos y muchos de ellos utilizan la arquitectura basada en componentes, por lo que se hace necesario, contar con guías que ayuden a los desarrolladores a desempeñar las diferentes actividades al trabajar con este tipo de patrón arquitectónico.

No tenerlas implicaría, que los equipos de proyectos realizan las actividades según sus conocimientos y no de forma estandarizada, donde queda definido el conocimiento de los expertos.

A pesar que la arquitectura basada en componentes brinda grandes beneficios y que existen metodologías que basan sus procesos en este tipo de estilo; no todos saben como partiendo del estudio del negocio, pueden llegar a diseñar los componentes que formarán parte de la aplicación y se carece de una guía que explique cómo realizar este proceso.

Lo planteado anteriormente, constituye una problemática ya que al no existir la guía, se corre el riesgo potencial de no tener éxito en el proyecto, creando dificultades, agotando recursos humanos, técnicos y financieros. Los tiempos de desarrollo necesarios para la realización del proyecto se alargarán por no contar con estándares que expliquen como realizar este flujo de trabajo.

Teniendo en cuenta las consideraciones anteriores se planteó el siguiente **problema a resolver:** ¿Qué guía seguir para realizar el análisis y diseño de las aplicaciones cuya arquitectura está basada en componentes?

En correspondencia con el problema planteado anteriormente el **objeto de estudio** lo constituye el proceso de desarrollo de software utilizando la arquitectura basada en componente y los procesos asociados a la misma.

El **campo de acción** que engloba este trabajo incluye: La disciplina de Análisis y Diseño en el proceso de desarrollo de software para aplicaciones con arquitectura basadas en componentes.

Para dar solución al problema presentado se plantea el siguiente **objetivo general**: Proponer una guía que contenga el flujo de actividades que se deben seguir para realizar el análisis y diseño de una aplicación cuya arquitectura está basada en componentes.

Los **objetivos específicos** que se deben cumplimentar son:

- Analizar la arquitectura basada en componentes para ver sus principales características y las ventajas que brinda la misma.
- Analizar los proyectos más importantes de la UCI para ver si los que utilizan la arquitectura basada en componentes tienen una guía para realizar el análisis y diseño de sus aplicaciones.
- Definir el proceso de análisis y diseño con una guía para su realización que permita a los desarrolladores tener un mayor éxito en su trabajo.

Para cumplir con estos objetivos y resolver la situación problémica planteada, se proponen las siguientes **tareas:**

1) Revisar bibliografía sobre arquitectura basada en componentes.
2) Aplicar entrevistas a arquitectos de los proyectos productivos más importantes de la universidad.
3) Estudiar herramientas de procesamiento de información.
4) Procesar información recogida en las entrevistas aplicadas.
5) Definir una guía para la ejecución del proceso de análisis y diseño de sistemas con arquitectura basada en componentes.

Como resultado de este trabajo, se pretende obtener una guía que contenga todo el flujo de actividades que deben seguir los desarrolladores de software para realizar el análisis y diseño de un aplicación cuya arquitectura está basada en componentes, esta será de gran utilidad para el

profesional informático, ya que facilitará su trabajo a la hora de desarrollar cualquier tipo de aplicación.

El contenido de este trabajo se encuentra estructurado de la siguiente manera:

En un primer capítulo, “Fundamentación Teórica”, se hace un análisis bibliográfico donde se investiga todo lo relacionado con el Desarrollo de Software Basado en Componentes y se realiza una exposición de los conceptos asociados al dominio del problema.

A continuación, en el capítulo 2, “Arquitectura de software en los proyectos productivos de la UCI”, se desarrolla un estudio sobre los patrones arquitectónicos que se utilizan en los proyectos productivos de la universidad, dicho estudio se centra principalmente en aquellos que utilizan la arquitectura basada en componentes.

Por último en el capítulo 3, “Guía para el análisis y diseño de aplicaciones con arquitectura basada en componentes”, se propone una guía para ayudar a los desarrolladores de software a realizar el análisis y diseño de las aplicaciones con arquitectura basada en componentes, explicándoles todas las actividades que deben realizar.

CAPÍTULO

FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

El principal objetivo del presente capítulo es abordar distintos aspectos que se utilizarán como soporte teórico de la investigación realizada.

Para el logro del objetivo planteado se enfatizan los conceptos de: aplicación, arquitectura de software, componente, arquitectura basada en componentes y se realiza un estudio de los aspectos más importantes de la disciplina de análisis y diseño. Por último se realiza un estudio sobre el desarrollo basado en componentes.

1.2 Conceptos asociados al dominio del problema.

1.2.1 ¿Qué es una aplicación?

En informática las aplicaciones son los programas con los cuales el usuario final interactúa, es decir, son aquellos programas que permiten la interacción entre el usuario y la computadora. Esta comunicación se lleva acabo cuando el usuario elige entre las diferentes opciones o realiza actividades que le ofrece el programa. Los procesadores de textos, hojas de cálculo, bases de

datos, programas de dibujo y paquetes estadísticos, programas de correo electrónico, entre otros, son aplicaciones.

1.2.2 Análisis y Diseño

Hoy en día existen en el mundo diversas metodologías para el desarrollo de aplicaciones, una de esta es el Rational Unified Process (RUP), la cual es muy utilizada en la UCI. Esta metodología agrupa las actividades en nueve disciplinas de trabajo. Una de ellas es el Análisis y Diseño (ver Fig.1.1), en la misma se describe como el sistema será realizado a partir de la funcionalidad prevista y de las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.

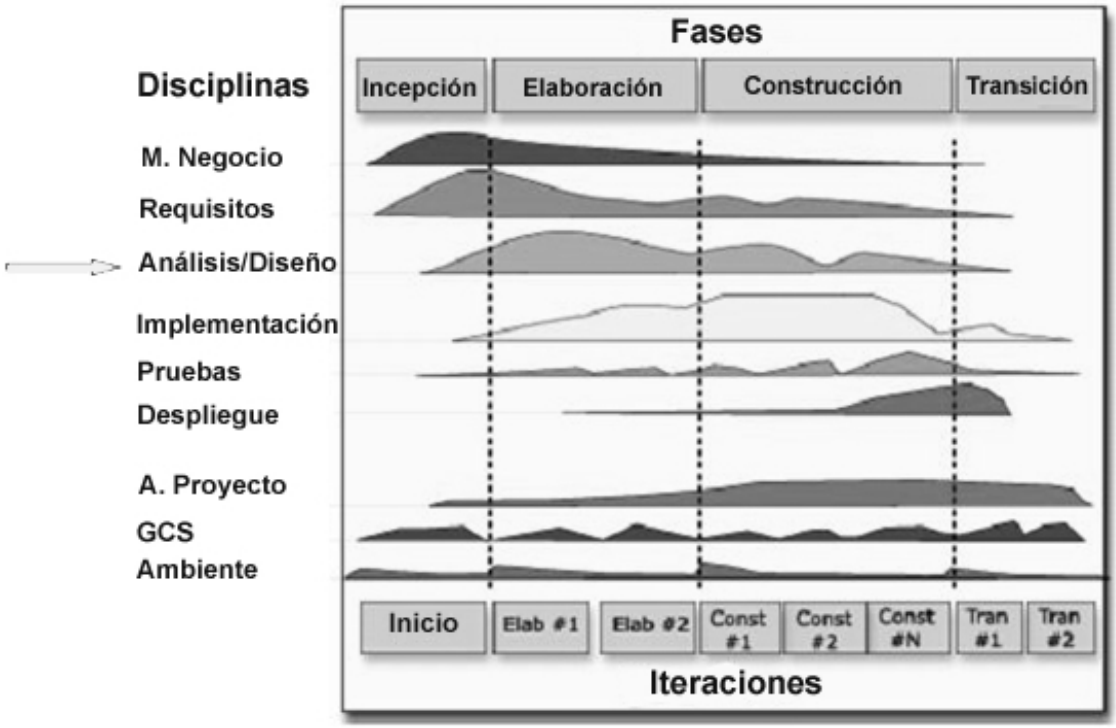

Fig. 1.1: Ciclo de vida de RUP.

El **análisis** consiste en obtener una visión del sistema que se preocupa por ver QUÉ debe hacer, de modo que sólo se interesa por los requisitos del nuevo sistema.

El **diseño** es un refinamiento del análisis que tiene en cuenta COMO serán implementados los requisitos funcionales para que satisfaga los requisitos no funcionales o de calidad definidos en el análisis. El diseño debe ser suficiente para que el sistema pueda ser implementado sin ambigüedades. De hecho, cuando la precisión del diseño es muy grande, la implementación puede ser hecha por un generador automático de código.

Al final de la fase de inicio hay que definir una arquitectura candidata:

- Crear un esquema inicial de la arquitectura del sistema.
- Identificar clases de análisis y actualizar las realizaciones de los casos de uso con las interacciones de las clases de análisis.

Durante la fase de elaboración se va refinando esta arquitectura hasta llegar a su forma definitiva. En esta fase se desarrolla el proceso por iteraciones, en cada iteración hay que analizar el comportamiento para diseñar componentes. Además si el sistema usará una base de datos, habrá que diseñarla también, obteniendo un modelo de datos. El resultado final más importante de este flujo de trabajo será el modelo de diseño. Consiste en colaboraciones de clases, que pueden ser agregadas en paquetes y subsistemas. Otro producto importante de este flujo es la documentación de la arquitectura software, que captura varias visiones arquitectónicas del sistema.

RUP define el análisis y el diseño como una única disciplina en la que hay actividades que se realizan desde la fase de Inicio.

Análisis

Los trabajadores que participan son:

- **Arquitecto:** Responde por la integridad del modelo de análisis y por la arquitectura del modelo de análisis.
- **Ingeniero de casos de uso:** Responde por la integridad de una o más realizaciones de casos de uso (descripción textual del flujo de sucesos, Diagrama de clases que muestra las clases del análisis participantes y los diagramas de interacción que muestran la realización de un flujo particular del caso de uso en términos de objetos del análisis), así como de que respondan los casos de uso a los requisitos que engloba cada uno.
- **Ingeniero de componentes:** Define y mantiene las clases y las relaciones entre ellas y la integridad de uno o varios paquetes del análisis.

Los artefactos que construyen estos trabajadores son:

- **Modelo de análisis:** Contiene clases del análisis y sus objetos organizados en paquetes que colaboran.
- **Clases de análisis:** Se centran en los requisitos funcionales y son evidentes en el dominio del problema porque representan conceptos y relaciones del dominio. Tienen atributos y entre ellas se establecen relaciones de asociación, agregación, composición, generalización especialización y tipos asociativos.
- **Realización de casos de uso del análisis:** Describe cómo se lleva a cabo y se ejecuta un caso de uso determinado en término de las clases del análisis y de sus objetos en interacción.
- **Paquete de análisis:** Organiza los artefactos del análisis en piezas manejables.

- **Descripción de la arquitectura (vista del modelo de análisis):** Muestra los artefactos significativos para la arquitectura (los anteriores).

Actividades:

- **Análisis de la arquitectura:** Identificar paquetes y clases del análisis.
- **Analizar un caso de uso:** Identificar clases cuyos objetos son necesarios para realizar el caso de uso y distribuir el comportamiento entre los objetos que participan en el caso de uso.
- **Analizar una clase:** Identificar atributos de las clases y relaciones entre ellas.
- **Analizar un paquete:** Garantizar alta cohesión y bajo acoplamiento de los paquetes y describir las relaciones entre ellos.

Modelo del análisis

En la construcción del modelo de análisis se tienen que identificar las clases que describen la realización de los casos de uso, los atributos y las relaciones entre ellas. Con esta información se construye el Diagrama de clases del análisis, que por lo general se descompone para agrupar las clases en paquetes. Esta descomposición tiene impacto por lo general en el diseño e implementación de la solución.

- El Modelo de Análisis es el resultado de la actividad de analizar los Casos de Uso.

Ventajas que proporciona la realización del modelo de análisis:

- Suaviza la transición al Diseño.
- Apoya el cambio a otra plataforma de programación.

- Sirve para tener una visión general de la propuesta del sistema**.**
- Sirve para planificar y dividir el diseño e implementación en pequeños módulos.
- Apoya la aplicación de reingeniería a aplicaciones existentes.

Clases del Análisis:

- Representan la abstracción a un concepto.
- Las operaciones y atributos son poco detallados.
- Siempre encajan en tres estereotipos básicos.

Diseño

El diseño es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable y sólida, y crear un plano del modelo de implementación. Durante la fase de construcción, cuando la arquitectura es estable y los requisitos están bien entendidos, el centro de atención se desplaza a la implementación.

El modelo de diseño está muy cercano al de implementación, lo que es natural para guardar y mantener el modelo de diseño a través del ciclo de vida completo del software.

Propósitos del diseño:

- Adquirir una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia y tecnologías de interfaz de usuario.

- Crear una entrada apropiada y un punto de partida para actividades de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.
- Descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo.
- Capturar las interfaces entre los subsistemas antes en el ciclo de vida del software, lo cual es muy útil cuando utilizamos interfaces como elementos de sincronización entre diferentes equipos de desarrollo.

Trabajadores

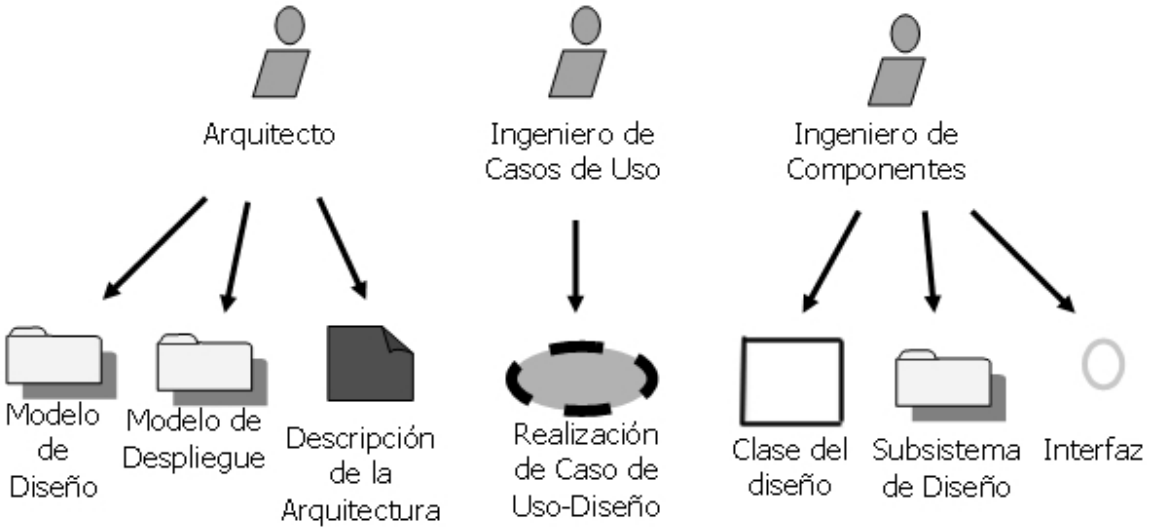

Fig. 1.2: Trabajadores que participan en el diseño.

Artefactos

Modelo de Diseño.

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar.

Sirve de abstracción de la implementación y es utilizada como entrada fundamental de las actividades de implementación.

El modelo de diseño se representa por un sistema de diseño que denota el subsistema de nivel más alto del modelo. La utilización de otro subsistema es, entonces, una forma de organización del modelo de diseño en porciones más manejables.

En el modelo del diseño, los casos de uso son realizados por las clases del diseño y sus objetos. Esto se representa por colaboraciones en el modelo de diseño y denota la realización de casos de uso del diseño. La realización de casos de uso del diseño es diferente de la realización de casos de uso de análisis.
Los artefactos del **Modelo de Diseño** son:

- Modelo de Despliegue.
- Descripción de la Arquitectura.
- Realización de Casos de Uso.
- Clase del Diseño.
- Subsistema de Diseño.
- Interfaz.

(Pressman 2005)

1.2.3 ¿Qué es la arquitectura de software?

La arquitectura de software son las estructuras del sistema, la cual consta de los elementos del software, las propiedades de estos elementos que son externamente visibles y las relaciones entre ellos.(Institute 2007)

La Arquitectura del Software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño.

El objetivo principal de la Arquitectura del Software es aportar elementos que ayuden a la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participen en un proyecto. Para conseguirlo, la Arquitectura del Software construye abstracciones, materializándolas en forma de diagramas (blueprints) comentados.

No hay estándares en cuanto a la forma y lenguaje a utilizar en estos blueprints. De todas formas, existe un consenso en cuanto a la necesidad de organizar dichas abstracciones en vistas, tal y como se hace al diseñar un edificio. La cantidad y tipos de vistas difieren en función de cada tendencia arquitectónica.

1.2.4 ¿Qué es un componente?

En este trabajo se utiliza la definición de Sterling Software, que plantea que un componente es:

- Una parte reutilizable que encapsula elementos del modelo (por ejemplo una DLL, documentos, tablas, biblioteca, etc.).
- Un paquete de software el cual ofrece servicios a través de sus interfaces.

- Un paquete de Software que pude ser usado para construir aplicaciones o componentes más grandes.(Software 2000)

1.2.5 ¿Qué es la arquitectura basada en componentes?

La definición de la arquitectura basada en componentes cubre aspectos únicamente lógicos y es totalmente independiente de la tecnología con la cual se implementarán los mismos y sobre la cual se hará el despliegue del sistema. Esta vista lógica nos permite medir el nivel de acoplamiento del sistema y razonar sobre los efectos de modificar o reemplazar un componente. La independencia de la tecnología nos permite abstraernos de los tecnicismos de éstas, así como elegir la más apta dependiendo del sistema que se esté desarrollando.(Vignaga, Perovich 2005)

1.3 Objeto de Estudio

1.3.1 Desarrollo de Software Basado en Componentes (DSBC)

EL desarrollo de software basado en componentes o la Ingeniería de Software basada en Componentes surgió a finales de los 90 como un enfoque basado en la reutilización para el desarrollo de los Sistemas de Software. Bajo este enfoque, los desarrolladores más que diseñar y después buscar componentes reutilizables, primero buscan estos componentes; de esta forma basan el diseño del software en los componentes disponibles.

Puede ser que el diseño sea menos eficiente que uno de propósito específico; sin embargo, los bajos costos en el desarrollo, una entrega más rápida del sistema y el incremento en la fiabilidad del sistema, compensan esto.

Bajo un desarrollo como el descrito anteriormente, la Arquitectura de Software del Sistema a ser construido se convierte en un factor de importancia para lograr que éste tenga un alto nivel de calidad. Poseer una buena Arquitectura de Software es de suma importancia, ya que ésta es el corazón de todo Sistema de Software y determina cuáles serán los niveles de calidad asociados al sistema. (Martínez 2003)

Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica. En un principio, hacia 1994, se planteaba como una modalidad que extendía o superaba la tecnología de objetos.

Con el paso de los años el antagonismo se fue aplacando y las herramientas (orientadas a objeto o no) fueron adaptadas para producir componentes. En la mayoría de los casos, los componentes terminan siendo formas especiales de DLLs, que necesitan registración y que no requieren que sea expuesto el código fuente de la clase.

El marco arquitectónico estándar para la tecnología de componentes está constituido por los cinco puntos de vista de RM-ODP (Empresa, Información, Computación, Ingeniería y Tecnología).

La evaluación dominante del estilo de componentes subraya su mayor versatilidad respecto del modelo de objetos, pero también su menor adaptabilidad comparado con el estilo orientado a servicios. Las tecnologías de componentes del período de inmadurez, asimismo, se consideraban afectadas por problemas de incompatibilidad de versiones e inestabilidad que ya han sido largamente superados en toda la industria.(Reynoso, Kicillof 2004)

Hoy en día podemos decir que existe un consenso al considerar el desarrollo basado en componentes como un avance significativo hacia la construcción de sistemas mediante el ensamblado de componentes prefabricados. El DSBC es una aproximación del desarrollo de

software, la cual trata de sentar las bases para el diseño y desarrollo de aplicaciones abiertas y distribuidas mediante el ensamblaje de partes de software reutilizables.

Dicha disciplina cuenta actualmente con un creciente interés, tanto desde el punto de vista académico como desde la industria, en donde la demanda de mecanismos y herramientas de desarrollo basados en componentes es cada día mayor.

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

El paradigma orientado a objetos enfatiza la creación de clases que encapsulan tanto los datos como los algoritmos que se utilizan para manejar los datos. Si se diseñan y se implementan adecuadamente, las clases orientadas a objetos son reutilizables por las diferentes aplicaciones y arquitecturas de sistemas basados en computadora.

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

enlazado dinámico y composición tardía, la extensión de los lenguajes de descripción de interfaces (IDLs) tradicionales, etc.

Una de las principales ventajas del desarrollo de software basado en componentes se basa en la "reutilización" de los mismos. De esta forma, los componentes se diseñan y desarrollan con el objetivo de poder ser reutilizados en otras aplicaciones, reduciendo el tiempo de desarrollo, mejorando la fiabilidad y flexibilidad del producto final (al usar componentes ya probados previamente), y siendo más competitivos en costos. Con el DSBC se logra un mayor retorno de la inversión (ROI), que es un aspecto muy importante que deben valorar los arquitectos de software a la hora de escoger una arquitectura.

Durante algunos años, DSBC fue referida como una filosofía conocida como "compre, y no construya" promulgada por Fred Brooks en 1987 y que abogaba por la utilización de componentes prefabricados sin tener que desarrollarlos de nuevo.

Aunque hasta ahora la reutilización suele suceder principalmente a nivel interno dentro de las organizaciones, el uso de los componentes comerciales comienza a extenderse. De esta forma se habla de componentes COTS (Commercial-Off-The-Shelf), que han sido definidos como una clase especial de componentes software, normalmente de grano grueso, que presentan las siguientes características:

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

La cada vez mayor disponibilidad y uso de este tipo de componentes está impulsando notablemente la creación de un mercado global de componentes COTS, que está pasando de ser la utopía de hace unos años a una realidad cada vez más cercana. La tecnología básica de componentes comienza a estar lo suficientemente madura (a través de plataformas de objetos distribuidos como EJB, CORBA o .NET) como para que numerosas empresas la adopten en sus nuevos desarrollos y sistemas de información.

Incluso el gobierno y el ejército norteamericano han anunciado su uso, y han empezado a apostar por la utilización de componentes comerciales como única vía de mantener sus costos de desarrollo y mantenimiento de software bajo control. Asimismo, están empezando a proliferar las empresas que venden con éxito componentes software al mercado general, como pueden ser componentsource, flashline, wrldcomp, etc.

En el desarrollo de software basado en componentes, se pueden identificar varias tareas específicas para la construcción de aplicaciones, entre estas se encuentra:

1. La búsqueda (**trading**) de componentes que satisfagan los requisitos impuestos tanto por el cliente como por la arquitectura de la aplicación.
2. La **evaluación** de los componentes candidatos para seleccionar los más idóneos.
3. La **adaptación** y/o extensión de los componentes seleccionados para que se ajusten a los requisitos anteriores.
4. La **integración**, configuración e interconexión de dichos componentes para construir la aplicación final.

Es importante señalar que un factor imprescindible en todas esas tareas es la documentación de los componentes, pues es preciso contar con especificaciones completas, concisas y precisas de los componentes para poder llevarlas a cabo. La mayoría de las propuestas existentes para

documentar componentes se basan en el uso de interfaces, los cuales proporcionan un mecanismo para describir la funcionalidad de los componentes.

Sin embargo, los lenguajes de descripción de interfaces (IDLs) existentes permiten documentar sólo los aspectos “sintácticos” (o “estáticos”) de dicha funcionalidad, sin tratar otros aspectos de vital importancia como son los protocolos de acceso a los servicios, el comportamiento de los métodos, o la semántica de las operaciones de los componentes.

Por supuesto, dichos IDLs tampoco permiten tratar los aspectos “extra-funcionales” de los componentes, como la fiabilidad, la seguridad, o las prestaciones.

Respecto a los problemas de trading, lo que nos encontramos es que solo existen propuestas para la búsqueda de objetos dentro de ciertos modelos (como RM-ODP) o plataformas (CORBA). Sin embargo, son demasiado básicas y simplistas como para poder ser utilizadas a nivel de componentes COTS.

En este sentido, comienzan a aparecer *traders* específicos de componentes, como por ejemplo el denominado “COTStrader”. Dicho *trader* permite buscar y seleccionar componentes a partir de la definición en plantillas XML de su funcionalidad y otras propiedades de calidad.

La segunda tarea se centra en los procesos y herramientas para la evaluación y selección de componentes COTS. Aparte de tener en cuenta los requisitos funcionales de la aplicación, es necesario considerar otros factores que también intervienen a la hora de seleccionar componentes. El problema es que este tipo de requisitos, denominados “extra-funcionales” son difíciles de evaluar, aunque todos somos conscientes de la importancia que representan.

Este tipo de factores priman muchas veces incluso más que los funcionales, pues un diseñador hasta es capaz de adaptar la arquitectura de un sistema para poder incluir un componente que

se desea que esté, o bien para evitar la presencia de un componente de un fabricante en el cual no se confía.

Algunas de las causas que intervienen en esa omisión de la valoración en los requisitos extra-funcionales son: la no existencia de una definición consensuada de las características o atributos de calidad que hay que medir, la situación de revisión actual de los estándares internacionales relativos a la calidad del producto software (fundamentalmente las normas ISO-9126 e ISO-14598), y la ausencia casi total de métricas que pudieran dar una estimación más objetiva de estos atributos.

Además, es importante señalar que los estándares internacionales proporcionan guías y modelos de calidad para temas muy generales, pero no suelen estar pensados para ofrecer soluciones en temas muy concretos. Igual les sucede a los procesos de selección de productos software, que suelen ser demasiado genéricos para tratar de forma efectiva con la problemática particular de los componentes COTS.

Una vez seleccionados los componentes candidatos, normalmente es preciso realizar una adaptación y/o extensión de los mismos a la hora de integrarlos. En general, al tratar de combinar los componentes suelen aparecer tres tipos de problemas típicos de los ambientes COTS: las lagunas (*gaps*), los solapamientos (*overlappings*), y la incompatibilidad de interfaces (*mismatches*). Los dos primeros aparecen al unir todos los componentes, pues podemos descubrir que alguno de los servicios requeridos por la arquitectura de la aplicación no está cubierto por ninguno de los componentes seleccionados, o bien que algunos componentes ofrecen servicios que se solapan parcialmente.

El problema del emparejamiento es de naturaleza distinta, pues surge al tratar de emparejar o conectar componentes cuyas interfaces no son compatibles, y que por tanto deben ser adaptadas como paso previo a su conexión.

A pesar de estos problemas, los beneficios que proporciona el DSBC están extendiendo su uso en entornos industriales de desarrollo de software, sobre todo en cuanto a reducción de esfuerzos y costos de desarrollo y mantenimiento, y a la mejora de la calidad final de los productos y sistemas construidos. Esta extensión de su uso está favoreciendo la aparición de numerosas metodologías y herramientas de desarrollo para realizar un efectivo DSBC. Sin embargo, esta disciplina dista aún de ser una verdadera "ingeniería", pues aún no se dispone ni de métricas adecuadas, ni de procesos precisos, ni de normativa internacional que la regule.

En este sentido, la definición de métricas de calidad para componentes y aplicaciones software, especialmente las desarrolladas en base a componentes COTS, aparece como una necesidad imperiosa.(Manuel F. Bertoa 2002)

Como se ha podido observar, la reutilización de componentes proporciona grandes beneficios a los ingenieros de software. Según estudios de reutilización, QSM Associates, Inc. Informa que el ensamblaje de componentes lleva a una reducción del 70 por 100 de tiempo de ciclo de desarrollo, un 84 por 100 del costo del proyecto y un índice de productividad del 26.2, comparado con la norma de industria del 16.9.

Aunque estos resultados están en función de la robustez de la biblioteca de componentes, no hay duda de que el ensamblaje de componentes proporciona ventajas significativas para los ingenieros de software.

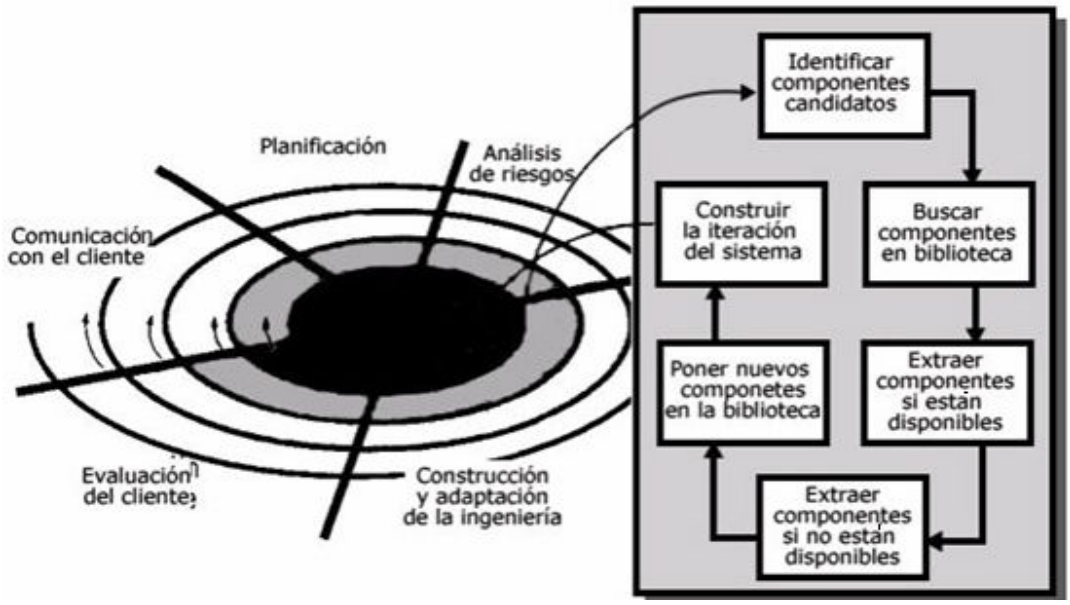

Fig. 1.3: Desarrollo de Software Basado en Componentes.

El modelo de desarrollo basado en componentes (Fig. 1.3) incorpora muchas de las características del modelo en espiral. Es evolutivo por naturaleza, y exige un enfoque iterativo para la creación del software. Sin embargo, el modelo de desarrollo basado en componentes configura aplicaciones desde componentes preparados de software (llamados <<clases>>).

El proceso unificado de desarrollo de software, el cual es el más conocido en la UCI, es un modelo de desarrollo basado en componentes. Utilizando el Lenguaje de Modelado Unificado (UML), el proceso define los componentes que se utilizarán para construir el sistema y las interfaces que conectarán los componentes. Utilizando una combinación del desarrollo iterativo e incremental, el proceso unificado define el modelo del sistema aplicando un enfoque basado en escenarios (desde el punto de vista del usuario). Entonces acopla el modelo con un marco de trabajo arquitectónico que identifica la forma que tomará el software.(Pressman 2005)

1.4 Conclusiones

El énfasis de este capítulo ha estado en profundizar acerca de conceptos fundamentales relacionados con el desarrollo de software basado en componentes y el estudio de los aspectos más importantes de la disciplina de análisis y diseño.

Se pudo ver que el DSBC permite crear sistemas fácilmente escalables, esto significa que las aplicaciones pueden ser modificadas con gran facilidad a medida que van apareciendo nuevos cambios. Además, se pudo observar que el principal beneficio que brinda utilizar componentes es la reutilización, permitiendo que los tiempos de desarrollo se reduzcan. Con lo que se logrará un mayor éxito en el desarrollo de las aplicaciones.

En el próximo capítulo, se realiza un estudio sobre los patrones arquitectónicos que se utilizan en los proyectos productivos de la Universidad de las Ciencias Informáticas, este se enfoca principalmente en aquellos que utilizan la arquitectura basada en componentes.

CAPÍTULO

UTILIZACIÓN DE LA ARQUITECTURA DE SOFTWARE EN LOS PROYECTOS PRODUCTIVOS DE LA UCI.

2.1 Introducción

En el presente capítulo se realiza el análisis de la arquitectura con la que trabajan los principales proyectos productivos de la universidad. Este se basó en la aplicación de entrevistas a líderes y arquitectos de los proyectos productivos seleccionados. El mayor peso de las entrevistas indaga sobre el tipo de arquitectura con la que trabajan, específicamente si lo hacen con arquitectura basada en componente, qué entienden los arquitectos de los proyectos por componente y si alguno tiene algún método o procedimiento definido a la hora de realizar el análisis y diseño utilizando este tipo de arquitectura.

2.2 Método y Técnica utilizada para la obtención y procesamiento de la información

Para la obtención de la información, se decidió aplicar el método de entrevistas, que se basa en la técnica de obtención de información mediante el diálogo mantenido en un encuentro formal y planeado, entre una o más personas entrevistadoras y una o más entrevistadas, en el que se transforma y sistematiza la información conocida por éstas.

Las entrevistas se clasifican en tres tipos fundamentales:

- *Estructuradas*: Consiste en realizar preguntas estudiadas y bien definidas, cuyas respuestas pueden ser:

    a. Respuestas abiertas: el entrevistado responde libremente a las preguntas realizadas por el entrevistador.
    b. Respuestas cerradas: el entrevistado elige entre una serie predefinida de respuestas.

- *No estructuradas*: Donde tanto las preguntas como las respuestas son libres.
- *Mixta:* Hacemos preguntas de los dos tipos.

Para el estudio, se decidió usar el método de las entrevistas cerradas (ver anexo 1) donde se le dió a los entrevistados un conjunto de posibles respuestas para que ellos pudieran seleccionar la que consideraban correcta, buscando de esta forma uniformidad, siendo así más fácil de administrar y de evaluar los resultados.

La entrevista fue confeccionada a partir de los datos que aportaron los Vice-Decanos de Producción de cada facultad, que informaron cuales eran los proyectos más importantes. Posteriormente se contactó a cada líder o arquitecto y estos ayudaron a la culminación del objetivo. Según los datos brindados por los Vice-Decanos de Producción, la UCI cuenta con 60 proyectos que son importantes, de los cuales fueron entrevistados los arquitectos o líderes de 32, representando aproximadamente un 53%, válido para sacar buenas conclusiones.

Se utilizó la herramienta Microsoft Office InfoPath 2003, la cual generó todas las entrevistas procesadas al Microsoft Office Excel 2003, en esta última se utilizaron fórmulas de conteo y gráficos para procesar y analizar la información recogida.

La herramienta Microsoft Office InfoPath 2003 fue diseñada para mejorar el proceso de recolección de información y facilita la reutilización de esa información en toda la organización. InfoPath es el "principal cliente inteligente" para los servicios Web XML, una aplicación fácil de usar que permite a las personas que trabajan con información en su escritorio utilizar los servicios Web XML y aprovechar la información empresarial. (Gonzalez 2005)

Como miembro de Microsoft Office System, InfoPath 2003 da soporte a tres actividades claves: la creación de formularios dinámicos, el llenado de estos formularios y el envío de estos formularios a los sistemas y procesos que necesitan la información recopilada en los formularios. (autores 2005)

La herramienta Toolinf fue desarrollada para Microsoft Excel, en ella se automatizan procedimientos que forman parte del análisis de la información. Con su utilización se logró una mayor agilidad y eficiencia en el análisis de las entrevistas realizadas.

2.3 Análisis de los resultados.

En este epígrafe se dan a conocer una serie de datos que brindan los resultados obtenidos a partir de las entrevistas realizadas a los proyectos productivos de la universidad.

Las primeras preguntas abordan temas generales como el tamaño, tipo de aplicación y metodología que utilizan los proyectos, pasando ya después a preguntas más concentradas de la arquitectura basada en componente.

De las entrevistas realizadas se hizo un análisis de los proyectos que utilizan el patrón de arquitectura basado en componente, en el mismo se pudo ver que de los 32 proyectos

entrevistados, 21 trabajan con este tipo de arquitectura, representando aproximadamente un 65%. A partir de este resultado se hizo un estudio para ver si los arquitectos que utilizan este patrón arquitectónico tienen bien definido que es un componente, si tienen dominio de cómo realizar el ensamblaje de los mismos, si conocen los beneficios que estos les proporcionan y si alguno tiene algún método definido para el análisis y diseño al trabajar con este tipo de arquitectura.

Resultados de la entrevistas.

2.3.1 Tamaño de los proyectos productivos de la UCI

Se pudo determinar que la mayoría de los proyectos son medianos o grandes. A continuación se muestra una gráfica que describe el estudio realizado.

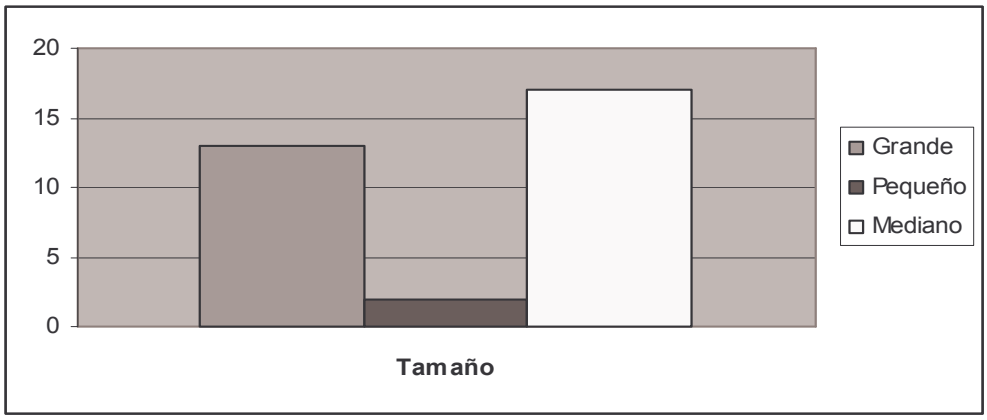

Fig. 2.1: Tamaño de los proyectos.

2.3.2 Tipos de aplicaciones que desarrollan los proyectos productivos.

Una aplicación es un programa informático que lleva a cabo una función con el objetivo de ayudar a un usuario a realizar una determinada actividad. Actualmente la UCI trabaja algunas de estas aplicaciones, las cuales son de gran interés e importancia en el mundo de la informática.

En las entrevistas realizadas a los 32 proyectos de la universidad se observa que la mayoría trabajan con dos tipos de aplicaciones fundamentalmente, que son la de Gestión y la Web, incluyendo que algunos trabajan las dos en conjunto.

Las aplicaciones de gestión son aquellas que se diseñan para sustituir uno o varios procedimientos, tanto comerciales como administrativos, que habitualmente realiza una persona en una empresa o institución. (autores 2001)

Una aplicación Web es un sistema informático que los usuarios utilizan accediendo a un servidor Web a través de Internet o de una intranet. Las aplicaciones Web son populares debido a la practicidad del navegador Web como cliente ligero. La habilidad para actualizar y mantener aplicaciones Web sin distribuir e instalar software en miles de potenciales clientes es otra razón de su popularidad. Aplicaciones como los webmails, wikis, weblogs, MMORPGS, tiendas en línea, son ejemplo de algunas de ellas.

No se quedan atrás los otros tipos de aplicaciones que incluyen las de Destock y las de Realidad Virtual. La gran minoría trata con el Procesamiento de Imágenes siendo esta el tipo de aplicación que menos se trabaja en la universidad.

Como se puede ver en la siguiente gráfica, la combinación de Gestión-Web tiene un mismo alcance que las de otras aplicaciones por lo que hay un balance entre dos grupos fundamentales.

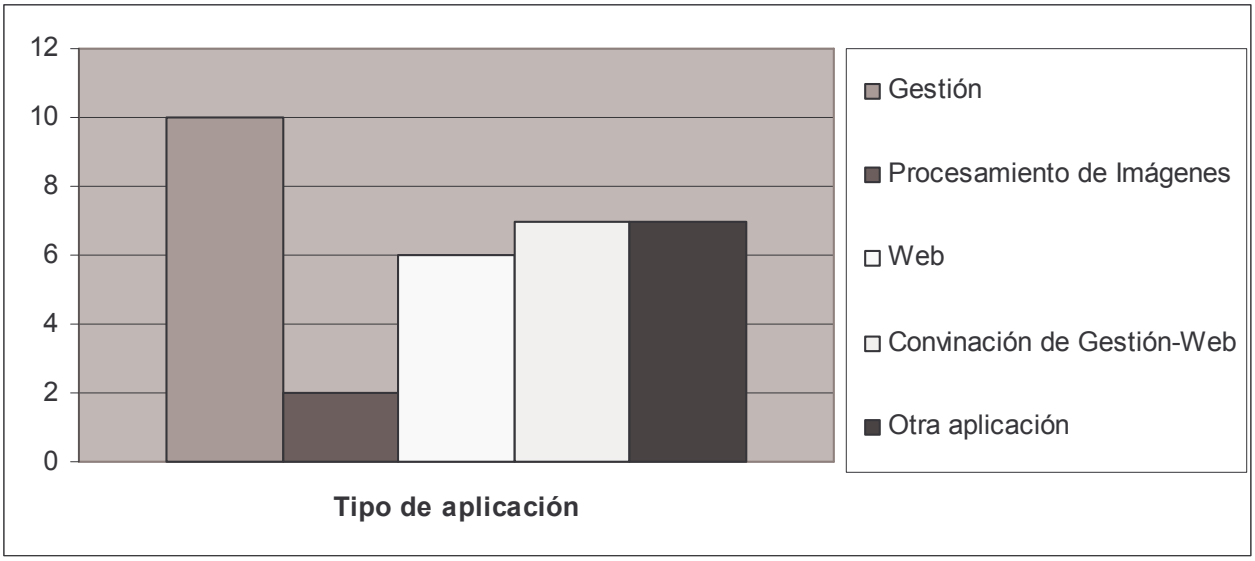

Fig. 2.2: Tipo de aplicación.

2.3.3 Metodologías que se utilizan en los proyectos.

Todos los procesos de desarrollo del software deben ser guiados por una metodología, encargada de elaborar estrategias de desarrollo de software que promueven prácticas adoptivas centradas en las personas o los equipos.

Existen diferentes tipos de metodologías, ágiles, tradicionales, cada una con características fundamentales, por las que se debe regir un desarrollador a la hora de elegir cual va a seguir en el desarrollo del producto software.

En los proyectos productivos de la universidad se utilizan generalmente dos metodologías XP y RUP, siendo esta última la más utilizada en la mayoría de los proyectos. En la siguiente gráfica se muestra este resultado.

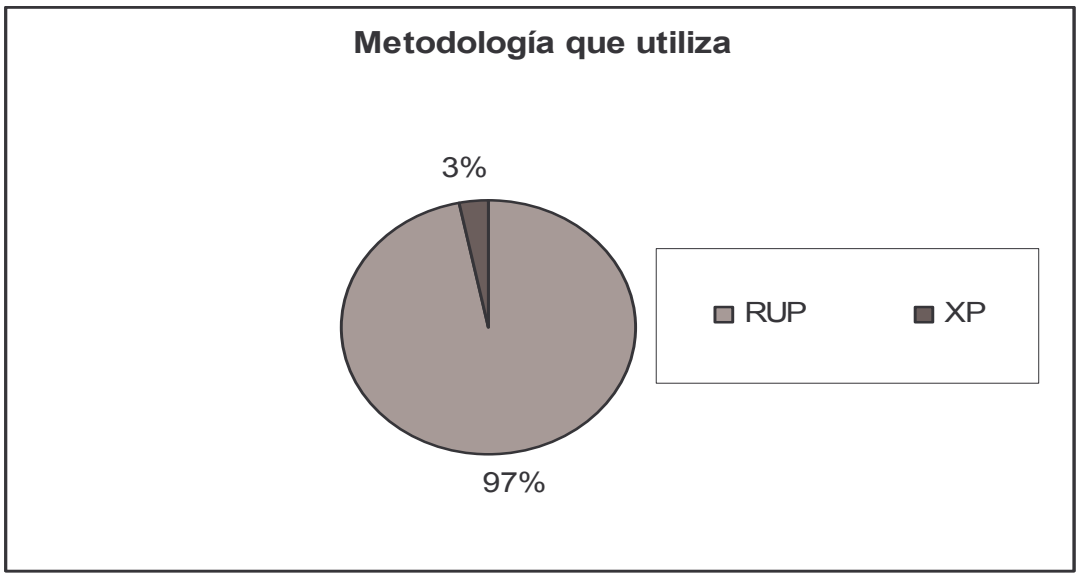

Fig. 2.3 Metodología que se utiliza.

Se pudo determinar que la mayoría los proyectos medianos y grandes utilizaban RUP como proceso de desarrollo, ya que la misma está preparada para desarrollar grandes y complejos proyectos. Esta metodología es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo a través del UML, y trabajo de muchas metodologías utilizadas por los clientes.

XP es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipos pequeños (hasta 10 programadores), preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. XP se define como especialmente adecuada

para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.(Letelier, Penadés 2002)

2.3.4 Patrones de arquitectura de software utilizados en los proyectos.

Los patrones de arquitectura son formas de descomponer, conectar y relacionar sistemas, trata conceptos como: niveles, tuberías y filtros. Es un nivel de abstracción mayor que el de los Patrones de Diseño.

Expresan un paradigma fundamental para estructurar u organizar un sistema software. Proporcionan un conjunto de subsistemas o módulos predefinidos, con reglas y guías para organizar las relaciones entre ellos. Ejemplo:

- Capas (Layers)
- Aplicaciones: JVM, API, Windows NI
- Pipes and Filtres
- Aplicaciones: UNIX
- Pizarrón (Blackboard)
- Aplicaciones: Hearsay, Inteligencia Artificial

Los patrones arquitectónicos también son llamados estilo arquitectónico o variante arquitectónica, estos definen a una familia de sistemas informáticos en términos de su organización estructural. Un estilo arquitectónico describe componentes y las relaciones entre ellos con las restricciones de su aplicación, la composición asociada y el diseño para su construcción.

Los sistemas empresariales distribuidos pueden agrupar los siguientes estilos arquitectónicos:

- Modelo-Vista-Controlador (MVC)
- Arquitecturas en Capas
- Arquitecturas Orientadas a Objetos
- Arquitecturas Basadas en Componentes
- Arquitecturas Orientadas a Servicios

(Jara 2004)

En la UCI la mayoría de los proyectos productivos trabajan con estos estilos de arquitectura. Muchos emplean uno solo de acuerdo a las necesidades que tengan en su proyecto, pero la gran mayoría hacen una combinación de algunos de ellos para así tener una mejor eficiencia y resultado en su trabajo, la siguiente gráfica recoge el resultado obtenido a partir los 32 proyectos entrevistados.

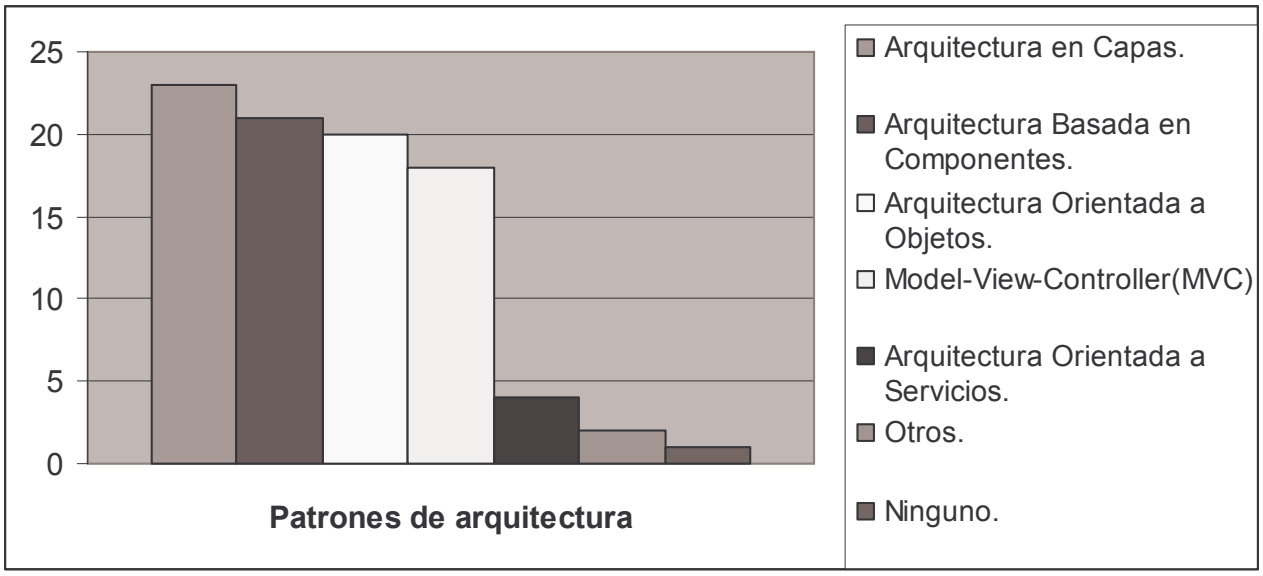

Fig. 2.4: Patrones de arquitectura.

El principal objetivo de este trabajo es el estudio de la arquitectura basada en componente. A partir de aquí, se realiza un análisis de los proyectos que trabajan con esta arquitectura. Apoyados en los datos que brinda la gráfica se puede observar que 21 proyectos la utilizan, según el estudio realizado hasta el momento se puede ver que todos los que utilizan este patrón arquitectónico utilizan RUP como proceso de desarrollo.

2.3.5 Beneficios que brinda la arquitectura basada en componente.

Como se pudo ver en el Capítulo 1, existe un consenso al considerar el desarrollo basado en componentes como un avance significativo hacia la construcción de sistemas mediante el ensamblado de componentes prefabricados.

En general, el desarrollo de software basado en componentes puede verse como una extensión natural de la programación orientada a objetos dentro del ámbito de los sistemas abiertos y distribuidos. Este paradigma se basa en el uso de los componentes de software como entidades básicas del modelo, entendiendo por componente “una unidad de composición de aplicaciones software que posee un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes, de forma independiente en tiempo y espacio”. (Manuel F. Bertoa 2002)

El uso de este paradigma posee algunas ventajas, las cuales sirvieron de apoyo a la hora de la realización de las entrevistas en los proyectos de la universidad.

- **Reutilización del software**. Nos lleva a alcanzar un mayor nivel de reutilización de software.

- **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. (autores 2000)

Aquí se muestra el resultado de cómo utilizan estas ventajas en los proyectos que trabajan con arquitectura basada en componente. Pues si de ventajas se trata la principal es la reutilización, ya que esta es una pieza de código preelaborado que encapsula alguna funcionalidad expuesta a través de interfaces estándar. Son los "ingredientes de las aplicaciones" que se juntan y combinan para llevar a cabo una tarea.

Todo esto hace que los ciclos de desarrollo sean más cortos. La adición de una pieza dada de funcionalidad tomará días en lugar de meses ó años. Funcionalidad mejorada. Para usar un componente que contenga una pieza de funcionalidad, solo se necesita entender su naturaleza.

Los 21 proyectos entrevistados emplean la reutilización como su principal beneficio, pues es el soporte de la arquitectura basada en componentes. De estos 21 proyectos que trabajan con arquitectura basada en componente ven como productividad solo el 71%, como aumento de la calidad el 38%, como abstracción del diseño 42%, como disminución de los costos el 33%, y el 9% ven otros beneficios. No se pude dejar de mencionar que la mayoría los proyectos tienen una combinación de todas estas ventajas ya que la unión de algunas de ellas le dan a conocer la utilidad que tienen para la culminación de la aplicación. A continuación se muestra una gráfica con estos resultados.

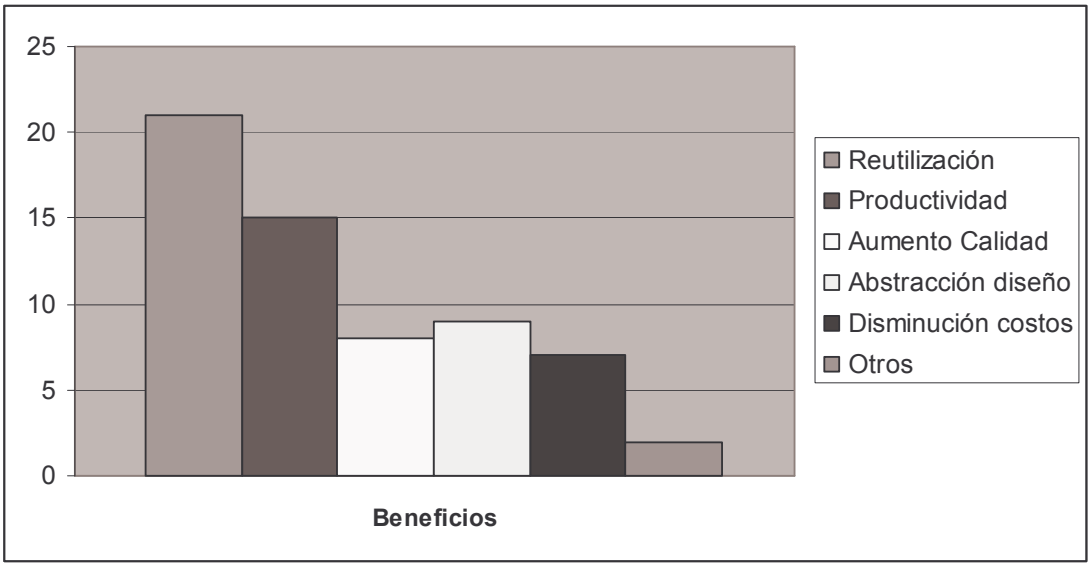

Fig. 2.5: Beneficios de la arquitectura basada en componente.

2.3.6 ¿Qué entienden los arquitectos de los proyectos entrevistados por componente?

En esta pregunta de la entrevista se pretendía conocer que entendían los líderes y arquitectos de los proyectos de la universidad por Componente, para ello se utilizó el concepto de componentes descrito en capítulo 1 del presente trabajo y que plantea:

- Es una parte reutilizable que encapsula elementos del modelo (por ejemplo una DLL, documentos, tablas, biblioteca, etc.).
- Es un paquete de software el cual ofrece servicios a través de su interfaces.
- Paquete de Software que puede ser usado para construir aplicaciones o componentes más grandes.

La elección de algunos como podrán ver en la gráfica que aparece a continuación, no fue la más ideal para todos aquellos arquitectos que trabajan con arquitectura basada en componente, pues

el 100% no marcó los 3 conceptos de manera general, pues cada cual tiene un concepto y lo ve hasta un punto de vista.

El 57% considera que un componente es una parte reutilizable que encapsula elementos del modelo (por ejemplo una DLL, documentos, tablas, biblioteca, etc.), el 42% lo ve como un paquete de software el cual ofrece servicios a través de su interfaces, y solo el 24% lo ve como un paquete de Software que puede ser usado para construir aplicaciones o componentes más grandes.

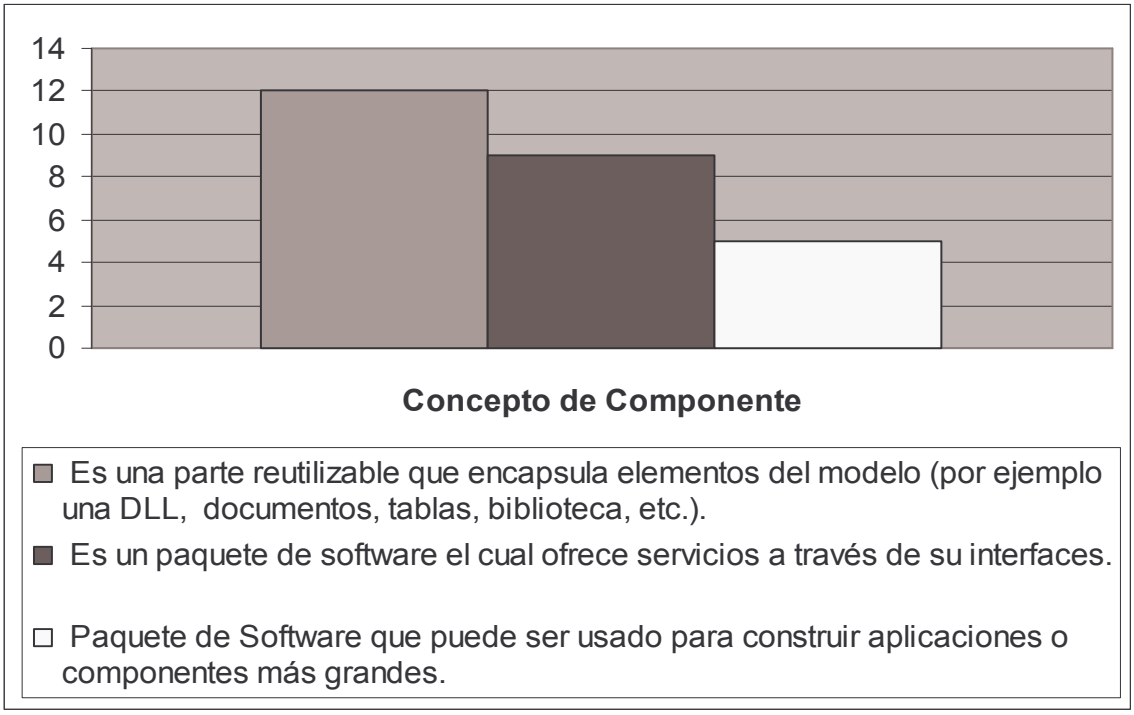

Fig.2.6: Concepto de Componente.

2.3.7 Ensamblaje de la arquitectura basada en componente.

En el estudio realizado se vio que hubo un 24% de los 21 proyectos que trabajan con arquitectura basada en componente que tienen un conocimiento errado de cómo realizar el ensamblaje de los componentes, pues este por ciento seleccionó que hacían el ensamblaje permitiendo el acceso a los elementos del componente, no concibiéndolo como se debe hacer que es mediante interfaces, pues estas son un aspecto fundamental en la composición de componentes y en la comunicación entre ellos, las mismas determinan la manera en que un componente se reutiliza.

Las interfaces definen una operación o un conjunto de operaciones llamadas servicios o responsabilidades, por lo general un componente produce (exporta) un resultado que es consumido (importado) por otro componente. (Pressman 2005)

Aproximadamente un 90% seleccionó el ensamblaje mediante interfaces, por lo que se puede observar, aunque sea la gran minoría, todavía hay personas que no tienen bien claro como se realiza el ensamblaje de los componentes.

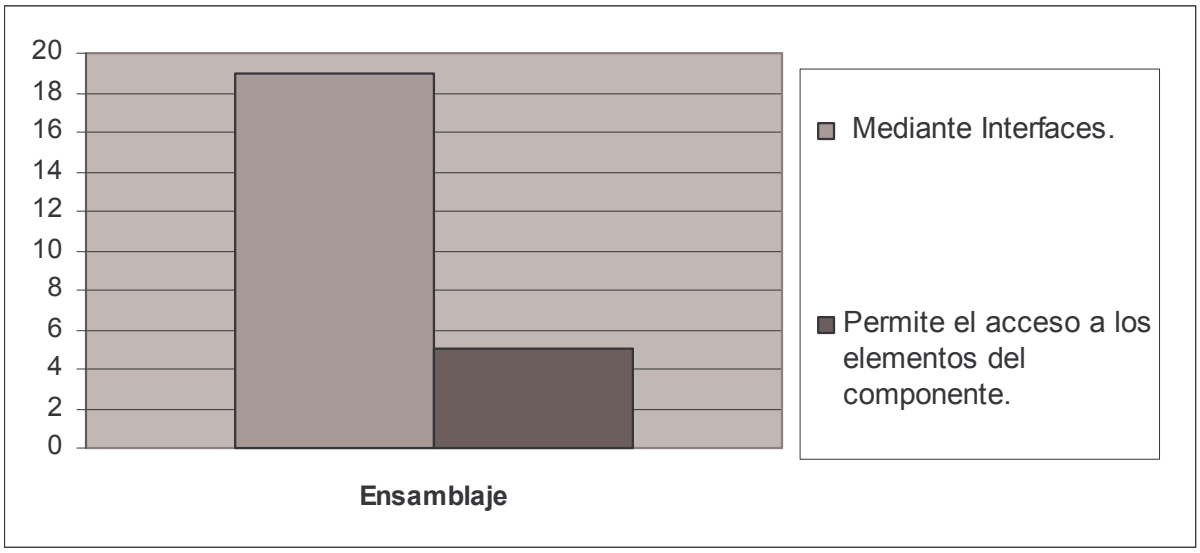

Fig.2.7: Ensamblaje de los componentes.

2.3.8 Perspectivas en el diseño de aplicaciones.

A medida que crece Internet y las tecnologías relacionadas, las organizaciones buscan integrar sus sistemas entre límites de departamentos y de organización, esto ha evolucionado un enfoque de generación de soluciones.

Las principales perspectivas para el uso de la arquitectura basada en componente son mediante paquetes, servicios e integración.

En el estudio realizado a los proyectos que trabajan con dicha arquitectura, se logró conocer que la perspectiva que más utilizan los arquitectos en el diseño de su aplicación es mediante paquetes, aunque algunos hacen una que otra combinación para así lograr un mayor progreso en su trabajo.

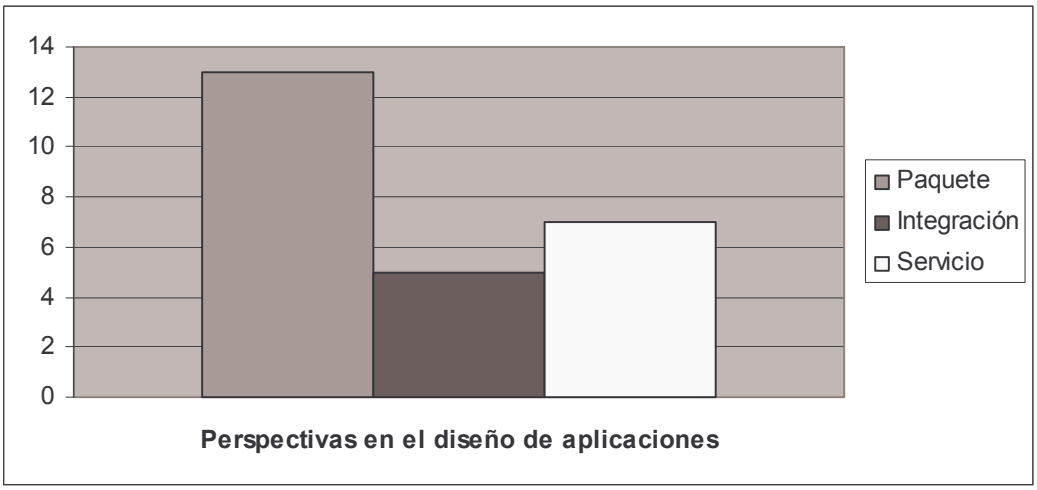

Fig.2.8: Perspectivas en el diseño de aplicaciones.

2.3.9 Guía de proceso

La última pregunta que se realizó en la entrevista fue para conocer si algunos de los arquitectos que trabajan en su proyecto con arquitectura basada en componentes tienen una guía, un proceso o una metodología a seguir cuando van a desarrollar el análisis y diseño de sus aplicaciones, siendo esta la pregunta más importante para la culminación de esta investigación. El 100% de los entrevistados no posee dicha guía, puesto que trabajan según los conocimientos que cada cual posee y sin seguir ningún proceso.

2.4 Conclusiones

Al finalizar este capítulo, se ha podido llegar a una serie de conclusiones, que servirán de apoyo para la culminación del presente trabajo.

Se pudo conocer que todos los proyectos que desarrollan aplicaciones con arquitectura basada en componentes, ven que el principal beneficio que proporciona este patrón arquitectónico es la reutilización, siendo para algunos un error decir que el ensamblaje de los componentes se puede hacer accediendo a los elementos del mismo y no mediante sus interfaces que es como debe ser.

Este estudio sirvió para detectar una serie de dificultades que se presentan a la hora de trabajar con este patrón de arquitectura. Muchos ni siquiera tienen definido claramente el concepto de componente.

Además en ninguno de los proyectos entrevistados se posee una guía para el análisis y diseño de las aplicaciones que utilizan este estilo arquitectónico. Al no contar con esta, corren el riesgo potencial de no tener éxito en el proyecto, siendo menos eficientes y económicos, ya que se agotan más recursos humanos, técnicos y financieros.

CAPÍTULO

GUÍA PARA LA REALIZACIÓN DEL

ANÁLISIS Y DISEÑO DE APLICACIONES CON

ARQUITECTURA BASADA EN COMPONENTES

3.1 Introducción

El análisis y diseño de aplicaciones informáticas, debe abordarse, con técnicas y metodologías adecuadas, más aún si la arquitectura con la que se está trabajando es basada en componentes. Ya que en la misma, se deben realizar un conjunto de actividades que deben quedar bien claras para los desarrolladores de software.

De ahí la importancia de poder contar con una guía, que explique paso por paso cada una de las actividades que se deben realizar en esta disciplina.

En el presente capítulo se propone una guía con todo el flujo de actividades que se deben seguir para realizar el análisis y diseño de las aplicaciones cuya arquitectura está basada en componentes.

3.2 Mapa de la guía

A continuación se muestra el flujo de actividades que se deben seguir para realizar el análisis y diseño de aplicaciones cuya arquitectura es basada en componentes:

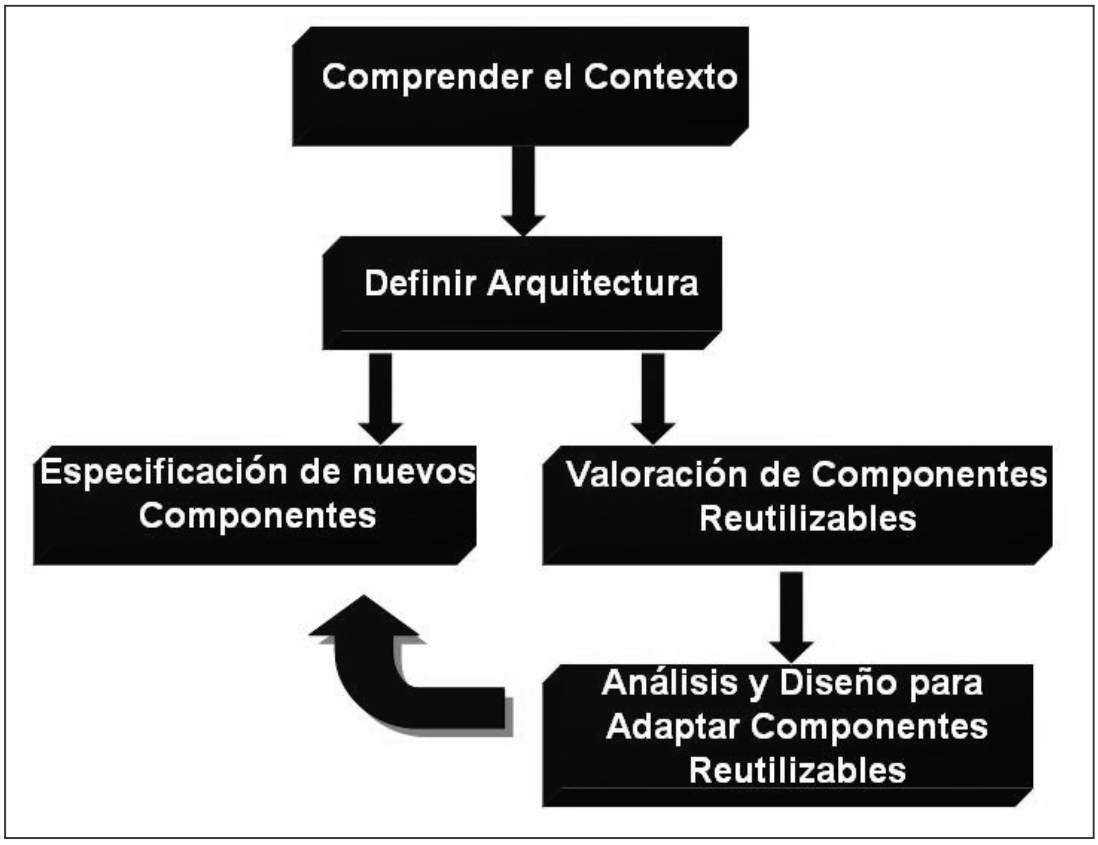

Fig. 3.1: Flujo de actividades a seguir para la realización del análisis y diseño

3.3 Descripción del proceso

Como se pudo observar en el capítulo 1 de esta tesis, existen tres variantes para realizar un proceso de desarrollo basado en componentes, estas son:

- Tomar la decisión de reutilizar los componentes existentes.
- Desarrollar componentes nuevos.
- Comprar componentes a un suministrador.

En el presente trabajo se definió una guía para la realización de un proceso de análisis y diseño basado en las dos primeras variantes, es decir, reutilizar componentes existentes o desarrollados desde sus inicios, no cubriendo el proceso de adquisición de un componente a un tercero.

El modelo que se propone, a diferencia de RUP, parte del análisis del dominio, que es modelado mediante un diagrama de clases. A partir de aquí, se definen los componentes y las clases que en el se implementarán para definir una arquitectura a un alto nivel, la cual se irá especificando y refinando en la medida que se evalúen los componentes a reutilizar y se diseñen los nuevos componentes.

Este proceso se realizará de forma iterativa e incremental, es decir, en la medida que se obtenga conocimiento del dominio y este pueda ser modelado, se irá definiendo y/o refinando la arquitectura realizándose la valoración y análisis de los componentes a reutilizar.

En la actividad “Comprender el contexto” es donde tienen mayor peso las tareas del análisis, aquí se debe entender y modelar el sistema actual y llegar a una comprensión común de todo el equipo de proyecto. Como resultado se obtiene un diagrama de clases con los atributos y relaciones.

En la actividad "Definir arquitectura", se toma el diagrama de clases y se comienza a esbozar un diagrama de componentes, tratando de analizar lo que pudiera ser reutilizado en otros proyectos de igual dominio, paralelamente se realiza una búsqueda en la biblioteca de componentes para investigar la existencia de componentes de igual dominio. Como resultado se obtiene un diagrama de componentes, con los componentes candidatos, sus interfaces y los mensajes entre ellos.

En la actividad "Valoración de componentes reutilizables" se toma el diagrama de clases y el diagrama de componentes, en los cuales se marcó los componentes a reutilizar. Se accede a la biblioteca de componentes para verificar si el componente a reutilizar cumple con los requisitos de calidad (requisitos no funcionales) definidos para el proyecto, si en sus interfaces expone las operaciones necesarias para la arquitectura del nuevo proyecto, su nivel de seguridad y manejo de errores. Si algún componente a reutilizar es rechazado entonces esté tendrá que ser desarrollado desde su inicio.

En la actividad "Análisis y diseño para adaptar componentes reutilizables" se accede a la biblioteca de componentes para extraerlo y chequear su código con el propósito de eliminar posibles conflictos por problemas de incompatibilidad con la nueva plataforma en la que será reutilizado el componente, para ello se podrán diseñar nuevas interfaces, agregar elementos al modelo, etc. Como resultado de esta actividad se actualiza el componente y se refina el diagrama de componentes.

En la actividad "Especificar nuevos componentes" se realizará el diseño del componente ya sea porque es nuevo o porque se rechazó un componente candidato a reutilizar. Como resultado de esta actividad se obtiene la especificación del componente en la cual se encuentra el diagrama de clases, los diagramas de interacción así como las interfaces y las operaciones que en ellas se encuentra. Al finalizar se refina el diagrama de componentes y se actualiza la biblioteca de componentes a la cual se le adiciona la nueva especificación.

3.4 Roles y responsabilidades

En el ciclo de vida de un software intervienen distintas personas, en el flujo de trabajo que se está estudiando en esta tesis los que intervienen son los arquitectos de software, diseñadores y analista del sistema. En la siguiente tabla se explican las responsabilidades que tiene cada cual.

Rol	Responsabilidad
Arquitecto de Software	Diseñar arquitectura de alto nivel. Realizar y refinar el diagrama de componentes.
Diseñador	Diseña arquitectura a bajo nivel. Diseñar componentes. Realizar para cada componente el diagrama de clases del diseño, diagramas de interacción y diseño de las interfaces.
Analista del sistema	Modelar negocio. Modelar sistema. Realizar el diagrama de clases del análisis.

Tabla 3.1: Roles y responsabilidades.

3.5 Descripción de las actividades de la guía para realizar el proceso

Para desarrollar la secuencia de actividades mostrada en la Fig. 3.1, los desarrolladores deben tener conocimiento del propósito que tiene cada una, los documentos entrantes y salientes, los

criterios de inicio y de fin, así como las subactividades que deben desarrollar. A continuación se realiza una descripción detallada de cada uno de estos procesos.

3.5.1 Descripción de la actividad: Comprender el Contexto.

Propósito de la actividad: Obtener un conocimiento del dominio a partir del sistema actual, los procesos del negocio y las necesidades de los usuarios.

Responsable de la actividad: Analista del sistema.

Documentos de entrada: A través de entrevistas, observación, estudio de la documentación que se crea en los diferentes procesos del negocio.

Documentos de salida: Modelo del dominio, el cual se realizará mediante un diagrama de clases en el que estará reflejado los objetos del negocio, sus atributos y relaciones.
Quedaran documentados las necesidades de los usuarios y los problemas existentes en el sistema actual.

Criterios de inicio: Cuando inicia un nuevo proyecto y se comienzan a realizar estudios del negocio.

Criterios de fin: Cuando se obtiene la comprensión del dominio mediante la aprobación con el cliente de los documentos de salida.

Subactividades:

- Realizar una descripción de alto nivel del comportamiento estático (diagrama de clases) y dinámico (Diagrama de interacción) del dominio, o realizar un modelo de dominio detallado: modelado de clases, modelado de casos de uso del negocio y modelado de colaboraciones (secuencia - colaboración).

3.5.2 Descripción de la actividad: Definir arquitectura.

Propósito de la actividad: Decidir a partir del modelo de dominio como ese comportamiento va a ser empaquetado en términos de componente.

Responsable de la actividad: Arquitecto de software, analista del sistema y diseñador.

Documentos de entrada: Modelo de dominio.

Documentos de salida: Diagrama de componente donde solo está establecida la comunicación entre ellos, es decir, su colaboración, por lo que contendrá los mensajes entre las interfaces de los componentes para solicitar una operación.

Criterios de inicio: Cuando se tenga el modelo de dominio y se comprenda.

Criterios de fin: Cuando la arquitectura este estable.

Subactividades:

- Identificar componentes a partir de los objetos del domino. Se debe prestar atención a lo que pudiera servir para la reutilización en otros proyectos de igual dominio.
- Analizar catálogo de componentes ya implementados que pueden ser reutilizados.
- Definir las interacciones de los componentes mediante la identificación de las operaciones en las interfaces y determinar las dependencias entre ellas así como los mensajes.
- Realizar una primera versión del diagrama de componentes.

3.5.3 Descripción de la actividad: Especificación de nuevos componentes.

Propósito de la actividad: Realizar la especificación de los nuevos componentes.

Responsable de la actividad: Analista del sistema y diseñador.

Documentos de entrada: Modelo de dominio y diagrama de componentes.

Documentos de salida:

- Diseño del componente. Para cada componente realizar su diagrama de clases del diseño y diagramas de interacción así como diseñar sus interfaces definiéndole las operaciones, es decir realizar la especificación del componente.
- Refinar el diagrama de componentes.
- Catálogo de componentes actualizado.
- Biblioteca de componentes.

Criterios de inicio: Cuando se tiene la decisión de desarrollar un componente.

Criterios de fin: Cuando se encuentren diseñados todos los nuevos componentes y actualizado el catálogo de componente.

Subactividades:

- Diseño de componente:
    a. A partir del modelo de dominio, determinar las clases que se implementarán en el componente
    b. Realizar el diagrama de clases del diseño del componente. Aquí se agregan las clases necesarias para la plataforma en que se implementará el componente y se asignarán responsabilidades aplicando los patrones de diseño que sean necesarios.
    c. Diseño de las interfaces del componente.
- Actualizar el catálogo de componentes.
- Refinar el diseño de las interacciones entre los componentes en el diagrama de componentes

3.5.4 Descripción de la actividad: Valoración de componentes reutilizables.

Propósito de la actividad: Investigar y/o analizar el ajuste del componente a reutilizar en la arquitectura propuesta.

Responsable de la actividad: Arquitecto de software y diseñador.

Documentos de entrada:

- Diagrama de componentes con la lista de componentes a reutilizar.
- Modelo de dominio.
- Catálogo de componentes.
- Biblioteca de componentes.

Documentos de salida:
Valoración del componente con su aceptación o no para ser reutilizado.

Criterios de inicio: Cuando se conoce que será reutilizado un componente.

Criterios de fin: Cuando se valoren todos los componentes candidatos a reutilizar.

Subactividades:

- Analizar las interfaces del componente para chequear que brinda las operaciones necesarias de acuerdo a los requerimientos del nuevo proyecto.
- Analizar los requerimientos de calidad: rendimiento, fiabilidad, usabilidad y portabilidad, de forma tal que estos cumplan con los requisitos no funcionales del nuevo proyecto.
- Chequear el nivel de seguridad.
- Verificar el manejo de errores.

3.5.5 Descripción de la actividad: Análisis y diseño para adaptar componentes reutilizables.

Propósito de la actividad: Detectar y eliminar posibles conflictos en el código fuente del componente que provoca su no compatibilidad con la arquitectura propuesta.

Responsable de la actividad: Arquitecto y diseñador de software.

Documentos de entrada:

- Modelo de dominio.
- Diagrama de componentes.
- Catálogo de componentes.

Documentos de salida:

- Solicitud de modificación del componente. Aquí se realiza la especificación de la modificación.

Criterios de inicio:

Cuando existe un componente que se valoró como positiva su reutilización.

Criterios de fin:

Cuando se realizaron las pruebas de integración del componente en la arquitectura que dió como resultado su reutilización con o sin modificaciones o cuando se decide no reutilizarlo

Subactividades:

- Examinar el comportamiento interno del componente.

- En caso de que existan incompatibilidades con la nueva arquitectura en la que el componente será reutilizado, estimar el impacto y el esfuerzo que implica los cambios para decidir o no su reutilización.

3.6 Conclusiones

Con la guía presentada en este capítulo los ingenieros de software podrán realizar un mejor trabajo en la confección de aplicaciones que trabajan con arquitectura basada en componente. En la misma se explican todos los pasos que se deben seguir para realizar el análisis y diseño de aplicaciones.

CONCLUSIONES

En este trabajo se estudió la arquitectura basada en componentes, además, se analizaron los proyectos productivos más importantes de la universidad para ver si las personas que trabajan con esta arquitectura tenían los conocimientos necesarios para trabajar con este patrón arquitectónico.

Por último se obtuvo una guía para la realización del análisis y diseño de las aplicaciones con arquitectura basada en componentes, siendo esta el principal objetivo del presente trabajo.

En la misma se aplican los resultados de la investigación llevada a cabo. Esta guía favorecerá el incremento de la eficiencia en las aplicaciones que se realizan en los proyectos productivos de la universidad.

RECOMENDACIONES

De forma general todos los objetivos de este trabajo fueron cumplidos. Queda una documentación bastante amplia de las experiencias acumuladas en este tiempo de trabajo con los resultados de todo el proceso de investigación desarrollado para cumplir los objetivos, que permitieron mejorar las prácticas en el desarrollo de software hasta este momento. Por tanto se hacen las siguientes recomendaciones:

- Según el resultado obtenido en las entrevistas, se debe mejorar la preparación de los integrantes del equipo de desarrollo en los temas de Ingeniería del software.

- Profundizar el estudio sobre el desarrollo basado en componentes en los proyectos productivos que utilicen este estilo.

- Escoger un proyecto productivo de la UCI para que trabaje con esta guía propuesta, lo que permitirá comprobar su utilidad y realizar las adecuaciones necesarias.

REFERENCIAS BIBLIOGRÁFICAS

Autores, c. d. (2000) About Component-Based Development. Disponible en: http://webcabcomponents.com/componentization.shtml

Autores, C. d. (2001) Introducción a las aplicaciones de gestión. Disponible en: http://sestud.uv.es/manual.esp/gestion/gestion0.htm

Autores, c. d. (2005). Trucos InfoPath 2003. Disponible en: http://www.microsoft.com/spain/office/trucos/infopath/t02.mspx

González, R. (2005) Apéndice TECNICAS DE ENTREVISTA. Disponible en: http://lsi.ugr.es/~ig1/docis/entre1.html

Institute, S. E. (2007) Published Software Architecture Definitions. Disponible en: http://www.sei.cmu.edu/architecture/published_definitions.html#Modern

Jara, O. H. (2004) Evolución y Orientaciones de Patrones. Disponible en: www.monografias.com

Letelier, P. and M. C. Penadés (2002) Métodologías ágiles para el desarrollo de software:eXtreme Programming (XP).

Manuel F. Bertoa, J. M. T. y. A. V. (2002) Aspectos de Calidad en el Desarrollo de Software Basado en Componentes.

Martínez, L. F. I. (2003). Un Modelo de Mediación para el Desarrollo de Software basado en Componentes COTS.

Pressman (2005). Ingeniería del Software. Un Enfoque Práctico.

Reynoso, C. and N. Kicillof (2004) Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. Disponible en:
http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/style.asp#13

Software, S. (2000). Component Based Development and Object Modeling.

Vignaga, A. and D. Perovich (2005) Enfoque Metodológico para el Desarrollo Basado en Componentes. Disponible en:
http://www.fing.edu.uy/inco

BIBLIOGRAFÍA

Abdurazik, A. (2000). Suitability of the UML as an Architecture Description Language with applications to testing.

Albin, S. (2003). "The Art of Software Architecture: Design methods and techniques."

Amor, M., L. Fuentes, et al. (2002) Interoperabilidad entre Plataformas de Agentes FIPA: Una Aproximación Basada en Componentes*.

Bachmann, F., L. Bass, et al. (2000). The Architecture Based Design Method.

Bollinger, T. and C. McGowen (1991). A Critical Look at Software Capability Evaluations.

Cantor, M. (2003) Thoughts on Functional Decomposition. Disponible en: http://www.therationaledge.com/content/jun_03/t_subsystem_ff.jsp

Casanovas, J. (2004) Usabilidad y arquitectura del software.

Ericsson, M. (2000). Developing Large-scale Systems with the Rational Unified Process.

Ferm, F. (2003) The what, why, and how of a subsystem. Disponible en: http://www.therationaledge.com/content/jun_03/t_subsystem_ff.jsp

Fuentes, L., J. M. Troya, et al. (2000) Desarrollo de Software Basado en Componentes.

González, A., M. Mijares, et al. (2005) Método de Evaluación de Arquitecturas de Software Basadas en Componentes.

González, R. and M. Torres (2005). "Hacia un Entendimiento del Desarrollo Basado en Componentes."

G. BOOCH, JACOBSON. I (2004). El Proceso Unificado de Desarrollo de Software.

Kruchten, P. (2001) The Nature of Software: What's So Special About Software Engineering?

M, P. (2000). Capability Maturity Model for Sotfware.

Manuel, J. (2004) Tipos de componentes.

NOTARIO, Á. (2004). Investigación científica en las instituciones de Educación Superior.

LARMAN, C. (2004). UML y Patrones. Introducción al análisis y diseño orientado a objetos.

Pekka Abrahamsson, Outi Salo, et al. (2002). Agile Software Development Methods.

Pérez, M., L. Mendoza, et al. (2004) Evaluación Arquitectónica de un Software Basado en Componentes.

R., K., C. P, et al. (2002). Evaluating Software Architectures: Methods and Case Studies.

PRESSMAN, R. (2005). Ingeniería del Software. Un enfoque práctico.

Reynoso, C. B. (2004) Introducción a la Arquitectura de Software

ROSANIGO, Z. B. and D. D. G. ROSSI (2000). Maximizando reuso en software para Ingeniería Estructural.

Softel (2006) Documento sobre la Arquitectura de Software para los componentes a emplear por el Sistema de Información para la Salud.

Terreros, J. C. (2003) Desarrollo de Software basado en Componentes

Webster, G. A. (2003). A Technical How-to Guide for Creating Components and Web Services in Rational Rapid Developer.

Wesley, A. (2002). Documenting Software Architectures: Views and Beyond.

ANEXOS

Entrevista realizada a arquitectos de proyectos productivos de la UCI vinculados al desarrollo de aplicaciones.

Tema: Arquitectura Basada en Componentes.

Nombre del Proyecto: _____

Nombre del Arquitecto: **_____**

1-Tamaño del Proyecto:

a) ____Grande

b) ____Mediano

c) ____Pequeño

2.- Tipo de aplicación que desarrollan:

a) ____Gestión

b) ____Procesamiento de imágenes

c) ____Web

d) ____Otro, diga cual_____

3- Metodología que utiliza:

a) ____RUP

b) ____XP

c) ____MDA

d) ____Otra, diga cual_____

4-Señale el o los patrones de arquitectura de software que utiliza en su proyecto:

a) ____Model-View-Controller (MVC)
b) ____Arquitecturas en Capas
c) ____Arquitecturas Orientadas a Objetos
d) ____Arquitecturas Basadas en Componentes
e) ____Arquitectura orientada a Servicios (SOA)
f) ____Ninguno
g) ____Otro, diga cual_____

5- La arquitectura basada en componentes brinda grandes beneficios, seleccione cuales de ellos le hicieron a usted seleccionar este tipo de patrón para realizar el diseño de su aplicación.

a) ____Reutilización
b) ____Productividad (aumento)
c) ____Aumento de la calidad
d) ____Abstracción en el diseño
e) ____Disminución de los costos
f) ____Otro, diga cual_____

6- Pudiera señalar que es para usted un componente :

a) ____Es una parte reutilizable en encapsula elementos del modelo (por ejemplo una DLL, documentos, tablas, biblioteca, etc.).
b) ____Es un paquete de software el cual ofrece servicios a través de su interfaces.

c) ____Paquete de Software que puede ser usado para construir aplicaciones o componentes más grandes.

7- En caso de utilizar la arquitectura basada en componentes, como realiza el ensamblaje de los mismos.

a) ____Mediante interfaces
b) ____Permite el acceso a los elementos del componente
c) ____Otros_____

8- La arquitectura basada en componentes posee varias perspectivas para su utilización, seleccione cual es la que usted utiliza en el diseño de su aplicación.

a) ____Paquete
b) ____Integración
c) ____Servicio
d) ____Ninguno
e) ____Otro, diga cual_____

9- Conoce o tiene definido un método para la realización del análisis y diseño de los componentes.

____No
____Sí, diga cual: _____

GLOSARIO

Búsqueda trading de componentes: Es una búsqueda de componentes comerciales.

Componentsource: Es una fuente de componentes.

COTS (Commercial-Off-The-Shelf): Es una clase especial de componentes software. Los componentes COTS son componentes comerciales.

COTStrader: Son componentes comerciales.

DLL: Sigla en inglés de Dynamic Linking Library. Es un archivo que contiene funciones que se pueden llamar desde aplicaciones u otras DLLs. Los desarrolladores utilizan DLLs para poder reciclar el código y aislar las diferentes tareas. Las DLLs no pueden ejecutarse directamente, es necesario llamarlas desde un código externo.

DSBC (Desarrollo de Software Basado en Componentes): Permite crear sistemas de software utilizando componentes.

EJB, CORBA y .NET: Son Plataformas de objetos distribuidos.

Flashline: Proporciona productos de componente de software, servicios y recursos que facilitan el desarrollo rápido de sistemas de software para el negocio, proporcionan la primera solución de reutilización de componente de software.

IDLs Siglas de Interface Definition Language ("Lenguaje de definición de interfaces" en inglés).

Existen distintos tipos de IDLs, por ejemplo:
CORBA IDL: lenguaje de definición de Interfaz usada en contornos CORBA y estandarizada por la OMG.

MMORPGS: Los juegos de rol multijugador masivo online o MMORPGs (Massive(ly) Multiplayer Online Role-Playing Games) son videojuegos que permiten a miles de jugadores introducirse en un mundo virtual de forma simultánea a través de Internet, e interactuar entre ellos.

RM-ODP: Sigla en inglés de Reference Model – Open Distributed Processing (Modelo de Referencia de ISO para el Procesamiento Abierto y Distribuido).
Provee la coordinación de los framework, creando una infraestructura dentro de la cual el soporte de distribución y la portabilidad puede ser integrada.

ROI: Sigla en inglés de Return Of Investment. Suele utilizarse para analizar la viabilidad de un proyecto y medir su éxito. En épocas de crisis, se convierte en fundamental que cada céntimo invertido en tecnología regrese, a ser posible, acompañado de más. En general, la usabilidad es la estrategia aplicada al desarrollo de proyectos que más ROI genera.

ROI= (Beneficios/Costos) x100

RUP: Sigla en inglés de Rational Unified Process. Es una metodología de desarrollo que está preparada para desarrollar grandes y complejos proyectos, puesto que es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo a través del UML, y trabajo de muchas metodologías utilizadas por los clientes.

UML: Por sus siglas en inglés, Unified Modeling Language es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un

sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Es importante remarcar que UML es un "lenguaje" para especificar y no un método o un proceso, se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. Es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para soportar una metodología de desarrollo de software (tal como el Proceso Unificado de Rational) UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas.

Webmails: Es un programa informático, concretamente un cliente de correo electrónico, que provee una interfaz web por la que acceder al correo electrónico.

El webmail permite listar, desplegar y borrar los correos almacenados en el servidor remoto. Los correos pueden ser consultados posteriormente desde otro computador conectado a la misma red (por ejemplo Internet) y que disponga de un navegador web.

Weblogs: Un weblog (también llamado blog) es un diario personal online o, si se quiere, un cuaderno de bitácora que un administrador provee constantemente y de forma cronológica de contenidos muy específicos. En los weblogs, se cuentan experiencias íntimas, se comparten noticias, se expresan opiniones, etc.

Wikis: Es un sitio web colaborativo que puede ser editado por varios usuarios. Los usuarios de una wiki pueden así crear, editar, borrar o modificar el contenido de una página web, de una forma interactiva, fácil y rápida; dichas

facilidades hacen de una wiki una herramienta efectiva para la escritura colaborativa.

La tecnología wiki permite que páginas web alojadas en un servidor público (las páginas wiki) sean escritas de forma colaborativa a través de un navegador, utilizando una notación sencilla para dar formato, crear enlaces, etc, conservando un historial de cambios que permite recuperar fácilmente cualquier estado anterior de la página. Cuando alguien edita una página wiki, sus cambios aparecen inmediatamente en la web, sin pasar por ningún tipo de revisión previa

XML: sigla en inglés de Extensible Markup Language (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas y permite definir la gramática de lenguajes específicos. XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

XP: La metodología XP es exitosa porque enfatiza la satisfacción del cliente y promueve el trabajo en equipo.

En XP, las actividades improductivas han sido eliminadas para reducir costos y frustraciones.

Esta metodología ha sido diseñada para solucionar el eterno problema del desarrollo de software por encargo: entregar el resultado que el cliente necesita a tiempo.

El desarrollo bajo XP tiene características que lo distinguen claramente de otras metodologías:
Los diseñadores y programadores se comunican efectivamente con el cliente y entre ellos mismos.